А.Г. КОРОЛЕВ

МОДЕЛИРОВАНИЕ СИСТЕМ СРЕДСТВАМИ OBJECT GPSS

ПРАКТИЧЕСКИЙ ПОДХОД В ПРИМЕРАХ И ЗАДАЧАХ

Северодонецк – 2008

УДК 62. 529

А.Г. Королёв Моделирование систем средствами Object GPSS. Практический подход в примерах и задачах. Учебное пособие. Библиогр.14

ISBN

Рассматриваются методы и техника моделирования систем массового обслуживания, основанные на дискретно- событийном подходе. Изложение ведется на основе средств Object GPSS. Изложение сопровождается большим количеством примеров.

Учебное пособие предназначено для студентов старших курсов по специальностям «Компьютерная инженерия», «Компьютерные системы и сети», «АСУ ТП», а также может быть использовано специалистами по технологиям обучения и инженерно-техническими работниками проектных организаций.

Оглавление

1 Введение

Имитационное моделирование, как и моделирование вообще – мощное средство для изучения сложных систем. [1..14]. Для имитационного моделирования используется целый спектр языков, однако наибольшую популярность среди специалистов приобрел язык GPSS. Он подробно описан, например, в [4,5,8] . Но наибольшей популярностью до сих пор пользуется книга Шрайбера Т. Дж. Моделирование на GPSS [14], которая также известна, как «красная книга». Эта книга является фундаментальным самоучителем по языку GPSS и прекрасно описывает как проблематику задач моделирования систем массового обслуживания, так и методы их решения с помощью данного языка.

В последнее время, в связи с появлением новой версии языка GPSS, Object GPSS, возникла настоятельная потребность в подготовке нового учебного пособия, описывающего особенности этой версии, и ее среды, обеспечивающей разработку моделей на GPSS. В этой книге собран многолетний опыт преподавания языка GPSS студентам. Она содержит большое количество примеров, иллюстрирующих особенности работы с оболочкой языка, а также особенности применения различных блоков новой версии для решения конкретных задач событийного моделирования.

Данное пособие предназначено как для студентов, изучающих компьютерное моделирование, так и для специалистов, использующих моделирование в практической работе.

Язык GPSS имеет довольно длинную историю, и пережил многие другие языки, которые разрабатывались для общения специалистов с ЭВМ. В первую очередь, упомянем, что этот язык предназначен для моделирования систем массового обслуживания, и, несмотря на многочисленные попытки сделать лучшие языки, выжил. Он прошел ряд процессов модернизации, и по-прежнему является наиболее популярным языком моделирования систем.

Концептуально, модель, положенная в основу языка, представляет собой следующее.

Имеется система, состоящая из одноканальных и многоканальных устройств, которые осуществляют обслуживание заявок (транзакций). Заявки перемещаются по системе, занимая эти устройства. Как правило, они задерживаются в устройствах на обслуживание. Траектория движения заявок по системе может быть разной, в зависимости от вида заявок, и от значений случайных и расчетных параметров, которые определяются заявками в процессе их движения по системе. В принципе, заявки вводятся в систему специальными блоками через заданные промежутки времени, которые могут быть и случайными. Заявки рано или поздно должны быть удалены из системы.

Этот язык имеет все основные вычислительные возможности традиционных языков программирования, но в отличие от них, вычисления здесь идут не в порядке расположения операторов, а в том порядке, в котором заявки движутся через блоки языка. Все операции происходят в так называемом модельном времени, и язык хорошо приспособлен для моделирования процессов, проходящих одновременно. Заявки конкурируют между собой за ресурсы системы, и в первую очередь за устройства. Поэтому в системе возникают очереди, из заявок, претендующих на один и тот же ресурс.

Если оценивать язык GPSS в целом, то можно сказать, что GPSS - это больше, чем язык программирования для имитационного моделирования систем массового обслуживания. Это также необычное явление в области программирования. Он появился в декабре 1961 года и был одним из самых удачных на то время проблемно-ориентированных языков.

Основой для создания GPSS послужил дискретно-событийный подход, разработанный Джеффри Гордоном. В своё время, GPSS произвёл настоящую революцию в области исследования систем массового обслуживания. Специалист в этой области, мог провести исследование работы сложнейшей системы в течение многих лет её реального функционирования, за считанные минуты и часы работы модели. Это было настолько быстро и необычно, что у специалистов в этой области как бы выросли крылья. И даже некоторая жесткость конструкций и недостаточная гибкость языка, не меняли общего восторженного отношения к GPSS .

Собственно систему общецелевого моделирования (General Purpose Systems Simulator) впервые также создал Джеффри Гордон (Geoffrey Gordon) в 1961 году. Тогда она была реализована на больших ЭВМ. В первой версии было сравнительно немного блоков, всего чуть больше 20.Уже в первой версии GPSS было два типа обслуживающих аппаратов, «Устройства», обслуживающие не более, чем одну заявку в каждый момент времени и «Памяти» (STORE) (переименованный в следующих версиях на STORAGE), которые могли обслуживать несколько заявок одновременно. Устройства могли быть

заняты и освобождены блоками SEIZE и RELEASE, а памяти - соответственно блоками ENTER и LEAVE . Дальнейшее развитие системы шло за счёт расширения номенклатуры блоков, и за счет удаления тех блоков, которые оказывались ненужными в новых версиях.

Ситуация более или менее стабилизировалась к моменту появления версии GPSS III. Эта версия системы была уже достаточно близкой к системе GPSS World 2000. После этого, GPSS стал более языком моделирования, чем системным имитатором. Последующие версии GPSS развивались вначале на базе ЭВМ типа IBM/360 и ЕС ЭВМ, а затем, с появлением в начале 80-х годов персональных компьютеров, новые версии стали создаваться и для них. Здесь ключевым этапом оказалось появление в 1984 году новой версии GPSS - GPSS /PC, разработанной фирмой Minuteman Software под руководством С. Кокса. Система GPSS /PC – это в основном не компилятор, а интерпретатор. Эта система работала под управлением MS-DOS, и когда возможности эксплуатации MS-DOS оказались исчерпанными, в 2000 году фирмой Minuteman Software была выпущена версия GPSS World 2000.

Однако на этом история GPSS не заканчивается. В процессе эксплуатации GPSS World выяснилось, что эта система недостаточно полно использует возможности операционной системы Windows. Многие её особенности, сохранившиеся от прошлого, не способствуют освоению GPSS новым поколением специалистов, а скорее мешают им при разработке моделей. Короче говоря, возникла потребность совместить возможности языка GPSS – с возможностями одного из современных языков программирования класса 4GL. Только такое совмещение могло придать новый импульс развития языку GPSS.

В рамках этой идеи, было предложено средство, позволяющее описывать модели систем массового обслуживания прямо на Object Pascal, но в стиле языка GPSS. В этом случае модель системы пишется как набор процедур на Object Pascal, обеспечивающих моделирование системы. Сама система поддержки процесса моделирования также написана на Object Pascal, а точнее, на Delphi. Такой подход естественным образом обеспечивает модели все те возможности, которые есть в базовом языке.

На первый взгляд, модель системы, написанная как набор процедур на Object Pascal, должна быть громоздкой и сложной. Однако это не так. Большую часть текста модели можно получить автоматическими средствами, а собственно содержательная часть модели обычно оказывается небольшой, и по объему ненамного превосходит текст модели на GPSS – World или GPSS/h .

В целом, возможности системы Object GPSS намного превосходят возможности других версий GPSS. Она является более стройной, содержит намного большее число блоков и процедур, и может быть легко развита за счет пользовательских процедур.

Пользование системой и разработка новых моделей не предполагает знания языка Object Pascal, а требует знания только основ программирования на любом языке высокого уровня.

Необходимые минимальные сведения об Object Pascal излагаются непосредственно при описании системы Object GPSS и составляют весьма небольшую часть объема пособия.

Изложению особенностей системы дискретно-событийного моделирования - Object GPSS и посвящена данная книга.

2 Достоинства Object GPSS в сравнении с другими языками моделирования

Новый продукт, Object GPSS содержит много новых функций и процедур, расширяющих возможности разработчика модели. Общее число блоков в системе доведено до 142. Система содержит 11 типов объектов и 173 процедуры для управления моделью. Общее число функций достигает 150.

Для работы с системой используется программа – конвертер, которая позволяет достаточно просто создавать, компилировать и запускать модели на выполнение. Скомпилированная модель и обеспечивает собственно моделирование.

Как программа – конвертер, так и скомпилированная модель имеют современный внешний вид, соответствующий возможностям ОС Windows и достаточно удобны как для построения моделей, так и для отображения результатов моделирования в текстовом или в графическом виде. В системе значительно облегчена отладка моделей, так как всегда можно наблюдать графики изменения нужных параметров модели, а также получать подробные «снимки» текущего состояния модели.

Функции и параметры модели – типизированы, и могут быть следующих базовых типов: целый, вещественный, строковый и логический.

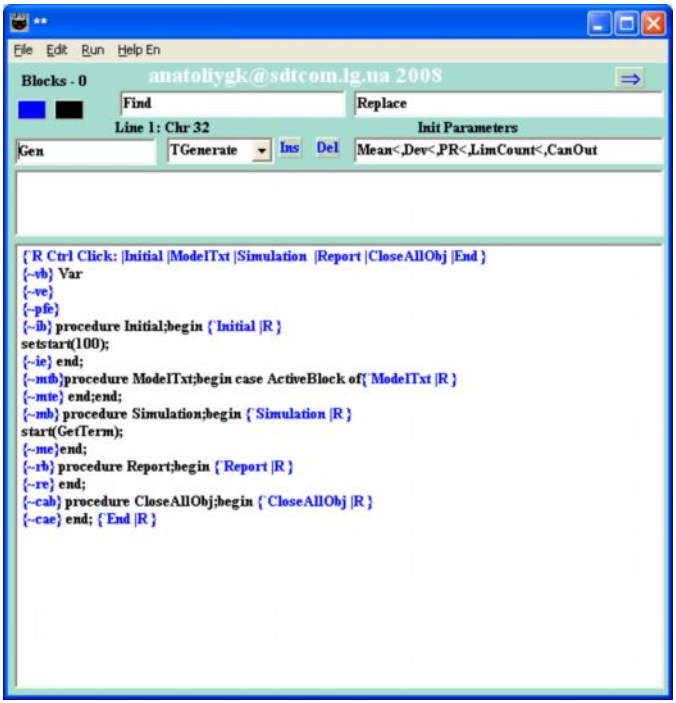
При моделировании можно использовать данные и процедуры на языке Object Pascal, которые подготавливаются разработчиком модели, если ему это нужно. Таким образом, можно упростить разработку сложных моделей, в том числе и таких, которые невозможно либо сложно реализовать на классическом GPSS. В системе можно программно управлять не только вычислениями, но и выводом результатов, но также управлять моделью как единым целым.

Система Object GPSS, ориентирована на дискретно- событийное моделирование систем массового обслуживания. Представление жизни модели как движения во времени заявок, перемещающихся в модели и обслуживающихся в устройствах очень естественно для многих задач имитационного моделирования. Система Object GPSS, достаточно легка для изучения. Студенты после короткого времени обучения могут создавать достаточно сложные модели. Автоматический сбор статистики - это огромная помощь для начинающих. Для многих реальных систем, моделирование на Object GPSS, выполняется гораздо легче, чем другими методами. Компактность текста модели и возможность использования графического интерфейса позволяют ускорить создание прототипов моделей. При этом на каждую из них можно получить быстрый отклик после улучшений проведенных пользователем.

3 Работа с системой Object GPSS

Подготовка модели ведется с помощью программы – конвертера, которая имеет оболочку, достаточно традиционную для приложений WINDOWS.

Внешний вид этой программы приводится ниже.



Основные возможности по управлению его работой предоставляет главное меню и дублирующие его возможности, горячие клавиши. Главное меню содержит следующие подменю.

File Edit Run Help.

Подменю File .

Это подменю содержит следующие пункты.

New

создать новую модель.

Open

открыть ранее созданную модель, или любой другой текстовый файл в формате TXT или RTF.

Save

сохранить новую версию модели в файле на диске.

Save As

сохранить модель в файле на диске, изменив имя и, возможно, папку в формате TXT или RTF.

Quit

закрытие программы – конвертера.

Все эти пункты достаточно традиционны для оконных приложений, и вряд ли нуждаются в особых пояснениях.

Подменю Edit

Это подменю содержит следующие пункты.

Insert Object

Позволяет вставить описание объекта, используемого в модели, а также вставить вызовы всех процедур, которые необходимы для корректной работы модели с этим объектом. Имя объекта задается в соответствующем поле, или формируется автоматически, а тип объекта выбирается из выпадающего списка.

Delete Object

Позволяет удалить описание объекта, используемого в модели, а также удалить вызовы всех процедур, которые были вставлены для корректной работы модели с этим объектом. Для удаления – следует вначале поместить курсор на строку описания объекта, или на строку со сгенерированным вызовом для объекта.

Find

Позволяет искать заданную подстроку. Регистр символов при поиске значения не имеет.

Replace

Позволяет заменить найденную подстроку на заданную подстроку и продолжить поиск заданной подстроки. Регистр символов при поиске значения не имеет.

Replace One Name

Заменяет имя, заданное в поле поиска, на имя заданное в поле замены. Имя для поиска не должно содержать разделителей, так как иначе этот пункт работать не будет.

Replace Names

Работает аналогично но пары имен должны задаваться в файле OldNewNames.txt в последовательно идущих парах строк.

FontSelected

вызов диалога изменения шрифта выделенной части модели.

Include

Union

Используются для моделей, содержащих имена файлов включения и соединения, смотри Приложение.ы

ReOpen

Читает заново файл текущей модели с диска.

Run.

выполняет конвертирование текста модели в набор процедур модели, то есть создает файл Model.pas, а также создает модель в виде EXE – файла и запускает ее на выполнение, если, конечно, не было ошибок компиляции. Иначе нужно исправить эти ошибки и повторить пункт Run. Кроме того, пункт сохраняет текущее состояние файла с текстом модели и помещает текст, сформированный компилятором, с возможными сообщениями об ошибках, в соответствующее поле конвертера. Этот текст позволяет найти и исправить формальные ошибки.

При перемещении по тексту модели, отслеживается номер строки, где находится курсор и для текста модели, и для файла Model.pas. Эта информация полезна при отладке синтаксических ошибок в модели. Сообщения об ошибках формирует компилятор с Object Pascal при выполнении пункта Run.

Подменю Help.

показывает файл справки. Справка доступна на Английском или Русском языке. По файлу справки можно перемещаться щелкая по синим гиперссылкам при нажатой клавише Ctrl, или выполнять поиск по кнопке Find.

Гиперссылка начинается с символа | и следует до пробела , а точка перехода – начинается с символа ` и тоже должна заканчиваться пробелом. По тексту модели также можно выполнять переходы по гиперссылкам. Но здесь гиперссылки и точки перехода должны находиться в комментариях, чтобы не мешать компиляции модели.

Синяя кнопка выделяет комментарии синим цветом, а текст самой модели – зеленым. Черная кнопка делает весть текст черным.

Дополнительное правое поле содержит имена всех переменных модели, а также полный перечень всех шаблонов процедур и функций системы моделирования.

Если имена переменных не соответствуют модели, то нужно закрыть и открыть это поле. Тогда они появятся все. В этом поле также можно перемещаться по гиперссылкам. Если кнопка над полем указывает влево, и вы держите нажатой клавишу Ctrl, то при щелчке по шаблону процедуры либо функции он переносится в соответствующую позицию модели. Если эта кнопка указывает вверх, и вы держите нажатой клавишу Ctrl, то шаблоны переносятся в верхнюю часть самого поля. То есть, можно вначале собрать все нужные для модели шаблоны вверху, а эатем их вставлять в нужных местах. Можно, правда и просто пользоваться буфером обмена.

4 Общие сведения о системе Object GPSS

Object GPSS – это высоко интегрированная компьютерная среда моделирования общего назначения, разработанная для профессионалов моделирования. Это - мощный инструмент моделирования, покрывающий, дискретно-событийное моделирование, с чрезвычайно высоким уровнем взаимодействия и визуализации. При использовании Object GPSS, возможно предсказать поведение весьма сложных реальных систем.

Многие дорогостоящие проекты в прошлом потерпели неудачу, потому что конечный результат не был предсказан достаточно точно. Определение максимальной пропускной способности системы, ее стоимости, и многого другого, всё это необходимо детально знать о системе при её разработке, причём как можно раньше. Хотя хорошие математические модели чрезвычайно ценны, и они должны использоваться там, где это возможно, сложность реальных систем требует использования компьютерного моделирования. В этих случаях и необходима система Object GPSS.

Эта версия, Object GPSS, - прямой потомок GPSS /PC и GPSS World для персональных компьютеров. Введение в 1984 году, GPSS /PC, а в 2000 году и GPSS World сохранило тысячам пользователей миллионы долларов. Теперь, система Object GPSS расширяет их возможности.

Язык Object GPSS разработан так, чтобы давать ответы быстро и надежно, с минимумом усилий.

Система Object GPSS была разработана таким образом, чтобы обеспечить прозрачность моделирования. Она позволяет видеть внутренние механизмы моделей и зафиксировать результат. Взаимодействие модели с пользователем позволяет ему не только провести исследование, но и обеспечивает управление моделью. Так что теперь, можно проводить эксперименты и оптимизацию моделей автоматически, с относительно небольшими усилиями.

Моделирование большинства систем требует знания только малого подмножества блоков и процедур системы.

Однако для сложных систем, нужно знакомства со всем тем, что может предложить Object GPSS. Это пособие должно быть интересно, не только для студентов, но и для квалифицированных пользователей различных систем GPSS.

Проведение моделирования требуют выполнения нескольких шагов. Эти шаги обычно включают:

формирование модели и совокупность данных;
тестирование и проверку;
собственно моделирование;
экспериментирование;
анализ результатов.

В Object GPSS, вы создаете и изменяете модель, с помощью текстового редактора. Вы затем создаете собственно компилированную модель, используя пункт Run. В модели, в ее процедуре Simulation, вы можете использовать мощный набор процедур для того, чтобы управлять ходом моделирования, а значит можно автоматизировать проведение экспериментов с моделью. Вы можете управлять моделью в интерактивном режиме, или включать процедуры управления прямо в первоначальную модель. В ходе тестирования и проверки, доступно слежение за значениями параметров модели как в виде графиков или значений, так и в текстовой форме.

Поскольку языком моделирования на самом деле является Object Pascal, то при построении модели доступны многочисленные функции и библиотеки этого языка. Написанные пользователем процедуры можно использовать наравне с библиотечными процедурами.

Типы данных в системе могут быть целые, вещественные, строковые и логические. Каждый тип строго контролируется, а преобразование типов, если необходимо, должно выполняться явным образом. В системе можно использовать многомерные массивы любых объектов, что намного повышает возможности разработчика моделей. В системе можно создавать и уничтожать наборы X- параметров в динамике.

Модели могут связываться с внешним миром, используя текстовые файлы или окна диалога. Вы можете использовать эти файлы, чтобы обратиться к данным, создавать файлы результата или заказные отчеты и делать многое другое.

5 Введение в предметную область языка GPSS

Этот язык предназначен для изучения поведения систем массового обслуживания, в которых происходит конкуренция людей или заданий на обработку, за ограниченные ресурсы. И в этой связи, люди или задания выстраиваются в очереди, претендуя на обслуживание.

Простейшим примером системы массового обслуживания является система с одним устройством и очередью.

Рассмотрим систему, состоящую из одного человека, выполняющего обслуживание. Это может быть кассир, кладовщик, парикмахер и т.п. Клиенты приходят к такому обслуживающему устройству в случайные моменты времени, ждут очереди на обслуживание, а затем - обслуживаются. Как и в жизни, обслуживание идет по принципу:

кто первым пришел, тот первым обслужен. Будем считать, что мы собираемся моделировать работу кассира.

Тогда наша система характеризуется двумя независимыми величинами: интервалом прихода клиентов и интервалом их обслуживания.

После их задания, можно исследовать поведение такой системы, и в частности узнать:

число клиентов, пришедших в течение заданного промежутка времени;
количество клиентов, сразу же попавших на обслуживание;
среднее время, проведенное клиентом в очереди;
среднюю длину очереди;
среднее время, затраченное клиентом на обслуживание;
максимальную длину очереди за всё время наблюдения;
степень занятости нашего кассира.

Можно получить и много другой информации о поведении такой системы, причем с минимальными усилиями.

Вначале приведем текст такой модели с минимальными пояснениями, а затем рассмотрим процесс моделирования для этого примера подробнее.

Пусть интервал прихода клиентов подчиняется экспоненциальному закону распределения, и в среднем равен 120 секундам. Этот закон распределения хорошо описывает совершенно не организованный поток клиентов, и на практике, многие реальные законы распределения близки к экспоненциальному закону.

Пусть на обслуживание клиента кассир тратит от 80 до 130 секунд, причем любое время обслуживания является одинаково вероятным.

Мы будем считать, что нам нужно промоделировать работу системы при обслуживании 10000 клиентов.

Чтобы создать такую модель и провести моделирование, необходимо выполнить следующие действия.

Запустить программу Converter.exe.

Вставить такие объекты, как очередь (TQueue), одноканальное устройство (TFacility), а также генератор потока заявок (Generate). Это выполняется следующим образом.

В выпадающем списке выбирается Tqueue, а в поле имени вводится QKass. Выполняется пункт меню Insert Object.

Далее в выпадающем списке выбирается TFacility, а в поле имени вводится Kass. Выполняется пункт меню Insert Object.

И, наконец, в выпадающем списке выбирается Generate, в поле имени вводится Gen, а в поле аргументов вводится **Exponential(120)** . Выполняется пункт меню Insert Object.

После этого, в следующие строки процедуры ModelTxt вносится текст модели. В итоге, в этой процедуре вы должны получить.

```
{/Gen} ::Gen_   *:Gen.generate(Exponential(120));
   *: Qkass.Queue ;
   *: Kass.Seize ;
   *: Qkass.Depart ;
   *: Advance (105,25) ;
   *:  Kass.release ;
```

***: ToPoint (1,aclock,Qkass.A) ;**
***: Terminate (1);**

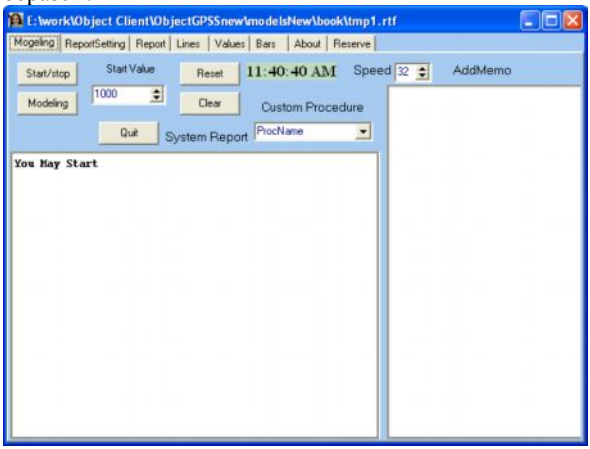
В процедуру Initial включите строку

SetStart(10000);

Этим вы зададите число обслуживаемых клиентов.

Затем сохраните полученный файл на диске (пункт Save As), и выполните пункт Run.

Вы получите модель нужной вам системы. Она будет выглядеть следующим образом.



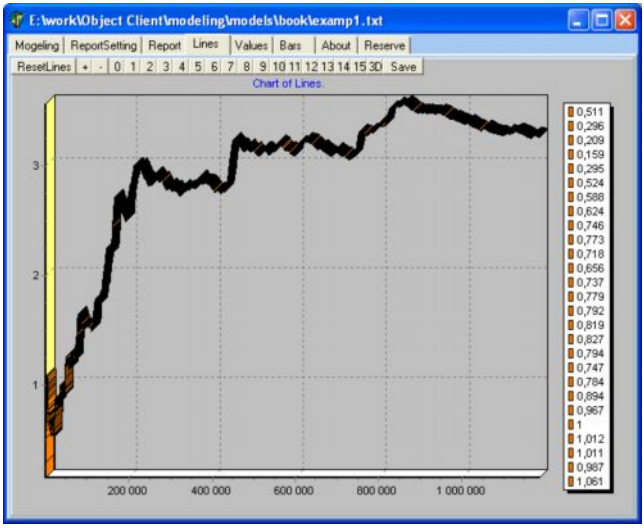
Нажмите на кнопку Simulation и дождитесь сообщения Stop Simulation.

Вы получите результат моделирования на вкладке Report и в файле Report.txt.

```
{/Gen} {::Gen_}   1:Gen.generate(Exponential(120));
     2: Qkass.Queue ;
     3: Kass.Seize ;
    4: Qkass.Depart ;
    5: Advance(105,25) ;
    6: Kass.release ;
    7: ToPoint(1,ac1,Qkass.A) ;
    8: terminate(1) ;

StartTime         0.00000 EndTime    1211278.33779
          BLOCK Report
    Location    Entries    Current
         1      10003          0
         2      10003          3
         3      10000          0
         4      10000          0
         5      10000          0
         6      10000          0
         7      10000          0
         8      10000          0
          Report CURRENT LIST
       TRAN      Assem        Pr    Current      Next       TimeStamp       TimeEnter Interrupt
     10001      10001         0          2         3   1211094.30851   1211094.30851         0
     10002      10002         0          2         3   1211142.35412   1211142.35412         0
     10003      10003         0          2         3   1211213.71985   1211213.71985         0
          Report FUTURE LIST
       TRAN      Assem        Pr    Current      Next       TimeStamp         EndTime Interrupt
     10004      10004         0          0         1   1211306.43774   1211306.43774         0
```

```
     Queue    Entries    Current        Max       Zero        AverQueue         AverTime    AverTime(-
Ze)     NumObj
     QKass      10003          3         27       1358          2.99449        362.60730
419.56748          3
  Facility    Entries    Current          Utility         AverTime     NextGo        TRAN       TRANP
NumObj
      Kass      10000          0          0.86898       105.25781          0           0       0
4
```

А на вкладке Lines вы увидите следующий график изменения средней величины очереди к кассиру во времени.



Теперь мы можем обсудить как саму модель, так и полученные результаты. Как правило, вам нет никакой надобности, внимательно изучать содержимое всех процедур модели, а достаточно сосредоточиться на процедуре ModelTxt. Содержимое остальных процедур система пишет «для себя», и в большинстве случаев оно для вас не важно, если, конечно вы там ничего не испортили.

{/Gen} ::Gen_ *:Gen.generate(Exponential(120));

Эта строка обеспечивает ввод в модель заявок, которые имитируют приход клиентов. Распределение интервалов прихода определяется экспоненциальным законом со средним интервалом 120 единиц модельного времени.

***: Qkass.Queue ;**

Эта строка моделирует поступление клиентов в очередь, с тем, чтобы можно было собрать статистику по очереди к кассе (Qkass)

***: Kass.Seize ;**

Эта строка моделирует начало работы кассира с клиентом. Клиент может попасть на обслуживание к кассиру, только тогда, когда кассир обслужит клиентов пришедших раньше. Когда с клиентом началась работа, кассир считается занятым.

***: Qkass.Depart ;**

Эта строка моделирует удаление клиента из очереди, так как он уже обслуживается у кассира. Если использовался блок QUEUE, то должен использоваться и блок DEPART. Только тогда будет собрана правильная статистика по очереди.

***: Advance(105,25) ;**

Эта строка моделирует задержку клиента на обслуживание у кассира. Средний интервал задержки равен 105 секундам модельного времени, а отклонение интервала от среднего составляет по 25 модельных секунд в обе стороны. Любое значение времени задержки из этого интервал абсолютно равновероятно.

***: Kass.release ;**

Эта строка моделирует завершение работы кассира с клиентом. После этого, кассир вновь становится свободным и может начать обслуживание следующего клиента. (если он есть)

***: ToPoint(1,aclock,Qkass.A) ;**

Эта строка обеспечивает вывод точек графика средней очереди к кассиру по ходу изменения модельного времени.

***: Terminate(1) ;**

Эта строка моделирует уход клиента из системы. А число 1 в ней обеспечивает завершение моделирования после того, как будет обслужено то число клиентов, которое указано в SetStart (то есть 10000).

Анализируя выходную статистику, можно определить следующие результаты моделирования. На обслуживание 10000 клиентов системе понадобилось 1211278.33779 секунд модельного времени. Всего поступило на обслуживание 10003 клиентов. Кассир был занят на 87%. Среднее время обслуживания клиента составило 105.26 секунды. Максимальная величина очереди составила 27 клиентов. 1358 клиентов вообще не стояли в очереди. Средняя величина очереди была 2.99. А среднее время, которое клиент провёл в очереди, равно 362.607 секунд.

Общий вывод – при такой скорости работы, система не очень хорошо справляется с работой, что может привести к потере клиентов. После нескольких экспериментов с различными временами обслуживания, нетрудно установить, что, заменив блок ADVANCE на

ADVANCE (80,25), мы получим следующие результаты.

Средняя занятость кассира - 0.669. Среднее время обслуживания - 80.235, Средняя длина очереди - 0.726. Среднее время в очереди- 87.025, что уже гораздо лучше.

Если посмотреть, что именно мы сохранили в качестве текста модели, то обнаружится следующее ее содержимое.

```
  {~vb} Var
{/QKass} QKass:Tqueue;
{/Kass} Kass:Tfacility;
{/Gen} Gen:Tgenerate;
  {~ve}
  {~pfe}
  {~ib} procedure Initial;begin
  SetStart(1000);
{/QKass}  init(QKass,' QKass');
```

```
{/Kass}  init(Kass,' Kass');
{/Gen}  init(Gen,' Gen',Gen_,Exponential(120));
  {~ie}end;
  {~mtb}procedure ModelTxt;begin case ActiveBlock of
{/Gen} ::Gen_   *:Gen.generate(Exponential(120));
     *: Qkass.Queue ;
     *: Kass.Seize ;
    *: Qkass.Depart ;
    *: Advance(80,25) ;
    *: Kass.release ;
    *: ToPoint(1,aclock,Qkass.A) ;
    *: terminate(1) ;
  {~mte} end;end;
  {~mb} procedure Simulation;  begin
   start(GetTerm);
  {~me}end;
  {~rb} procedure Report;begin
  {~re} end;
  {~cab} procedure CloseAllObj;begin
  {~cae} end;
```

Здесь можно обнаружить раздел описания объектов модели, раздел ее инициализации (procedure initial), раздел собственно описания модели (procedure modeltxt) , раздел манипуляций с моделью (procedure Simulation) и раздел формирования отчета (procedure report). Раздел удаления элементов модели (procedure CloseAllObj). Однако так как фактически мы написали только раздел описания модели, то обсуждение полного текста модели пока отложим. Здесь и в дальнейшем, текст, содержащий собственно модель будем выделять зеленым цветом.

Не вдаваясь в подробный анализ полученных результатов, попробуем глубже вникнуть в суть процесса моделирования этой простой системы. Представим себе логическую схему реализации модели этой системы на компьютере.

При моделировании систем массового обслуживания (СМО), ключевым понятием является событие.

В системе с одним обслуживающим элементом (кассиром) и очередью такие изменения, как приход клиента, начало обслуживания, конец обслуживания, называются событиями. Каждое событие в системе вызывают изменения состояния системы. Для построения модели, нужно для каждого события определить, как реализовать это событие и как корректировать состояние системы в связи ним. Среди всех событий ключевую роль при моделировании играют основные события.

Рассмотрим события, которые еще не возникли, но должны возникнуть. Эти события могут возникнуть немедленно (в текущий момент модельного времени) или попозже, если нужный момент для события еще не наступил. Такие события называются будущими событиями, и время наступления некоторых из них можно запланировать. Когда наступит время для запланированного события, оно непременно произойдёт.

Все события в системе разделим на две группы, основных и вспомогательных.

Основное - это такое событие, время возникновения которого можно запланировать заранее. Остальные события назовём вспомогательными, так как время

их возникновения заранее запланировать нельзя. Вспомогательные события возникают как следствие основных событий, и поэтому любое вспомогательное событие по времени возникновения обязательно совпадает с одним из основных событий.

В системе с кассиром и клиентами есть две группы основных событий – приход очередного клиента, и завершение обслуживания очередного клиента. Времена прихода клиентов должны соответствовать распределению интервалов прихода клиентов, а времена завершения обслуживания клиентов должны соответствовать распределению времен их обслуживания.

Предположим, что к кассиру пришёл клиент. Необходимо запланировать время прихода следующего клиента. Для этого:

1. Разыгрывают случайное число в соответствии с интегральным законом распределения интервалов прихода клиента:

T<прихода>

2. Эта величина прибавляется к текущему времени в модели (к текущему значению модельного таймера). Результат и определяет момент, когда в будущем придет следующий клиент. Время прихода следующего клиента предсказывается, но оно предсказывается точно, так как определяется на основе известного закона распределения случайных величин. После планирования, система работает так, что клиент придет именно тогда, когда она запланировала его приход.

При достижении моделью времени прихода клиента, он появляется на входе системы, и вновь планируется приход следующего клиента. Время прихода первого клиента обычно планируется, таким образом, как если бы в нулевой момент времени пришел клиент, но на обслуживание он не поступил.

Рассмотрим событие завершения обслуживания. Завершение планируется, когда обслуживание началось и ведется аналогично планированию времени прихода. Когда момент времени завершения обслуживания будет достигнут, может разыгрываться следующее время завершения обслуживания (если заявка есть в очереди или заявка только что поступила).

В общем случае возникновение одного основного события может вызвать планирование нескольких новых основных событий.

В системе с одним обслуживающим элементом, к вспомогательным событиям относятся:

поступление заявки на обслуживание. Оно возникает, например, когда обслуживающий элемент свободен и поступила заявка в систему.

Это вспомогательное событие вызывает переход обслуживающего элемента из свободного состояния в занятое и планирование времени окончания обслуживания.

Это же самое вспомогательное событие может возникнуть и как реакция на завершение обслуживания, если в очереди есть заявки.

Таким образом, нельзя планировать время поступления заявок на обслуживание.

Дальше будем называть устройством, оборудование, способное обрабатывать не более одной заявки одновременно.

6 Схема обработки основных событий

Приход заявки вызывает планирование следующего прихода.

Проверку состояния устройства - свободно? Да или нет.

По нет, происходит поступление заявки в очередь.

По да – происходит поступление заявки на обслуживание.

Это в свою очередь вызывает переход устройства в занятое состояние и планирование окончания обслуживания.

Окончание обслуживания вызывает.

Проверку состояния очереди - есть заявка? Да или нет.

По нет, происходит переход устройства в свободное состояние.

По да - поступление заявки на обслуживание.

Это в свою очередь вызывает продвижение заявок в очереди и планирование времени окончания обслуживания

Можно считать, что все выполняемые в ходе моделирования действия, кроме планирования, являются вспомогательными событиями. В списке операций, вызываемых событиями, у нас нет тех, которые обеспечивают сбор статистики. На практике эти операции входят в работу модели: например, если нас интересует максимальная длина очереди, то обработка события поступления заявки в очередь или ухода заявки из очереди должны быть расширены, с тем, чтобы обеспечивать сбор всей статистики по очереди.

7 Модельный таймер и завершение моделирования

Обычно в начале моделирования, значение таймера равно нулю и разработчик модели сам должен решить, какому времени этот нуль соответствует. Затем нужно выбрать единицу модельного времени - сутки, час, минута и т.п. Предположим, что моделируется система, и в данный момент ее состояние изменилось. Следующий логический шаг – это увеличение значения таймера.

Чисто теоретически, существуют два подхода:

1) увеличить его на некоторую единицу, проверить состояние системы и определить те из запланированных событий, которые должны произойти при новом значении таймера.

Если такие события есть, то нужно выполнить операции, реализующие события, и снова изменить значение таймера на эту единицу и т.д. Если событий на данный момент не запланировано, то значение таймера просто должно увеличиваться на эту единицу.

2) второй подход используют концепцию переменного приращения значения таймера. В соответствии с ним, время в модели наращивается до ближайшего основного события.

Очевидно, что выгоднее второй подход, т.к. при этом не обрабатываются те моменты времени, на которые не запланировано событий.

Реальный выигрыш во многом зависит как от вида модели, так и от качества реализации системы.

Ближайшее событие определяется на основе списка будущих событий. События в нем стоят в порядке наступления. Когда обрабатываются основные события, то список будущих событий может быть перестроен в соответствии со временем наступления каждого будущего события.

В принципе, рассмотренный подход обеспечивает бесконечное время моделирования системы. В действительности, рано или поздно, моделирование нужно прекращать. Завершение моделирование не всегда одномоментный процесс, например,

может быть желательно не принимать новых заявок в систему, а выполнить обслуживание уже имеющихся заявок.

Для завершения моделирования, в GPSS предусмотрен так называемый счетчик завершений. Процедура Start из Simulation задаёт начальное значение этого счётчика, а каждая заявка, покидающая систему, может уменьшать значение счетчика на некоторую величину (обычно на единицу), или не менять его значения. Как только в ходе моделирования содержимое счётчика окажется равным нулю или отрицательным, моделирование немедленно останавливается и выдаётся стандартная итоговая статистика. При желании, разработчик модели может использовать значение счётчика , как и другие системные параметры, для управления процессом моделирования. В GPSS принято завершать моделирование либо после обработки заданного количества заявок, либо после истечения определенного времени моделирования.

8 Одновременные события

События одновременны, если они происходят при одном значении модельного времени.

Пусть одновременно завершается обслуживание и поступает заявка. Тогда выбор ближайшего события неоднозначен, и в моделировании возникают тонкости. Зависимость от порядка обработки событий может повлиять на правильность моделей. Для нашего примера можно убедиться, что порядок обработки событий безразличен.

1. Пусть приходит заявка и очередь не пуста. Тогда в любом случае заявка попадает в очередь.

2. Пусть очередь пуста в момент прихода заявки. Тогда:

а) пусть сначала обрабатывается завершение обслуживания. Так как очередь пуста, то устройство станет свободным. Когда будет обрабатываться приход заявки - устройство будет свободно, и заявка попадает на обслуживание;

б) пусть сначала обрабатывается приход заявки. Она попадает в очередь. Затем происходит завершение обслуживания. Далее заявка из очереди попадает на обслуживание. Т.к. все это происходит в один момент модельного времени, то результат тот же самый.

Если задачу видоизменить и отказывать в обслуживании тем заявкам, которые приходят, когда устройство занято, то порядок обработки заявок станет важен.

Управлять порядком обработки можно с помощью приоритетов, которые также важны и при определении порядка принятия на обслуживание тех или иных заявок.

9 Выводы

Рассмотрение основных вопросов моделирования в простейшем случае, позволяет представить: какие средства нужно включить в язык моделирования Систем Массового Обслуживания (СМО).

Здесь нужны:

1) генераторы случайных чисел;

2) средства получения стандартных законов распределения (экспоненциальный, нормальный и др.);

3) средства задания эмпирических (экспериментальных) законов распределения;

4) встроенный таймер модельного времени;

5) автоматическое выполнение таких логических операций, как:
проверка состояния очереди при завершении обслуживания,
определение наличия заявок для обслуживания и т.п.;
6) автоматическое продвижение таймера к следующему основному событию;
7) автоматическая передача управления в ту часть модели, где находится схема обработки нужного события;
8) возможность присвоения приоритета заявкам, чтобы управлять последовательностью обработки событий;
9) средства автоматического сбора статистики в тех местах модели, которые интересуют разработчика. Обычно собирают статистику по очередям и по устройствам, которые есть в модели;
10) возможность обслуживания заявок в том порядке, в каком предпочитает пользователь;
11) автоматическая выдача итоговой статистики по модели.
Фактически все эти средства включены в язык GPSS.
Запись на языке GPSS очень компактна по числу операторов. Поэтому многие детали моделирования исчезают из поля зрения. Следовательно, разработка моделей может потребовать большей тщательности, чем это может показаться на первый взгляд.

10 Основные концепции моделирования на GPSS

Предварительные суждения.
G P S S - General Purpose Simulation System (общецелевая система моделирования).
Это система, воспринимает текст модели, и позволяет пользователю производить эксперименты с моделью на ЭВМ.
Модель на GPSS составляется из блоков, входящих в язык, и в этом виде поступает на моделирование.
Элементы модели GPSS.
Модель строится на основе объектов 4-х видов:
динамических,
аппаратно-ориентированных,
статистических,
операционных.
Динамические объекты - это элементы потока обслуживания (заявки). Они создаются или уничтожаются в модели специальными операторами. Работа системы отображается в модели в виде перемещения заявок от блоков GENERATE к блокам TERMINATE через другие блоки модели. Заявки являются абстрактными подвижными элементами, которые могут моделировать объекты реального мира (людей, сообщения, программы, транспортные средства и т.д.). Перемещаясь между блоками модели, заявки вызывают и испытывают различные воздействия. Возможны их задержки в некоторых местах модели, изменение маршрута в зависимости от условий, расщепление заявки на несколько копий и т.п. Каждая заявка перемещается вместе с набором своих параметров. Набор включает: номер заявки; номер блока, где сейчас находится заявка; номер блока, куда она должна перейти; время начала движения; приоритет, который определяет

порядок обработки, а также её личный набор параметров (P - параметров), которые задают желаемые характеристики моделируемого подвижного объекта.

11 Списки в GPSS

Любая из заявок хранится в GPSS в списках. Каждая заявка может быть:
в списке текущих событий,
в списке будущих событий,
в списке пользователя,
в списке прерываний,
в списке синхронизируемых заявок.

В списке текущих событий заявки расположены в порядке убывания приоритетов, а при равном приоритете - в порядке поступления в список.

Каждая заявка в этом списке может быть в потенциально активном состоянии, когда она подлежит просмотру в данный момент модельного времени, в состоянии движения, либо в состоянии задержки.

Если заявка находится в состоянии движения, то GPSS пытается продвинуть ее к следующим блокам, вплоть до выхода из системы. Если вход в очередной блок невозможен, то заявка переходит в состояние задержки. Если заявка попадает в блок ADVANCE, то она переходит из списка текущих событий в список будущих событий и для неё планируется время начала движения (после задержки). Если заявку удалость продвинуть по модели, то просмотр списка текущих событий начинается сначала, иначе, очередная заявка, из числа потенциально активных заявок, получат возможность двигаться. Когда окажется, что ни одна заявка из списка текущих событий не может быть продвинута, то считается, что настало время продвинуть таймер к очередному основному событию и его значение наращивается. Тогда из списка будущих событий в список текущих событий переходят те заявки, чьё время начала движения уже наступило, и просмотр списка текущих событий начинается сначала.

Список будущих событий содержат заявки, у которых еще не настало время начала движения. Эти заявки располагаются строго в порядке возрастания времени начала движения. В этом списке приоритеты не действуют. В нем хранятся заявки, находящиеся в блоке ADVANCE или запланированные к выходу из блока GENERATE.

Список пользователя содержит заявки, временно удаленные из списка текущих событий с помощью блока LINK. Их возвращение в список текущих событий возможно с помощью блока UNLINK. Количество списков пользователя может быть произвольным.

Список прерываний содержит ссылки на заявки, обслуживание которых прервано на устройстве. В этот список ссылки заносятся по мере поступления, а извлекаются по мере надобности. В этом списке для каждой заявки имеется ссылка на ту заявку, обслуживание которой прервала эта заявка. Поэтому возможно выстраивание цепочек прерываний.

Список синхронизируемых заявок. Он содержит заявки, которые ожидают комплектования в блоках ASSEMBLE и GATHER или ждут парной заявки в блоке MATCH.

12 Основные виды объектов в GPSS

В системе GPSS имеются процедуры и функции, которые не принадлежат никакому классу.

Clock - текущее значение условного времени моделирования. Оно автоматически изменяется программой и устанавливается в 0 управляющими процедурами ClearModel или ResetModel;

AClock - текущее значение абсолютного времени моделирования. Оно автоматически изменяется программой. Эта величина не меняется под действием управляющей процедуры ResetModel и устанавливается в 0 лишь под действием процедуры ClearModel;

Term - число, равное текущему значению счетчика завершений. Заявки, вошедшие в блоки TERMINATE с ненулевым операндом, уменьшают значение этого счетчика на число, равное значению операнда. Начальное значение счетчика определяет процедура START. При значении счетчика равном 0 или отрицательном, моделирование останавливается.

Наиболее важным объектом системы Object GPSS являются заявки. Они появляются из блока Generate и проходят через различные блоки модели, а затем уничтожаются блоками Terminate. С заявками связан ряд параметров, которые можно использовать для организации логики работы модели.

Параметры заявки (функции):

Tran Выдает номер активной заявки.

TranAge – определяет время пребывания заявки в модели. Эта величина может изменяться блоком MARK. Время пребывания вычисляется следующим образом: TranAge равно разнице текущего значения абсолютного времени и отметки времени рождения активной заявки;

Prior - приоритет активной в данный момент заявки. Эта величина может изменяться блоком PRIORITY. По умолчанию приоритет равен 0.

Assem - номер ансамбля, к которому принадлежит заявка.

P – параметры заявки могут иметь следующий вид: RP(номер) или IP(номер) или SP(номер) или BP(номер), обычно значение параметра «номер» выбирается для активной заявки. Эти параметры являются вещественными, целыми, строковыми или логическими соответственно. Если вы хотите узнать значение P- параметра, какой либо другой заявки (не активной), то следует указать ее номер вторым аргументом. Параметры заявки перемещаются и гибнут вместе с заявкой. Параметр создается при первом присваивании ему значения.

MatchBlock(номер) - флаг синхронизации: 1 , если заявка, находящаяся в блоке с данным номером, принадлежит тому же семейству, что и активная заявка; 0 - в противном случае.

С параметрами (функциями) других объектов GPSS, можно ознакомиться в Приложении 1.

Аппаратно-ориентированные объекты соответствуют элементам оборудования, которое управляет заявками. Применительно к ЭВМ - это устройства и накопители (многоканальные устройства).

Устройство – это оборудование, которое может одновременно обслуживать одну заявку, многоканальное устройство - это оборудование, которое может одновременно обслуживать некоторое число заявок. Устройства моделируют объекты, в которых может

происходить обработка заявки. Как правило, обработка связана с затратами времени. Существует полная аналогия между устройством и каналом системы массового обслуживания.

В GPSS можно моделировать прерывания на устройствах. Имеются средства логической проверки состояния устройств. Каждое из действий в устройстве отображается в модели определенным блоком. Занятие и освобождение устройства моделируются блоками: Seize, Release. Для проверки различного рода условий используется блок Test. Прерывания моделируют блоки: Preempt, Return, и так далее.

Многоканальные устройства служат для моделирования объектов обладающих емкостью.

Вход и выход для них моделируют блоки: Enter и Leave. Входящая в блок Enter заявка занимает часть емкости многоканального устройства, а выходящая через блок Leave заявка освобождает часть емкости. Емкость многоканального устройства указывается с помощью процедуры Init.

Для представления в модели системных данных, также используются P – параметры, только вместо номера заявки, указывается величина ObjName.N. Тогда любая функция, процедура или блок, которые работают с P – параметрами, будут работать с системными данными. Например, доступ к P – параметрам будет иметь вид: RP(номер, ObjName.N) или IP(номер, ObjName.N) или SP(номер, ObjName.N) или BP(номер, ObjName.N), Эти параметры являются вещественными, целыми, строковыми или логическими соответственно. P - параметры системы обычно существуют до конца моделирования. Данные, записанные в P – параметры, системы доступны любой заявке.

Статистические объекты - это очереди и таблицы. Они используются для наблюдения за поведением системы и не влияют на ее работу. В каждой точке модели может возникнуть список задержанных заявок. Заявка задерживается перед блоками, вход в которые в данных условиях не возможен. Они могут задерживаться перед блоками Seize, Enter, Test и т.д.

Для сбора статистики об очередях, в местах задержки ставят блоки Queue и Depart. При входе в первый блок текущая длина очереди обычно увеличивается на 1, при выходе - уменьшается.

Для сбора статистики по значениям какого-либо параметра и представления ее в виде стандартной таблицы используется блок Tabulate, а для ее описания – процедура Init.

Для временного хранения заявок, которые заведомо не должны пока двигаться по модели, используются списки пользователя. Заявки заносятся в списки блоками Link, а извлекаются из списков блоками UnLink.

13 Процедуры Object GPSS. Блоки и собственно процедуры

Каждый блок занимает в тексте модели определенное положение и имеет свой номер. Блоком называется особая процедура, которая работает только тогда, когда в нее входит заявка. Имя такой процедуры имеет право появиться только в процедуре ModelTxt.

Каждому блоку можно сопоставить символьную метку, которая располагается перед ключевым словом блока, отделяясь от него, по крайней мере, одним пробелом. Обыкновенный блок и блок с меткой должны выглядеть следующим образом.

```
*:Блок;
```

::метка *:Блок;

На символьную метку можно ссылаться непосредственно в выражениях и операндах. Метка может входить в выражения целого типа. Обычно метки имеют те блоки, на которые ссылаются другие блоки модели. В роли ссылки на метку можно использовать комбинацию символов !* , которая всегда замещается конвертером на номер следующего по порядку блока.

Операция блока - это глагол, определяющий основную роль блока.

Generate - генерирует

Test - проверяет

Queue - ставит в очередь

Advance - задерживает

Операнды задают уточняющую информацию для работы блока. Всего операндов может быть до 5. Типично когда в блоке 1 или 2 операнда.

Пример:

::Mm1 *:ADVANCE(300,20);

Блок ADVANCE задерживает заявку на 300+-20 единиц времени.

Операнды разделяются запятыми.

Комментарий – это текст, заключенный в символы { } или текст от пары символов // - до конца строки или текст, заключенный в пары символов (* *) . Вид комментария определяется по виду начала комментария. Рекомендуется первый или второй вид комментариев.

С блоками модели связаны следующие параметры (функции):

Total(номер) - общее число заявок, которое вошло в блок с данным номером. Подсчет ведется программой автоматически. Например, Total (MET1) - счетчик числа входов в блок с меткой ::MET1. Этот счетчик изменяется при каждом входе заявки в блок;

Current (номер) - текущее число заявок, которые находятся в блоке с данным номером. Значение этого счетчика также подсчитывается автоматически. Например, Current(MET2) - счетчик текущего числа заявок в блоке с меткой ::MET2.

Собственно процедуры ни в коем случае нельзя указывать в процедуре ModelTxt, и наоборот, блоки нельзя использовать вне процедуры ModelTxt. Обычно в подобных случаях возникает сообщение системы об ошибке или, реже - системная ошибка.

14 Основные процедуры и блоки GPSS

Процедура Start(число) - запускает моделирование. Она при работе с моделью может вызываться неоднократно. После прекращения моделирования, можно изменить модель с помощью процедур и вновь запустить ее с помощью процедуры Start. Здесь число - определяет содержимое счетчика завершений.

Блоки Terminate могут вычитать определенное значение из счетчика завершений. Когда в счетчике завершений будет 0 или отрицательная величина, то моделирование завершится.

Моделирование можно приостановить досрочно, нажав кнопку Start/Stop. Затем можно продолжить моделирование, вновь нажав эту же кнопку. Пример текста простой модели на Object GPSS мы рассматривали выше.

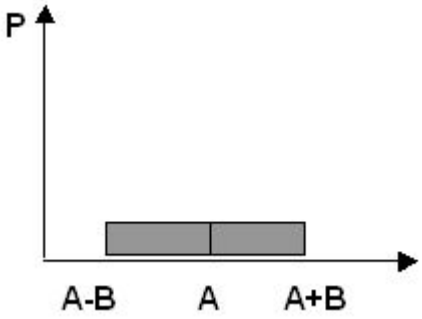
Для генерирования заявок используется блок GENERATE, имеющий вид:

::Gen_ *:Gen.Generate(Mean, Dev, PR, LimCount, CanOut);

Блок задает параметры потока заявок.

Здесь ::Gen_ - это обязательная метка блока, а Gen – это имя блока.

С помощью операндов Mean и Dev, задаваемых как вещественные или целые положительные числа (возможно, выражения), определяется равномерное распределение интервалов при генерации заявок. Интервалы между заявками могут быть от Mean-Dev до Mean +Dev включительно. Плотность вероятности в интервале равна 1/(2*Dev). По умолчанию, параметр Dev равен 0.



Здесь всегда Mean должно быть больше или равно Dev. Параметр PR – определяет приоритет заявки, по умолчанию он равен 0. Это самый низкий приоритет. Параметр LimCont – определяет граничное число заявок, которое может выпустить генератор, по умолчанию он равен 0 и ограничений на число заявок - нет. Параметр CanOut определяет, может ли быть выпущена заявка из генератора, по умолчанию, он имеет значение истина (True).

Приход новой заявки планируется только после выхода предыдущей.

Для каждого генератора в разделе инициализации помещается вызов процедуры

Init(Gen,' Gen',Gen_, Mean, Dev, PR, LimCount);

Она создает генератор с именем Gen и меткой Gen_, а также планирует приход в него первой заявки. Остальные параметры этой процедуры имеют тот же смысл, что и в блоке Generate, но относятся к первой заявке.

Как правило, параметры Mean, Dev, PR, LimCount – одинаковы в Generate и в Init.

Для блока

::Gen1_ *:Gen1.Generate(100,20);

заявки будут появляться через случайные интервалы. Каждый интервал может равновероятно принимать любое значение от 80 до 120.

С объектом генератор, связаны следующие функции.

Gen.C – выдает значение числа выпущенных заявок.

Gen.Max – выдает предельное число заявок.

Gen.Tran– выдает номер последней запланированной заявки.

Здесь Gen – это имя генератора.

Ввод заявок в систему ведет блок Generate, а вывод заявок из системы ведет блок Terminate. Заявка, попавшая в Terminate, удаляется из системы навсегда.

Одновременно, если в блоке Terminate указано число, то при прохождении через него заявки, из счетчика завершений вычитается это число. Когда в счетчике завершений будет 0 или отрицательная величина, моделирование останавливается и выдается итоговая статистика.

Общий вид блока – следующий:

*:Terminate(Num);

Моделирование задержки заявки ведет блок Advance, который имеет 1 или 2 параметра. Блок очень похож на Generate и его параметры Mean и Dev имеют тот же смысл. Они определяют интервал задержки заявки.

Общая форма блока:

*:Advance(Mean,Dev);

Войти в Advance и находиться в нем одновременно, может любое число заявок. Обычно блок Advance моделирует обслуживание заявок в устройстве или в многоканальном устройстве и поэтому он обрамляется блоками входа/выхода в них.

В случае устройств, занятие устройства отображается блоком

*:Ustr.Seize;

а освобождение устройства отображается блоком

*: Ustr.Release;

Здесь Ustr – это имя устройства.

Когда устройство занято, то в него не могут входить другие заявки, а когда оно свободно, то в него может войти первая по очереди заявка.

Устройство перестанет быть занятым, когда занявшая его заявка. пройдет через блок Release этого устройства.

Заявка может занять любое число устройств .

Заявки, стоящие в очереди перед Seize, обслуживаются в порядке поступления, с учетом приоритетов. Заявка, занимающая устройство, не должна покидать систему.

Для моделирования доступности устройства и включения отсылки заявок в определенное место модели используется блок.

*: Ustr.SetGoto(NumBl);

Если NumBl=0, то заявкам будет разрешено входить в устройство. Если NumBl<0, то заявкам будет запрещено входить в устройство. Если NumBl>0, то заявки, претендующие на устройство, в безусловном порядке будут отсылаться на блок с данным номером. По умолчанию, NumBl=0.

С устройствами связаны следующие функции:

Ustr.L - текущее состояние устройства с данным номером или именем. Эта величина равна 0, если устройство свободно, и 1 - во всех остальных случаях. Этот атрибут изменяется блоками Seize, Release, Preempt и Return.

Ustr.IsPree – логическое значение, определяющее, работает ли устройство в состоянии прерывания.

Ustr.A - определяет коэффициент занятости устройства.

Ustr.V - определяет временной интеграл устройства.

Ustr.T -определяет среднее время занятости устройства одной заявкой.

Ustr.C - определяет число заявок, вошедших в устройство.

Ustr.Tran - определяет номер обслуживаемой заявки для устройства.

Ustr. InterruptTran - определяет номер последней прерванной заявки для устройства.

Ustr.Go - определяет значение параметра NextGo, который установил блок *: Ustr.SetGoto.

Для слежения за очередями, а совсем не для их создания используются блоки

*: QUst.Queue(Num) ;

и

*: QUst.Depart(Num) ;

Здесь QUst – это имя очереди, а Num- это количество занимаемых или освобождаемых единиц очереди.

Эти блоки следят за очередями, которые возникают независимо от блоков слежения. Если Вы поставили эти блоки, то в итоговой статистике будет информация о соответствующей очереди.

При входе в блок *: QUst.Queue(Num) текущая длина очереди увеличивается на Num , при выходе через *: QUst.Depart(Num) , уменьшается на Num . По умолчанию, значение Num равно 1.

С очередями связаны следующие функции:

QUst.A- Определяет среднюю величину очереди.

QUst.T - Определяет среднее время в очереди для одной заявки.

QUst.X - Определяет среднее время в очереди для одной заявки.
Оно определяется без учета заявок, в очереди не задержавшихся.

QUst.C- Определяет число вхождений в очередь, с учетом кратности.

QUst.L - Определяет текущую длину очереди с учетом кратности.

QUst.M - Определяет текущую максимальную длину очереди с учетом кратности.

QUst.V - Определяет временной интеграл очереди с учетом кратности.

QUst.Z- Определяет текущее число вхождений в очередь с учетом кратности. Оно определяется для заявок, в очереди не задержавшихся.

Внимание!

При создании моделей не забывайте вставить в модель описания всех объектов, с которыми ваша модель работает.

Не забывайте, что каждый блок Generate является новым объектом, а поэтому нельзя создавать новые блоки Generate путем копирования.

Пример:

Пусть заявки приходят на обслуживание с средним интервалом 100 и отклонением 80 единиц времени. Распределение интервалов прихода – равномерное. Затем они обслуживаются в устройстве по очереди за среднее время 70 с отклонением в 20 единиц времени. Интервалы обслуживания распределены равномерно. Промоделировать работу такой системы в течение 100000 единиц времени. Организовать наблюдение за очередью.

Текст такой модели (процедура ModelTxt) выглядит следующим образом.

```
{/Gen} ::Gen_   *:Gen.generate(100,80);
   *: Qkass.Queue ;
   *: Kass.Seize ;
   *: Qkass.Depart ;
   *: Advance(70,20) ;
   *: Kass.release ;
   *: terminate ;
```

{/Gen0} ::Gen0_ *:Gen0.generate(100000);
***: terminate(1) ;**

В процедуре SetStart должно быть указан операнд , равный 1.

В данном примере, заявки приходят в систему с интервалом 100+- 80 единиц. Поступают в очередь Qkass, затем занимают устройство с именем Kass. Далее покидают очередь Qkass. Задерживаются в устройстве на 70+- 20 единиц времени. Затем покидают устройство Kass и удаляются из системы. Когда одна из заявок освобождает устройство, то его может занять первая по очереди заявка, и она пройдет тот же путь. Когда наступит момент времени 100000, то второй генератор выдаст первую заявку, которая будет тут же уничтожена и содержимое счетчика завершений станет равным 0, а значит, моделирование остановится и будет выдана итоговая статистика.

В этой модели должны быть описаны следующие объекты:

Gen, Gen0, Qkass, Kass.

То есть нужно описать два генератора заявок, очередь и одноканальное устройство.

В данном случае итоговая статистика модели будет выглядеть следующим образом.

```
{/Gen} {::Gen_}   1:Gen.generate(100,80);
     2: Qkass.Queue ;
     3: Kass.Seize ;
     4: Qkass.Depart ;
     5: Advance(70,20) ;
     6: Kass.release ;
     7: terminate ;
{/Gen0} {::Gen0_}   8:Gen0.generate(100000);
     9: terminate(1) ;

StartTime         0.00000 EndTime     100000.00000
          BLOCK Report
    Location   Entries   Current
         1       990         0
         2       990         0
         3       990         0
         4       990         0
         5       990         1
         6       989         0
         7       989         0
         8         1         0
         9         1         0
          Report FUTURE LIST
       TRAN     Assem        Pr   Current      Next       TimeStamp         EndTime Interrupt
       991       991         0         5         6     99939.75348    100012.98925         0
       992       992         0         0         1    100117.80994    100117.80994         0
       993       993         0         0         8    200000.00000    200000.00000         0
     Queue    Entries   Current       Max       Zero       AverQueue         AverTime    AverTime(-
Ze)    NumObj
     QKass          990           0           4         603           0.17436          17.61241
45.05500         3
  Facility   Entries   Current          Utility         AverTime     NextGo        TRAN       TRANP
NumObj
      Kass        990          1          0.69079         69.77651          0        991         0
4
```

Пусть в предыдущем примере, мы организуем многократное моделирование обслуживания по 500 групп заявок. Всего разрешим генератору выпустить 1200 заявок. В конце, когда моделирование завершится по ошибке, посмотрим содержимое отчета. Моделирование будем вести по количеству обслуженных заявок. Тогда вторая ветвь модели, начинающаяся со второго генератора – отсутствует, а вызов процедуры SetStart имеет вид SetStart(500).

В такой модели строка инициализации генератора заявок будет выглядеть следующим образом.

{/Gen} init(Gen,' Gen',Gen_,100,80,0,1200);

Нетрудно убедиться, что в этом случае моделирование действительно прервалось по ошибке.

Выходная статистика этого моделирования, имеют вид:

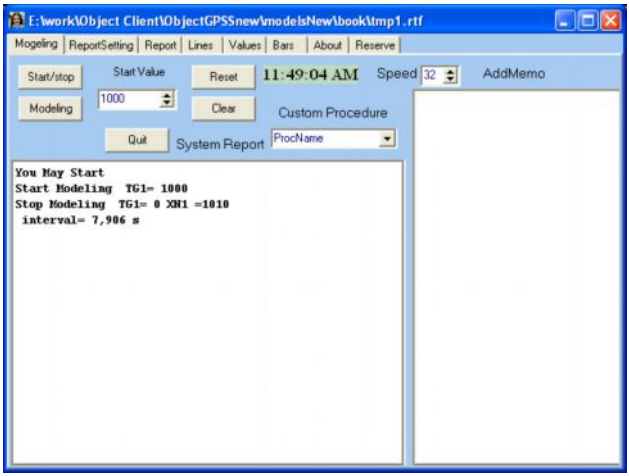
```
{/Gen} {::Gen_}   1:Gen.generate(100,80);
     2: Qkass.Queue ;
     3: Kass.Seize ;
     4: Qkass.Depart ;
     5: Advance(70,20) ;
     6: Kass.release ;
     7: terminate(1) ;

StartTime         0.00000 EndTime     119274.13288
          BLOCK Report
    Location   Entries   Current
         1      1200         0
         2      1200         0
         3      1200         0
         4      1200         0
         5      1200         0
         6      1200         0
         7      1200         0
     Queue   Entries   Current       Max       Zero       AverQueue         AverTime    AverTime(-
Ze)    NumObj
     QKass        1200            0           3         698            0.17152           17.04809
40.75242         3
  Facility   Entries   Current          Utility         AverTime     NextGo        TRAN       TRANP
NumObj
      Kass       1200          0          0.70551         70.12456          0          0       0
460   0
```

По выходной статистике данного примера видно, что генератор действительно выпустил 1200 заявок.

Процедура Start в ходе моделирования может быть выполнена несколько раз, и каждый раз будет запущен процесс продолжения моделирования из того состояния, в котором застала модель выдача итоговой статистики по нулевому значению счетчика моделирования.

15 Работа с моделью в Object GPSS

На вкладке Simulation приложения для работы с моделью расположены следующие кнопки.



Кнопка Simulation

Нажатие этой кнопки запускает выполнение процедуры Simulation, описанной в модели системы. В простейшем варианте, в этой процедуре находится только процедура Start(GetTg1); в которую передается значение поля, определяющего начальное содержимое счетчика завершений. В более сложных ситуациях здесь находятся процедуры управления моделью.

Кнопка Start/Stop

Нажатие этой кнопки позволяет временно приостановить выполнение моделирования. Однако, нажатие этой кнопки завершает выполнение текущей процедуры Start, и продолжение текущего процесса моделирования возможно только при повторном нажатии этой же кнопки.

Кнопка Reset

Нажатие этой кнопки вызывает сброс текущей статистики в текущее состояние. При этом все заявки остаются на прежних местах, и после выполнения процедуры Start – моделирование продолжится, однако статистика начнет собираться заново.

Кнопка Clear

Нажатие этой кнопки вызывает полный сброс модели, и она оказывается в том же состоянии, как и при запуске приложения.

Кнопка Quit

Нажатие этой кнопки завершает работу приложения.

На вкладке также находится поле для ввода значения счетчика завершений TG1.

Кроме того, на вкладке есть выпадающий список, который позволяет вызывать пользовательские процедуры управления моделью.

Для текстовых полей по правой клавише мыши вызываются пункты

Open

Save

Font

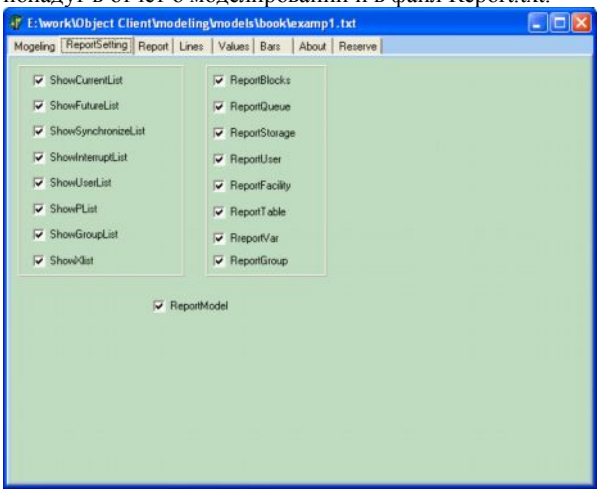
SelectAll

Copy

Paste

Они позволяют открыть в поле текстовый файл, сохранить содержимое поля как текстовый файл, изменить шрифт, а также обеспечивают работу с буфером обмена.

На вкладке Report Setting можно установить перечень частей отчета, которые попадут в отчет о моделировании и в файл Report.txt.



Эти части – следующие.

ShowCurrentList – отображать или нет список текущих событий.

ShowFutureList– отображать или нет список будущих событий.

ShowSynchronizeList – отображать или нет список синхронизируемых заявок.

ShowInterruptList– отображать или нет список ссылок на прерванные заявки.

ShowUserList– отображать или нет списки пользователя.

ShowPList – отображать или нет список P - параметров.

ShowGroupList– отображать или нет списки групп.

ShowXList – отображать или нет список X - параметров.

ReportBlocks - формировать или нет отчет по блокам.

ReportQueue - формировать или нет отчет по очередям.

ReportStorage - формировать или нет отчет по многоканальным устройствам.

ReportUser- формировать или нет отчет по спискам пользователя.

ReportFacility - формировать или нет отчет по одноканальным устройствам.

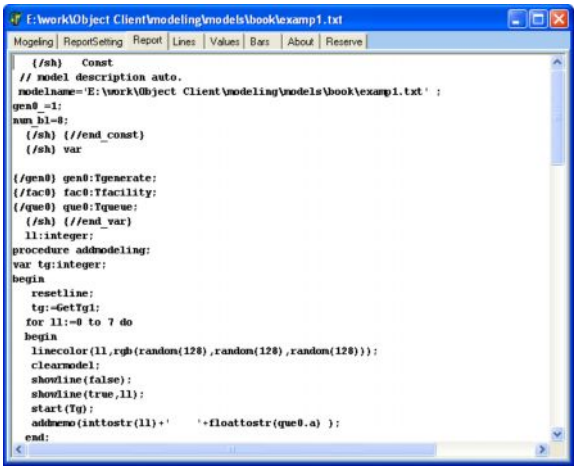
ReportTable - формировать или нет отчет по таблицам.

ReportVar - формировать или нет отчет по переменным модели.

ReportGroup - формировать или нет отчет по группам.

ReportModel - формировать или нет отчет по тексту модели.

На вкладке Report можно увидеть отчет, соответствующий настройкам предыдущей вкладки.



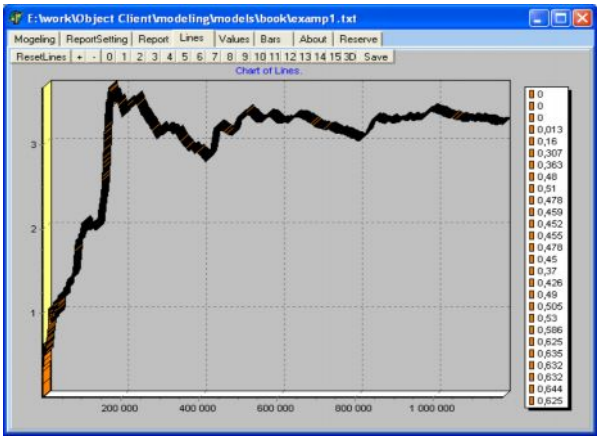
По правой клавише мыши вызываются пункты

Open
Save
Font
Word Wrap
Select All
Copy
Paste

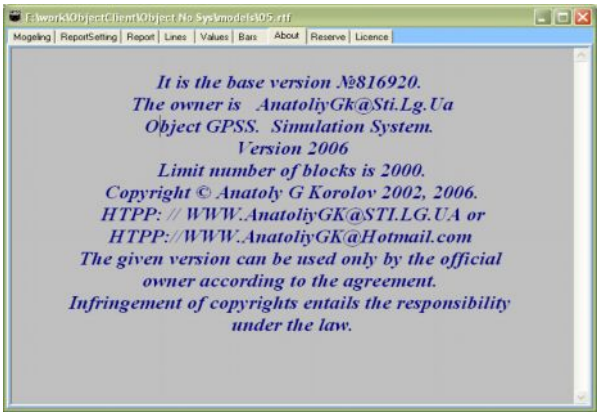
По первым шести ранее упомянутым пунктам делается то же самое, что и в остальных всплывающих меню, а по Word Wrap – устанавливается и отменяется перенос строк для компонента с текстом.

На вкладках Lines и Bars имеются кнопки для очистки графиков и гистограмм, а также для их сокрытия и отображения, и сохранения изображение в файле на диске. Если на этих вкладках левой кнопкой выделить прямоугольник, содержащий фрагмент изображения перемещением мыши левой клавишей вправо и вниз, то этот прямоугольник «растянется» на всю вкладку. Выделение прямоугольника другим способом – возвращает изображение к исходному виду.

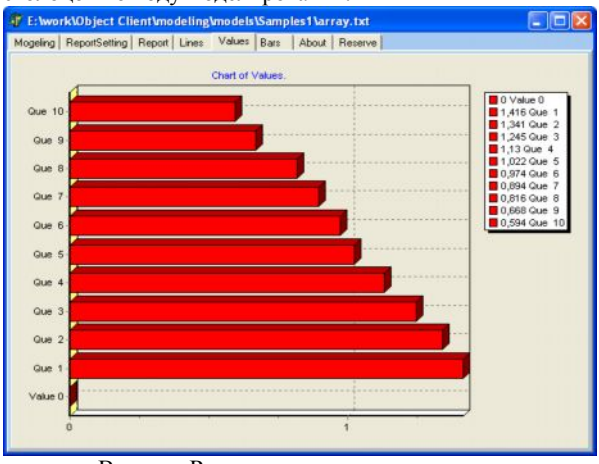
Правой кнопкой можно перемещать графики или гистограммы как единое целое.



На вкладке About приводятся сведения об авторских правах и адрес разработчика.



Вкладка Values предназначена для вывода числовых значений в виде величин столбцов по ходу моделирования.



Вкладка Reserve пока не используется.

16 Техника построения модели и проведения моделирования

Эта работа состоит из следующих этапов.

Вначале необходимо разработать детальное описание задачи, и сформулировать ее, используя понятия языка Object GPSS, а также указать назначение всех объектов модели. Здесь же нужно указать, что принято за единицу модельного времени.

После этого нужно сформировать текст модели, желательно с комментариями, используя для этого пункты File/New.

Перед набором текста, а если нужно то и по ходу набора, необходимо вставлять или удалять объекты, используя пункты Edit/Insert Object и Edit/Delete Object. Имейте в

виду, что удаляются автоматически только те строки с объектом, у которых псевдокомментарий вида {/Kass} размещен с первой позиции строки.

Когда текст будет набран, нужно выбрать пункт Run. Будет создана компилированная модель. Если при компиляции были ошибки, то модель не будет создана, а ошибки можно посмотреть в поле сообщений компиляции. Ошибки следует исправить в тексте, используя сведения о строках с ошибками. В программе – конвертере строка с надписью вида Line 40 определяет номера строк в исходном файле, и в файле Model.pas. В окне компиляции номера строк с ошибками будут определяться по файлу Model.pas, так что обнаружение строк с ошибками не составит труда.

После исправления ошибок можно запустить модель на выполнение, повторив пункт Run.

Когда моделирование будет завершено, то сформируется итоговая статистика на вкладке Report. Имейте в виду, что статистика формируется каждый раз заново, при каждом щелчке по этой вкладке. Одновременно эта статистика попадает и в файл Report.txt

При необходимости можно будет продолжить процесс моделирования, щелкнув по кнопке Simulation. Перед этим можно выполнить полный или частичный сброс статистики кнопками Clear или Reset.

Подробности истолкования итоговой статистики можно узнать из приложения 2.

Если сохранить файл ObjectGPSSProject.exe, то вы сможете эксплуатировать модель и без программы – конвертера. Это говорит о том, что компилированная модель сама по себе является программой и имеет самостоятельную ценность. В частности, она может быть настроена нужным образом с помощью процедур Object GPSS, или путем использования текстовых файлов, которые компилированная модель может прочитать. В процессе моделирования можно наблюдать за работой модели с помощью вкладок Lines, Values, Bar, а также прямо на вкладке Simulation в ее текстовых элементах. Подробнее эти возможности мы будем обсуждать позднее.

Рассмотрим способы управления компилированной моделью с помощью окон диалога и переменных.

Прежде всего, заменим все числа - переменными , и обеспечим задание их значения с помощью функций ввода.

Дополнительно нужно будет описать 5 вещественных переменных, а именно:

Inter_in, Inter_Work, Del_in, Del_Work, Time

Получим, например, следующий текст модели.

```
  {~vb} Var
 Inter_in:Double;
 Inter_Work:Double;
 Del_in:Double;
 Del_Work:Double;
 Time:Double;
{/QKass} QKass:Tqueue;
{/Kass} Kass:Tfacility;
{/Gen} Gen:Tgenerate;
{/Gen0} Gen0:Tgenerate;
  {~ve}
```

```
 {~pfe}
 {~ib} procedure Initial;begin
 Inter_in:=Rinput('Интервал поступления');
 Inter_Work:=Rinput('Интервал обслуживания');
 Del_in:=Rinput('Отклонение поступления');
 Del_Work:=Rinput('Отклонение обслуживания');
 Time:=Rinput('Время моделирования');
 SetStart(1);
{/QKass}  init(QKass,' QKass');
{/Kass}  init(Kass,' Kass');
{/Gen}  init(Gen,' Gen',Gen_,Inter_in,Del_in);
{/Gen0}  init(Gen0,' Gen0',Gen0_,Time);
 {~ie}end;
 {~mtb}procedure ModelTxt;begin case ActiveBlock of
{/Gen} ::Gen_   *:Gen.generate(Inter_in,Del_in);
   *: Qkass.Queue ;
   *: Kass.Seize ;
   *: Qkass.Depart ;
   *: Advance(Inter_Work,Del_Work) ;
   *: Kass.release ;
   *: terminate() ;
{/Gen0} ::Gen0_   *:Gen0.generate(Time);
   *: terminate(1) ;
 {~mte} end;end;
 {~mb} procedure Simulation;  begin
  start(GetTerm);
 {~me}end;
 {~rb} procedure Report;begin
{/Inter_in}  reportval(Inter_in,'Inter_in = ');
{/Inter_Work}  reportval(Inter_Work,'Inter_Work = ');
{/Del_in}  reportval(Del_in,'Del_in = ');
{/Del_Work}  reportval(Del_Work,'Del_Work = ');
{/Time}  reportval(Time,'Time = ');
 {~re} end;
 {~cab} procedure CloseAllObj;begin
 {~cae} end;
```

Проведя моделирование, нетрудно убедиться, что результат этого моделирования будет точно таким же, как и у исходной модели, если, конечно, вы введете те же исходные данные.

Впредь мы, не всегда будем приводить итоговую статистику. Мы будем приводить ее только в тех случаях, когда это особенно важно для правильного понимания работы той или иной модели.

Очевидно что, перед запуском модели заново, мы можем задать новые значения одной или нескольким переменным модели. Например, мы можем задать следующие значения переменных.

Inter_work = 90.35

Time = 200000

Получим другие результаты в файле итоговой статистики.

Аналогичным образом можно управлять другими параметрами модели. В этой модели каждая функция **Rinput** выведет окно для ввода вещественного значения, а текст, указанный в вызове попадет в заголовок окна.

В качестве законов распределения для интервалов прихода заявок, и интервалов обслуживания, кроме равномерного распределения довольно часто используется экспоненциальное и нормальное распределения интервалов. Для их задания используют встроенные функции Exponential(A) и Normal(A,B). Здесь в Exponential(A)

A –это среднее значение величины, распределенной по указанному закону.

A в Normal(A,B)

A– это среднее значение величины, распределенной по указанному закону.

B – это среднеквадратичное отклонение интервала.

Например, если в предыдущей задаче интервал поступления заявок подчиняется экспоненциальному закону, интервал обслуживания – нормальному, то текст модели может приобрести следующий вид.

```
  {~vb} Var
Inter_in:Double;
 Inter_Work:Double;
 Del_in:Double;
 Del_Work:Double;
 Time:Double; {/QKass} QKass:Tqueue;
{/Kass} Kass:Tfacility;
{/Gen} Gen:Tgenerate;
{/Gen0} Gen0:Tgenerate;
  {~ve}
  {~pfe}
  {~ib} procedure Initial;begin
  Inter_in:=Rinput('Интервал поступления');
  Inter_Work:=Rinput('Интервал обслуживания');
  Del_in:=Rinput('Отклонение поступления');
  Del_Work:=Rinput('Отклонение обслуживания');
  Time:=Rinput('Время моделирования');
  SetStart(1);
{/QKass}  init(QKass,' QKass');
{/Kass}  init(Kass,' Kass');
{/Gen}  init(Gen,' Gen',Gen_,Inter_in,Del_in);
{/Gen0}  init(Gen0,' Gen0',Gen0_,Time);
  {~ie}end;
  {~mtb}procedure ModelTxt;begin case ActiveBlock of
{/Gen} ::Gen_   *:Gen.generate(exponential(Inter_in));
    *: Qkass.Queue ;
    *: Kass.Seize ;
    *: Qkass.Depart ;
    *: Advance(Normal(Inter_Work,Del_Work)) ;
```

```
    *: Kass.release ;
    *: terminate() ;
{/Gen0} ::Gen0_   *:Gen0.generate(Time);
    *: terminate(1) ;
  {~mte} end;end;
  {~mb} procedure Simulation;  begin
   start(GetTerm);
  {~me}end;
  {~rb} procedure Report;begin
{/Inter_in}  reportval(Inter_in,'Inter_in = ');
{/Inter_Work}  reportval(Inter_Work,'Inter_Work = ');
{/Del_Work}  reportval(Del_Work,'Del_Work = ');
{/Time}  reportval(Time,'Time = ');
  {~re} end;
  {~cab} procedure CloseAllObj;begin
  {~cae} end;
```

Рассмотрим расширенный пример использования блоков работы с устройствами.

Пусть на одном компьютере работают два пользователя. Один из них использует его со средним интервалом 6 часов, который распределен по экспоненциальному закону. Работа на компьютере продолжается в течение 1.5 +-0.5 часа и включает в себя обращение к интернет через сервер в течении 20+-10 минут в начале работы.

Другой использует его со средним интервалом 4 часа, распределенным по экспоненциальному закону. Работа на компьютере продолжается в течении 1.3 +-0.75 часа и включает в себя обращение к интернет через сервер в течении 15+-7 минут в начале работы.

На сервер поступают другие задания со средним интервалом 30 минут, требующие обслуживания в течении 15 минут. Эти интервалы распределены по экспоненциальному закону.

Промоделировать работу системы в течение 120 месяцев по 22 рабочих дня, то есть в течение 120*22*8 часов. Собрать статистику по очередям, и по занятости устройств каждым пользователем.

Для сбора статистики по занятости устройств будем использовать дополнительные очереди, так как по очередям собирается наиболее обширная статистика. При описании модели следует вставить 2 устройства: **PC, SERV,**
4 очереди: **QPC, QSERV, Quser1, Quser2** и 4 генератора **gen0, gen1, gen2, gen3.** В дальнейшем, виды вставляемых устройств и их имена смотрите в разделе **Var** модели.

Текст модели может выглядеть, например, следующим образом.

```
  {~vb} Var
{/PC} PC:Tfacility;
{/SERV} SERV:Tfacility;
{/QPC} QPC:Tqueue;
{/QSERV} QSERV:Tqueue;
{/Quser1} Quser1:Tqueue;
{/Quser2} Quser2:Tqueue;
```

```
{/gen0} gen0:Tgenerate;
{/gen1} gen1:Tgenerate;
{/gen2} gen2:Tgenerate;
{/gen3} gen3:Tgenerate;
  {~ve}
  {~pfe}
  {~ib} procedure Initial;begin
  SETSTART( 120);
{/PC}  init(PC,' PC');
{/SERV}  init(SERV,' SERV');
{/QPC}  init(QPC,' QPC');
{/QSERV}  init(QSERV,' QSERV');
{/Quser1}  init(Quser1,' Quser1');
{/Quser2}  init(Quser2,' Quser2');
{/gen0}  init(gen0,' gen0',gen0_,exponential(6*60));
{/gen1}  init(gen1,' gen1',gen1_,exponential(4*60));
{/gen2}  init(gen2,' gen2',gen2_,exponential(30));
{/gen3}  init(gen3,' gen3',gen3_,22*8*60);
  {~ie}end;
  {~mtb}procedure ModelTxt;begin case ActiveBlock of
{/gen0} ::gen0_   *:gen0.generate(exponential(6*60));
    *: QPC.Queue;
    *: PC.Seize;
    *: QPC.Depart;
    *: Quser1.Queue;
    *: QSERV.Queue;
    *: SERV.SEIZE;
    *: QSERV.Depart;
    *: Advance( 20,10);
    *: SERV.Release;
    *: Advance(1.5*60,0.5*60);
    *: Quser1.Depart;
    *: PC.Release;
    *: Terminate;
{/gen1} ::gen1_   *:gen1.generate(exponential(4*60));
    *: QPC.Queue;
    *: Quser2.Queue;
    *: PC.Seize;
    *: QPC.Depart;
    *: QSERV.Queue;
    *: SERV.Seize;
    *: QSERV.Depart;
    *: Advance(15,7);
    *: SERV.Release;
    *: Advance(1.3*60,0.75*60);
    *: Quser2.Depart;
```

```
    *: PC.Release;
    *: Terminate;
{/gen2} ::gen2_   *:gen2.generate(exponential(30));
   *: QSERV.Queue;
   *: SERV.Seize;
   *: QSERV.Depart;
   *: Advance(exponential(15));
   *: SERV.Release;
   *: Terminate;
{/gen3} ::gen3_   *:gen3.generate(22*8*60);
    *: Topoint(0,aclock,Quser1.t);
    *: Topoint(1,aclock,Quser2.t);
    *: Terminate(1);
  {~mte} end;end;
  {~mb} procedure Simulation;  begin
   LineReset;
    start(GetTerm);
  {~me}end;
  {~rb} procedure Report;begin
  {~re} end;
  {~cab} procedure CloseAllObj;begin
  {~cae} end;
```

В этой модели используется 2 устройства PC – персональный компьютер, и Serv – сервер. Используются также очереди к устройствам Serv и PC. А для сбора статистики по занятости персонального компьютера и сервера каждым из пользователей, используются очереди QPC, QSERV, Quser1, Quser2.

Результат моделирования может выглядеть, например, следующим образом.

```
{/gen0} {::gen0_}   1:gen0.generate(exponential(6*60));
     2: QPC.Queue;
     3: PC.Seize;
     4: QPC.Depart;
     5: Quser1.Queue;
     6: QSERV.Queue;
     7: SERV.SEIZE;
     8: QSERV.Depart;
     9: Advance( 20,10);
     10: SERV.Release;
     11: Advance(1.5*60,0.5*60);
     12: Quser1.Depart;
     13: PC.Release;
     14: Terminate;
{/gen1} {::gen1_}   15:gen1.generate(exponential(4*60));
   16: QPC.Queue;
     17: Quser2.Queue;
     18: PC.Seize;
     19: QPC.Depart;
     20: QSERV.Queue;
     21: SERV.Seize;
     22: QSERV.Depart;
     23: Advance(15,7);
     24: SERV.Release;
     25: Advance(1.3*60,0.75*60);
     26: Quser2.Depart;
     27: PC.Release;
     28: Terminate;
{/gen2} {::gen2_}   29:gen2.generate(exponential(30));
```

```
   30: QSERV.Queue;
   31: SERV.Seize;
   32: QSERV.Depart;
   33: Advance(exponential(15));
   34: SERV.Release;
   35: Terminate;
{/gen3} {::gen3_}   36:gen3.generate(22*8*60);
     37: Topoint(0,aclock,Quser1.t);
     38: Topoint(1,aclock,Quser2.t);
     39: Terminate(1);

StartTime         0.00000 EndTime    1267200.00000
          BLOCK Report
    Location   Entries   Current
         1      3544         0
         2      3544         0
         3      3544         0
         4      3544         0
         5      3544         0
         6      3544         0
         7      3544         0
         8      3544         0
         9      3544         0
        10      3544         0
        11      3544         0
        12      3544         0
        13      3544         0
        14      3544         0
        15      5272         0
        16      5272         0
        17      5272         0
        18      5272         0
        19      5272         0
        20      5272         0
        21      5272         0
        22      5272         0
        23      5272         0
        24      5272         0
        25      5272         0
        26      5272         0
        27      5272         0
        28      5272         0
        29     42168         0
        30     42168         0
        31     42168         0
        32     42168         0
        33     42168         1
        34     42167         0
        35     42167         0
        36       120         0
        37       120         0
        38       120         0
        39       120         0
          Report FUTURE LIST
      TRAN     Assem        Pr   Current      Next       TimeStamp         EndTime Interrupt
     51107     51107         0         0        29   1267203.55807   1267203.55807         0
     51106     51106         0        33        34   1267191.83113   1267228.17054         0
     51091     51091         0         0        15   1267276.62181   1267276.62181         0
     51073     51073         0         0         1   1267426.48392   1267426.48392         0
     51108     51108         0         0        36   1277760.00000   1277760.00000         0
  Facility   Entries   Current         Utility        AverTime    NextGo       TRAN      TRANP
NumObj
        PC      8816         0         0.76965       110.62848         0          0      0
3
  Facility   Entries   Current         Utility        AverTime    NextGo       TRAN      TRANP
NumObj
      SERV     50984         1         0.61703        15.33630         0      51106      0
4
     Queue   Entries   Current       Max      Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
```

```
      QPC      8816         0        12      2058         1.18283       170.01770
221.79285         5
    Queue   Entries   Current       Max      Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
    QSERV     50984         0        15     20422         0.82305        20.45684
34.12642         6
    Queue   Entries   Current       Max      Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
   Quser1      3544         0         1         0         0.33642       120.29074
120.29074         7
    Queue   Entries   Current       Max      Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
   Quser2      5272         0        11         0         1.13732       273.37183
273.37183         8
```

В результате анализа статистики работы пользователей в течение 10 лет, мы получили следующие результаты. Хотя на персональном компьютере работало всего 2 пользователя, максимальная очередь по заданиям, которые они выполняли, составила12 заданий, а средняя очередь – 1.18 задания. Сам персональный компьютер имел занятость 0.77 . Сервер имел максимальную очередь в 15 заданий, а среднюю – в 0.82 задания. Первый пользователь выполнил 3544 заданий, а второй - 5272 заданий. Среднее время выполнения задания пользователями составило 110.63 минуты. А среднее время выполнения задания сервером, составило 15.33 минуты. Пользователям удалось сразу сесть за персональный компьютер всего 2058 раз. В то время, как всего было выполнено 8816 заданий. И это далеко не вся информация, полученная в ходе моделирования такой простой модели!

Прокомментируем попутно блоки и процедуры вывода графических данных.

Всего в модели одновременно можно выводить до 16 графиков. Вывод выполняют блоки *:ToPoint(Num,X,Y);

Здесь Num – это номер графика, он может иметь значение от 0 до 15.

X,Y – координаты очередной точки графика.

Для того, чтобы графики выводились корректно при каждом запуске процесса моделирования, в процедуру Simulation, описывающую план моделирования, можно включить процедуру LineReset (Num); которая удаляет график с номером Num. Если параметр опущен, то удаляются все графики.

17 Работа с многоканальными устройствами

Для моделирования работы с многоканальными устройствами используют блоки

Stor.*:Enter (Units); и

Stor.*:Leave (Units);

Здесь Stor.- это имя устройства, а - Units - количество занимаемых или освобождаемых единиц его емкости.

Для описания таких устройств используется процедура

Init(Stor,'Stor',Lim);

Здесь Stor.- это имя устройства, а - Lim - количество единиц его емкости.

Емкость должна быть целым числом. Она обычно определяет число заявок, которые могут одновременно обслуживаться в многоканальном устройстве.

Занятие многоканального устройства выполняется с помощью блока Stor.*:Enter (Units);

Освобождение выполняется с помощью блока

Stor.*:Leave (Units);

По умолчанию, Units равно 1, то есть заявка претендует на один канал.

Заявка может войти в устройство, если в нем есть то число свободных каналов, на которое она претендует. Иначе заявка ждет, пока нужное количество каналов освободится. При освобождении устройства, заявка может освободить любое число каналов, хотя обычно она освобождает то число каналов, которое она заняла. Недопустимо освобождать больше каналов, чем их занято в данный момент.

Если емкость многоканального устройства исчерпана или недостаточна для вхождения в него заявки, то заявка остается на выходе предыдущего блока.

Если в ходе моделирования нужно установить новую емкость многоканального устройства, то следует использовать блок

Stor.*:SetStorage (Units);

Здесь Units – новое количество каналов.

С целью управления доступностью многоканального устройства и, возможно, установления нового блока для принудительной отсылки заявки, претендующей на устройство, используется блок.

Stor.*:SetGoto (NumBl);

Смысл этого блока в точности такой же, как у одноименного блока для одноканальных устройств.

С многоканальными устройствами связаны следующие функции.

Stor.A - определяет среднее число занятых каналов.

Stor.AM - определяет среднее предельное число занятых каналов.

Stor.V - определяет временной интеграл устройства.

Stor.VM - определяет временной интеграл предельного числа занятых каналов.

Stor.T - определяет среднее время, проведенное заявкой в устройстве.

Stor.X - Определяет среднее время, проведенное заявкой в устройстве, для заявок, в устройстве задержавшихся.

Stor.C - определяет общее число занятий каналов.

Stor.L – определяет текущее число занятых каналов.

Stor.M - определяет максимальное число занятых каналов.

Stor.Z - определяет общее число занятий каналов, при которых заявки в устройстве не задерживались.

Stor.R - определяет текущее число свободных каналов.

Stor.U - определяет среднюю занятость устройства в расчете на 1 канал.

Stor.I - выдает минимальную занятость устройства.

Stor.Go - выдает значение поля NextGo, установленное блоком Stor.*:SetGoto.

Для своего обслуживания, заявка может одновременно занимать любое число одноканальных и многоканальных устройств.

Простейший пример применения многоканальных и одноканальных устройств совместно:

Пусть заявки первого типа поступают на обслуживание в систему с интервалом 100+-50 секунд. Они проходят вначале через 5 канальное устройство Nac, где обслуживаются в течение 400+- 100 сек. А затем они проходят через одноканальное устройство F, где обслуживаются в течение 90+- 20 сек, и покидают модель.

Пусть также заявки второго типа поступают на обслуживание в систему с интервалом 1000+-100 секунд. Они проходят через 5 канальное устройство Nac, где обслуживаются в течение 350+- 200 сек, занимая по 2 канала. Затем они покидают

модель. Промоделировать обслуживание 1000 заявок. Текст такой модели, мог бы иметь, например, следующий вид:

```
  {~vb} Var
{/F} F:Tfacility;
{/Nac} Nac:Tstorage;
{/QF} QF:Tqueue;
{/QNac} QNac:Tqueue;
{/G0} G0:Tgenerate;
{/G1} G1:Tgenerate;
 {~ve}
 {~pfe}
 {~ib} procedure Initial;begin
SetStart(1000);
{/F}  init(F,'F');
{/Nac}  init(Nac,' Nac',5);
{/QF}  init(QF,' QF');
{/QNac}  init(QNac,' QNac');
{/G0}  init(G0,' G0',G0_,100,50);
{/G1}  init(G1,' G1',G1_,1000,100);
 {~ie}end;
 {~mtb}procedure ModelTxt;begin case ActiveBlock of
{/G0} ::G0_   *:G0.generate(100,50);
     *: QNac.Queue ;
     *: Nac.Enter ;
     *: QNac.Depart ;
     *: Advance(400,100) ;
     *: Nac.Leave ;
     *: QF.Queue ;
     *: F.Seize ;
     *: QF.Depart ;
     *: Advance(90,20) ;
      *: F.Release ;
     *: Terminate(1) ;
{/G1} ::G1_   *:G1.generate(1000,100);
     *: QNac.Queue ;
     *: Nac.Enter(2) ;
     *: QNac.Depart ;
    *: Advance(350,200) ;
     *: Nac.Leave(2) ;
     *: Terminate ;
 {~mte} end;end;
 {~mb} procedure Simulation;  begin
  start(GetTerm);
 {~me}end;
 {~rb} procedure Report;begin
```

{~re} end;
{~cab} procedure CloseAllObj;begin
{~cae} end;

Результаты моделирования могут выглядеть, например, следующим образом:

```
{/G0} {::G0_}   1:G0.generate(100,50);
      2: QNac.Queue ;
      3: Nac.Enter ;
      4: QNac.Depart ;
      5: Advance(400,100) ;
      6: Nac.Leave ;
      7: QF.Queue ;
      8: F.Seize ;
      9: QF.Depart ;
      10: Advance(90,20) ;
      11: F.Release ;
      12: Terminate(1) ;
{/G1} {::G1_}   13:G1.generate(1000,100);
      14: QNac.Queue ;
      15: Nac.Enter(2) ;
      16: QNac.Depart ;
      17: Advance(350,200) ;
      18: Nac.Leave(2) ;
      19: Terminate ;

StartTime         0.00000 EndTime      99158.38168
          BLOCK Report
    Location   Entries   Current
         1      1006         0
         2      1006         1
         3      1005         0
         4      1005         0
         5      1005         3
         6      1002         0
         7      1002         2
         8      1000         0
         9      1000         0
        10      1000         0
        11      1000         0
        12      1000         0
        13       100         0
        14       100         0
        15       100         0
        16       100         0
        17       100         1
        18        99         0
        19        99         0
          Report CURRENT LIST
       TRAN     Assem        Pr   Current      Next       TimeStamp       TimeEnter Interrupt
      1100      1100         0         7         8     98491.04395     99011.66382         0
      1102      1102         0         7         8     98732.74902     99086.19113         0
      1107      1107         0         2         3     99107.67802     99107.67802         0
          Report FUTURE LIST
       TRAN     Assem        Pr   Current      Next       TimeStamp         EndTime Interrupt
      1108      1108         0         0         1     99184.90657     99184.90657         0
      1103      1103         0         5         6     98798.97843     99185.89617         0
      1094      1094         0        17        18     98905.17555     99247.61604         0
      1104      1104         0         5         6     98865.44204     99345.77134         0
      1105      1105         0         5         6     99012.28163     99493.60769         0
      1106      1106         0         0        13     99845.27716     99845.27716         0
  Facility   Entries   Current         Utility        AverTime    NextGo       TRAN      TRANP
NumObj
         F      1000         0         0.90370        89.60903         0          0      0
3
   Storage   Entries   Current       Max       Min        AverStor        AverTime
Utility
       Nac      1205         5         5         2         4.71657       388.12223
0.94331
    NextGo       TRAN             AC1    ACapac  Capacity    NumObj
```

```
        0      1105     99086.19113     5.000         5         4
     Queue   Entries   Current       Max       Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
       QF      1002         2         5       188         0.86314        85.41719
105.14499         5
     Queue   Entries   Current       Max       Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
      QNac     1106         1         6       342         1.28250       114.98271
166.45403         6
```

18 Арифметические, логические и строковые выражения в Object GPSS

В Object GPSS можно использовать различные выражения в качестве операндов.

Вид выражений фактически тот же, что и в Object Pascal. В выражении используют функции модели, метки, константы, знаки операций, вызовы библиотечных функций, и круглые скобки. Выражения должны составляться по правилам элементарной алгебры. Выражения вычисляются слева направо с учетом приоритетов операций. Числовые вычисления ведутся с двойной точностью.

Строковые константы представляют собой текст, заключенный в апострофы, например, ‘это строковая константа’ ‘It is string.’.

Логические константы могут быть только истина (True) и ложь (False).

Ниже приводятся операции и библиотечные функции, используемые в выражениях.

Арифметические выражения.

* - оператор арифметического умножения;

/ - оператор арифметического деления;

Div - оператор деления нацело, используется только для целых данных;

Mod - оператор деления по модулю, используется только для целых данных. Его результатом является остаток от деления нацело;

Shl - оператор сдвига целого числа влево. Он эквивалентен умножению числа на соответствующую степень двойки.

Shr - оператор сдвига целого числа вправо. Он эквивалентен делению числа на соответствующую степень двойки.

\+ - оператор арифметического или строкового сложения;

\- - оператор арифметического вычитания.

\- знак минус;

Отношения.

Они могут соединять пару арифметических или строковых выражений.

Если это действительно выражения, то они должны быть в круглых скобках.

< - меньше;

<= - меньше или равно;

> - больше;

>= - больше или равно;

= - равно;

<> - не равно;

Логические выражения.

Они могут соединять логические данные, и, в частности, отношения.

NOT - логическое отрицание: True, если операнд False;

False, если операнд True;

AND - оператор логического умножения: True, если оба операнда True, False - в противном случае;

OR - оператор логического сложения: True, если хотя один из операндов True, False - в противном случае;

XOR - оператор сложения по модулю два: True, если операнды различны, False - в противном случае;

Элементарные функции.

Abs( ) - абсолютное значение операнда;

ArcTan( ) арктангенс операнда, результат выражен в радианах;

Cos( ) - косинус операнда в радианах;

Exp( ) - экспонента операнда;

Int( ) - целая часть числа, однако сам результат - вещественный;

Ln( ) - натуральный логарифм операнда;

Log10 ()- десятичный логарифм операнда;

Pi – предопределенная константа равная π.

Power (A,B) – возведение в вещественную степень B положительного вещественного числа A . Всегда может быть заменено выражением Exp(Ln(A)*B) .

Random – случайное число, равномерно распределенное в диапазоне от 0 до 1.

Random(Num) – случайное целое число, равномерно распределенное в диапазоне от 0 до Num-1.

RandSeed – целое число, представляет собой текущее значение встроенного генератора случайных чисел.

Round() – вещественное число округляется до целого.

Sin( ) - синус операнда в радианах;

Sqr( ) - квадрат операнда;

Sqrt() – корень квадратный из операнда.

Trunc() усеченное целое, дробная часть отбрасывается.

Tan( ) - тангенс операнда в радианах;

Функции преобразования.

StrToInt() – преобразует строку – в целое число.

IntToStr() – преобразует целое число – в строку.

StrToFloat() – преобразует строку – в вещественное число.

FloatToStr() – преобразует вещественное число – в строку.

Для строк можно выполнять операцию + , которая означает соединение строк, а также операции сравнения.

Порядок действий соответствует следующим приоритетам, расположенным от высшего - к низшему:

Логическое не. (Not)

Умножение, деление, остаток, логическое и. (* / Div Mod And Shl Shr)

Сложение и вычитание, логическое или. (+ - Or)

Отношения. (< <= > >= = <>)

Кроме выше приведенных функций, имеется большая группа встроенных функций.

19 Табулирование переменных

В Object GPSS можно табулировать значения различных параметров и выражений.

Например: табулировать распределение длины очереди, табулировать распределение времени жизни заявки и так далее.

Пусть имеется некоторое выражение. Возможные значения этого выражения разобьем на ряд отрезков одинаковой длины. Получим ряд одинаковых интервалов и, кроме того, получим интервал от [-бесконечности до первой точки] и от [последней точки до +бесконечности] - эти последние интервалы называются интервалами переполнения.

При проходе заявки через блок Tabulate, который наблюдает за значениями данного выражения, определяется, в какой интервал попадает значение выражения. При этом корректируется таблица, которая фиксирует, сколько раз значение попало в каждый из интервалов, включая два бесконечные. Собранная информация выводится в итоговой статистике. Фактически при табулировании получается ненормированная гистограмма табулируемой величины.

Блок имеет следующий вид. *:tab.tabulate(Value,Num);

Здесь tab – имя таблицы, Value табулируемое значение, а Num – добавляемая к интервалу величина. Она равна 1 по умолчанию.

Таблица описывается процедурой

init(Tab,'Tab',Beg,Step,Num);

Здесь Beg - это первое граничное значение для интервалов;

Step- ширина интервала;

Num - это количество интервалов, включая два бесконечных.

Нижняя граница частотного класса всегда включается в предыдущий частотный класс.

С таблицами связаны следующие функции.

Tab.C - определяет число вхождений в таблицу с учетом кратности.

Tab.A - определяет среднее значение табулируемой величины с учетом кратности.

Tab.D - определяет среднеквадратичное отклонение табулируемой величины от средней.

Если вы хотите вывести гистограмму, соответствующей таблицы, то в процедуре Simulation нужно после Start поместить процедуру.

Show(Tab,Anum); или

GShow(Tab,Anum);

IShow(Tab,Anum);

Первая из них выводит таблицу как гистограмму, вторая выводит таблицу в виде графика, третья выводит таблицу в виде графика интегрального распределения. В этих процедурах Anum – номер ряда, используемого для вывода гистограммы или графика. Всего имеется 8 рядов (0.. 7) для гистограмм, и 16 рядов (0.. 15) для графиков.

Рассмотрим пример использования таблиц.

Пусть интервал поступления заявок в одноканальное устройство равен в среднем 110 и распределен по экспоненциальному закону.

Пусть время обслуживания равно 100, и тоже распределено по экспоненциальному закону. Протабулируем полное время обслуживания заявок и текущую длину очереди.

Тогда модель работы такого устройства может иметь вид.

```
  {~vb} Var
{/q2} q2:Tqueue;
{/fu} fu:Tfacility;
{/tbl} tbl:Ttable;
{/ng1} ng1:Tgenerate;
{/ng3} ng3:Tgenerate;
{/Tab0} Tab0:Ttable;
  {~ve}
  {~pfe}
  {~ib} procedure Initial;begin
 SetStart(1);
{/q2}  init(q2,'q2');
{/fu}  init(fu,'fu');
{/tbl}  init(tbl,' tbl',0,10,20);
{/ng1}   init(ng1,' ng1',ng1_,Exponential(110));
{/ng3}   init(ng3,' ng3',ng3_,100000);
{/Tab0} init(Tab0,' Tab0',0,5,100);
  {~ie}end;
  {~mtb}procedure ModelTxt;begin case ActiveBlock of
{/ng1} ::ng1_   *:ng1.generate(exponential(110));
    *:q2.queue;
    *:fu.seize;
    *:q2.depart;
    *: mark ;
    *:advance(exponential(100));
    *:fu.release;
    *:tbl.tabulate(tranAge);
   *:tab0.tabulate(q2.L);
    *:terminate;
{/ng3} ::ng3_   *:ng3.generate(100000);
      *:terminate(1);
  {~mte} end;end;
  {~mb} procedure Simulation;  begin
 {/sh}     start(GetTerm);
  {~me}end;
  {~rb} procedure Report;begin
  {~re} end;
  {~cab} procedure CloseAllObj;begin
  {~cae} end;
```

Результаты моделирования могут выглядеть, например, следующим образом:

```
{/ng1} {::ng1_}   1:ng1.generate(exponential(110));
    2:q2.queue;
    3:fu.seize;
    4:q2.depart;
    5: mark ;
    6:advance(exponential(100));
```

```
    7:fu.release;
    8:tbl.tabulate(TranAge);
    9:tab0.tabulate(q2.L);
    10:terminate;
{/ng3} {::ng3_}   11:ng3.generate(100000);
      12:terminate(1);

StartTime         0.00000 EndTime     100000.00000
          BLOCK Report
    Location   Entries   Current
         1       930         0
         2       930        14
         3       916         0
         4       916         0
         5       916         0
         6       916         1
         7       915         0
         8       915         0
         9       915         0
        10       915         0
        11         1         0
        12         1         0
          Report CURRENT LIST
       TRAN     Assem        Pr   Current      Next       TimeStamp       TimeEnter Interrupt
       918        918         0         2         3     98862.81090     98862.81090         0
       919        919         0         2         3     98876.88954     98876.88954         0
       920        920         0         2         3     98884.46790     98884.46790         0
       921        921         0         2         3     98901.95857     98901.95857         0
       922        922         0         2         3     99012.72591     99012.72591         0
       923        923         0         2         3     99089.54497     99089.54497         0
       924        924         0         2         3     99089.75108     99089.75108         0
       925        925         0         2         3     99379.37958     99379.37958         0
       926        926         0         2         3     99646.61849     99646.61849         0
       927        927         0         2         3     99663.92943     99663.92943         0
       928        928         0         2         3     99715.56225     99715.56225         0
       929        929         0         2         3     99784.76744     99784.76744         0
       930        930         0         2         3     99802.74517     99802.74517         0
       931        931         0         2         3     99872.17533     99872.17533         0
          Report FUTURE LIST
       TRAN     Assem        Pr   Current      Next       TimeStamp         EndTime Interrupt
       917        917         0         6         7     99850.25705    100005.32145         0
       932        932         0         0         1    100017.33636    100017.33636         0
       933        933         0         0        11    200000.00000    200000.00000         0
     Queue   Entries   Current       Max      Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
        q2       930        14        28        76         8.53572       917.81894
999.49838         3
  Facility   Entries   Current         Utility        AverTime    NextGo       TRAN      TRANP
NumObj
        fu       916         1         0.91878       100.30302         0       917      0
4
 Table       tbl  Entries       915.00000
 Mean       100.24898 StdDev       100.01203    NumObj         5
           Range       Frequency
         0.00000         0.00000
        10.00000        88.00000
        20.00000        79.00000
        30.00000        68.00000
        40.00000        59.00000
        50.00000        53.00000
        60.00000        57.00000
        70.00000        57.00000
        80.00000        48.00000
        90.00000        46.00000
       100.00000        39.00000
       110.00000        28.00000
       120.00000        27.00000
       130.00000        20.00000
       140.00000        19.00000
       150.00000        16.00000
```

```
      160.00000        16.00000
      170.00000        21.00000
      180.00000        14.00000
 1.000000E+0200       160.00000
Table      Tab0  Entries       915.00000
Mean         9.52240 StdDev          6.81631    NumObj         8
          Range       Frequency
        0.00000        75.00000
        5.00000       196.00000
       10.00000       199.00000
       15.00000       213.00000
       20.00000       169.00000
       25.00000        38.00000
       30.00000        25.00000
       35.00000         0.00000
```

20 Функции в Object GPSS

Для задания пользовательских функций с помощью точек или выражений, и некоторых других целей, используются числовые функции.

Вызов функции выполняется следующим образом.

Функция CFun(X, [,,],[,,]) -вычисляет функцию как непрерывную от заданного аргумента. В первых квадратных скобках следует перечислить через запятую значения аргумента X графика, а во вторых квадратных скобках следует перечислить через запятую значения функции Y графика. Можно без квадратных скобок указать имена динамических массивов с соответствующими значениями.

Функция DFun(X, [,,],[,,]) - вычисляет функцию как дискретную от заданного аргумента.

Функции ICFun(X, [,,],[,,]) и IDFun(X, [,,],[,,]) – аналогичны рассмотренным выше, но их значения округляются до целых.

Если вы хотите вывести числовую функцию как непрерывный или дискретный график, то в процедуре Simulation следует указать процедуру

CShow([,,],[,,],Anum);

DShow([,,],[,,],Anum);

Смысл ее параметров – такой же, как и у аналогичной процедуры, для таблицы, только список координат роднит ее с Cfun, DFun.

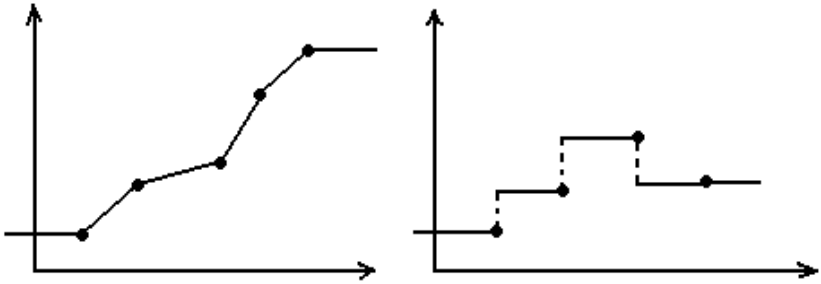


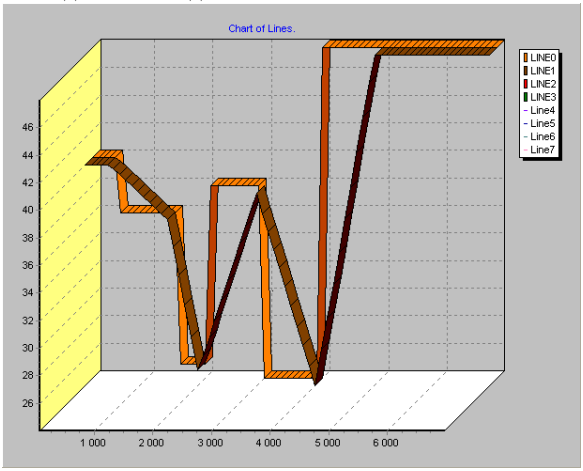
График непрерывной функции: График дискретной функции:

Пример описания и применения функции:

{~vb} Var

```
{/ng1} ng1:Tgenerate;
{/ng3} ng3:Tgenerate;
  {~ve}
  {~pfe}
  {~ib} procedure Initial;begin
  SetStart(1);
{/ng1}      init(ng1,' ng1',ng1_,100,50);
{/ng3}      init(ng3,' ng3',ng3_,7000);
  {~ie}end;
  {~mtb}procedure ModelTxt;begin case ActiveBlock of
{/ng1} ::ng1_    *:ng1.generate(100,10);
*:ToPoint(0,Clock,Dfun(Clock,[500,1500,2000,3000,4000,5000],[40,36,25,38,24,48]));
*:ToPoint(1,Clock,Cfun(Clock,[500,1500,2000,3000,4000,5000],[40,36,25,38,24,48]));
*:terminate;
{/ng3} ::ng3_    *:ng3.generate(7000);
     *:terminate(1);
  {~mte} end;end;
  {~mb} procedure Simulation;  begin
   LineReset;
     start(GetTerm);
   {~me}end;
  {~rb} procedure Report;begin
  {~re} end;
  {~cab} procedure CloseAllObj;begin
  {~cae} end;
```

График дискретной и непрерывной функции, заданной одним и тем же набором точек можно видеть на вкладке Lines.



Значения функций вычисляется следующим образом. Если аргумент имеет значение, меньшее, чем в первой точке, то функция имеет то же значение, что и в первой

точке. Если аргумент имеет значение, большее, чем в последней точке, то функция имеет то же значение, что и в последней точке. В остальных случаях значение функций вычисляется по-разному, в зависимости от их вида.

Для непрерывных функций, если аргумент лежит в интервале между двумя точками, то значение функции определяется с помощью линейной интерполяции.

Для дискретных функций, если значение аргумента лежит в интервале между двумя точками, то значение функции будет равно значению в правом конце интервала. Причем правый конец входит в интервал, а левый - нет.

Часто функции используются таким образом, что в качестве аргумента используется генератор случайных чисел: (Random).

Для задания случайной величины, распределенной по известному закону, всегда используется интегральная функция распределения. Так, например.

Пусть интервал поступления заявок в одноканальное устройство равен в среднем 110 и распределен по экспоненциальному закону.

Пусть время обслуживания для 10% заявок равно 100, для 20% заявок оно равно 80, для 40% заявок оно равно 120, а для остальных 30% - время обслуживания равно 60.

Тогда модель работы такого устройства может иметь вид.

```
  {~vb} Var
{/q2} q2:Tqueue;
{/fu} fu:Tfacility;
{/tbl} tbl:Ttable;
{/ng1} ng1:Tgenerate;
{/ng3} ng3:Tgenerate;
  {~ve}
  {~pfe}
  {~ib} procedure Initial;begin
 SetStart(1);
{/q2}  init(q2,' q2');
{/fu}  init(fu,' fu');
{/tbl}  init(tbl,' tbl',0,10,20);
{/ng1}   init(ng1,' ng1',ng1_,Exponential(110));
{/ng3}   init(ng3,' ng3',ng3_,100000);
  {~ie}end;
  {~mtb}procedure ModelTxt;begin case ActiveBlock of
{/ng1} ::ng1_   *:ng1.generate(300,50);
   *:q2.queue;
   *:fu.seize;
   *:q2.depart;
   *: mark ;
   *:advance(DFun(random,[0.1,0.3,0.7,1],[100,80,120,60]));
   *:fu.release;
   *:tbl.tabulate(tranAge);
   *:terminate;
{/ng3} ::ng3_   *:ng3.generate(100000);
     *:terminate(1);
```

{~mte} end;end;
{~mb} procedure Simulation; begin
start(Getterm);
show(tbl);
{~me}end;
{~rb} procedure Report;begin
{~re} end;
{~cab} procedure CloseAllObj;begin
{~cae} end;

Итоговая статистика этой модели имеет вид.

```
{/ng1} {::ng1_}   1:ng1.generate(300,50);
    2:q2.queue;
    3:fu.seize;
    4:q2.depart;
    5: mark ;
    6:advance(DFun(random,[0.1,0.3,0.7,1],[100,80,120,60]));
    7:fu.release;
    8:tbl.tabulate(TranAge);
    9:terminate;
{/ng3} {::ng3_}   10:ng3.generate(100000);
       11:terminate(1);

StartTime         0.00000 EndTime     100000.00000
Count0
          BLOCK Report
    Location   Entries   Current
         1       332         0
         2       332         0
         3       332         0
         4       332         0
         5       332         0
         6       332         0
         7       332         0
         8       332         0
         9       332         0
        10         1         0
        11         1         0
          Report FUTURE LIST   Count=  2
       TRAN     Assem        Pr   Current      Next       TimeStamp         EndTime Interrupt
       334       334         0         0         1    100267.87302    100267.87302         0
       335       335         0         0        10    200000.00000    200000.00000         0
     Queue   Entries   Current       Max      Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
        q2       332         0         1       332         0.00000         0.00000
0.00000         3
  Facility   Entries   Current         Utility        AverTime    NextGo       TRAN      TRANP
NumObj
        fu       332         0         0.29760        89.63855         0         0      0
4
 Table       tbl  Entries       332.00000
 Mean        89.63855 StdDev        26.12754    NumObj         5
           Range       Frequency
         50.00000         0.00000
         60.00000       116.00000
         70.00000         0.00000
         80.00000        65.00000
         90.00000         0.00000
        100.00000        26.00000
        110.00000         0.00000
        120.00000       125.00000
        130.00000         0.00000
```

Рассматривая текст этой модели можно заметить, что функция задает интегральное распределение в соответствии с заданием, и сама функция **DFun** является дискретной.

Для реализации возможностей других функций, которые применяются в GPSS, в Object GPSS реализованы следующие встроенные функции.

Uniform(VMin,VMax) Выдает вещественную случайную величину распределенную по равномерному закону в интервале от VMin до VMax.

DUniform(VMin,VMax) Выдает целую случайную величину распределенную по равномерному закону в интервале от VMin до VMax.

Exponential(Mean) Выдает случайную величину распределенную по экспоненциальному закону со средним значением Mean.

Normal(Mean,Dev) Выдает случайную величину распределенную по нормальному закону. Ее среднее значение Mean и среднеквадратичное отклонение Dev.

RByBool([],[]) Содержит список логических выражений и список вещественных выражений. Оба списка указываются в [] через запятую. Для первого истинного выражения, вычисляется соответствующее выражение из второго списка.

IByBool([],[]) Содержит список логических выражений и список целых выражений. Оба списка указываются в [] через запятую. Для первого истинного выражения, вычисляется соответствующее выражение из второго списка.

SByBool([],[]) Содержит список логических выражений и список строковых выражений. Оба списка указываются в [] через запятую. Для первого истинного выражения, вычисляется соответствующее выражение из второго списка.

RByNum(Num,[]) Содержит целое выражение и список вещественных выражений. Список указываются в [] через запятую. Вычисляется выражение номер Num из списка.

IByNum(Num,[]) Содержит целое выражение и список целых выражений. Список указываются в [] через запятую. Вычисляется выражение номер Num из списка.

SByNum(Num,[]) Содержит целое выражение и список строковых выражений. Список указываются в [] через запятую. Вычисляется выражение номер Num из списка.

BByNum(Num,[]) Содержит целое выражение и список логических выражений. Список указываются в [] через запятую. Вычисляется выражение номер Num из списка.

Рассмотрим пример применения таких функций.

Например, пусть интервал поступления заявок в одноканальное устройство равен в среднем 100 и распределен по экспоненциальному закону.

Пусть время обслуживания заявок зависит от длины очереди, причем, для очереди, длина которой меньше либо равна 2, время обслуживания выражается формулой, 120 - <длина очереди>*5.

Если длина очереди меньше либо равна 7, время обслуживания выражается формулой, 110 - <длина очереди>*8.

Если длина очереди меньше либо равна 11, время обслуживания выражается формулой, 100 - <длина очереди>*6. В остальных случаях время обслуживания равно 100 - <длина очереди>*15.

Модель работы такого устройства может иметь вид.

{~vb} Var
{/q2} q2:Tqueue;

```
{/fu} fu:Tfacility;
{/tbl} tbl:Ttable;
{/ng1} ng1:Tgenerate;
{/ng3} ng3:Tgenerate;
{/Tab0} Tab0:Ttable;
  {~ve}
  {~pfe}
  {~ib} procedure Initial;begin
 SetStart(1);
{/q2}  init(q2,' q2');
{/fu}  init(fu,' fu');
{/tbl}  init(tbl,' tbl',0,10,20);
{/ng1}  init(ng1,' ng1',ng1_,Exponential(100));
{/ng3}  init(ng3,' ng3',ng3_,100000);
{/Tab0}  Ttable.init(Tab0,' Tab0',0,1,100);
  {~ie}end;
  {~mtb}procedure ModelTxt;begin case ActiveBlock of
{/ng1} ::ng1_   *:ng1.generate(exponential(100));
   *:q2.queue;
   *:fu.seize;
   *:q2.depart;
   *: mark ;
   *:advance(RByBool([q2.L<=2,q2.L<=7,q2.L<=11,True],
      [120-q2.L*5,110-q2.L*8,100-q2.L*6,100-q2.L*15]));
   *:fu.release;
   *:tbl.tabulate(TranAge);
   *:tab0.tabulate(q2.L);
   *:terminate;
{/ng3} ::ng3_   *:ng3.generate(100000);
     *:terminate(1);
  {~mte} end;end;
  {~mb} procedure Simulation;  begin
 {/sh}     start(GetTerm);
  {~me}end;
  {~rb} procedure Report;begin
  {~re} end;
  {~cab} procedure CloseAllObj;begin
  {~cae} end;
```

Выходная статистика модели имеет вид.

```
{/ng1} {::ng1_}    1:ng1.generate(exponential(100));
    2:q2.queue;
    3:fu.seize;
    4:q2.depart;
    5: mark ;
    6:advance(RByBool([q2.L<=2,q2.L<=7,q2.L<=11,True],
         [120-q2.L*5,110-q2.L*8,100-q2.L*6,100-q2.L*15]));
    7:fu.release;
    8:tbl.tabulate(tranAge);
    9:tab0.tabulate(q2.L);
    10:terminate;
```

```
          {/ng3} {::ng3_}   11:ng3.generate(100000);
                 12:terminate(1);

          StartTime         0.00000 EndTime     100000.00000
                    BLOCK Report
              Location   Entries   Current
                   1       987         0
                   2       987         4
                   3       983         0
                   4       983         0
                   5       983         0
                   6       983         1
                   7       982         0
                   8       982         0
                   9       982         0
                  10       982         0
                  11         1         0
                  12         1         0
                    Report CURRENT LIST
                  TRAN     Assem        Pr   Current      Next       TimeStamp       TimeEnter
Interrupt
                  985       985         0         2         3     99841.47241     99841.47241
0
                  986       986         0         2         3     99893.70941     99893.70941
0
                  987       987         0         2         3     99912.47714     99912.47714
0
                  988       988         0         2         3     99944.75151     99944.75151
0
                    Report FUTURE LIST
                  TRAN     Assem        Pr   Current      Next       TimeStamp         EndTime
Interrupt
                  984       984         0         6         7     99998.55025    100076.55025
0
                  989       989         0         0         1    100080.79398    100080.79398
0
                  990       990         0         0        11    200000.00000    200000.00000
0
               Queue   Entries   Current       Max      Zero       AverQueue        AverTime
AverTime(-Ze)    NumObj
                q2       987         4        10        57         2.51679       254.99369
270.62233         3
            Facility   Entries   Current         Utility        AverTime    NextGo       TRAN
TRANP    NumObj
                  fu       983         1         0.95657        97.31175         0       984
0         4
           Table       tbl  Entries       982.00000
           Mean        97.40927 StdDev        19.54877    NumObj         5
                    Range       Frequency
                  40.00000         0.00000
                  50.00000         3.00000
                  60.00000        14.00000
                  70.00000       116.00000
                  80.00000       133.00000
                  90.00000       191.00000
                 100.00000         0.00000
                 110.00000       212.00000
                 120.00000       313.00000
                 130.00000         0.00000
           Table      Tab0  Entries       982.00000
           Mean         3.41752 StdDev         1.86955    NumObj         8
                    Range       Frequency
                  0.00000         56.00000
                  1.00000          0.00000
                  2.00000       100.00000
                  3.00000       156.00000
                  4.00000       212.00000
                  5.00000       191.00000
                  6.00000       134.00000
                  7.00000        85.00000
```

```
 8.00000        31.00000
 9.00000         9.00000
10.00000         5.00000
11.00000         3.00000
12.00000         0.00000
```

21 Функции и процедуры работы со строками

Эти функции могут использоваться для манипуляций со строками при вводе – выводе данных, а также для формирования значений X и P – параметров.

Кроме ранее рассмотренных, имеется следующий набор функций и процедур для работы со строками:

Функция

Pos(Substr, S)

Она ищет подстроку Substr в строке S. Когда найдена нужная подстрока, то позиция начала совпадения считается результатом, иначе результат равен 0.

Функция

Copy(S,Index,Count:)

Выделяет из строки S, начиная с позиции Index заданное число символов. Полученная строка считается результатом.

Процедура

Delete(S,Index,Count);

В строке S удаляется Count символов, начиная с символа с укзанным номером Index. Полученная строка заменяет S.

Процедура

Insert(Source,S, Index);

В строку S вставляется строка Source, начиная с позиции index. Полученная строка заменяет S.

Функция

Length(S) – определяет длину строки S.

Функция

StringReplace(S,OldPattern,NewPattern,Flags)

Возвращает новую строку, которая получена следующим образом. В строке S заменяется многократно подстрока OldPattern на подстроку NewPattern. Флаги, записанные в квадратных скобках управляют заменой.

Если есть флаг rfIgnoreCase, то при замене не учитывается регистр символов, то есть не различаются большие и малые буквы, иначе большие и малые буквы считаются различными.

Если есть флаг rfReplaceAll, то заменяются все подстроки, иначе только первая попавшаяся подстрока.

22 Блоки присваивания в GPSS

В Object GPSS имеется широкий набор блоков присваивания, так как каждый тип и вид данных, имеет свой блок присваивания.

Для задания приоритета заявки используется следующий блок.

*:Priority (Num,ATran);

Здесь Num – это новый приоритет активной заявки. Если нужно установить новый приоритет, какой – либо другой заявке, то номер этой заявки указывается вторым параметром.

После присваивания заявке нового приоритета, ее продвижение по модели прекращается, она устанавливается последней в своем приоритетном классе, и просмотр списка текущих событий начинается сначала.

Заявки с более высоким приоритетом, во всех устройствах принимаются на обслуживание раньше заявок с более низким приоритетом. Минимальный приоритет (по умолчанию) равен 0. Приоритет должен быть целым и положительным.

Пример.

Пусть имеется два потока заявок, обслуживаемых в двухканальном устройстве. Заявки первого типа требуют для обслуживания одного канала, а второго – двух. Интервалы поступления и обслуживания заявок распределены по экспоненциальному закону со средними значениями равными.

Для заявок первого типа – интервал поступления – 145 мс., обслуживания – 25мс.

Для заявок второго типа – интервал поступления – 170 мс., обслуживания – 100мс. Заявки второго типа обслуживаются с приоритетом.

Промоделировать работу системы в течение 1000000 мс.

Текст такой модели может иметь вид:

```
  {~vb} Var
{/ust0} ust0:Tqueue;
{/ust1} ust1:Tqueue;
{/ust2} ust2:Tqueue;
{/ust} ust:Tstorage;
{/G} G:Tgenerate;
{/G0} G0:Tgenerate;
{/G1} G1:Tgenerate;
  {~ve}
Const
Int1=145;
Wint1=25;
Int2=170;
Wint2=100;
  {~pfe}
  {~ib} procedure Initial;begin
     SetStart(1);
{/ust0}  init(ust0,' ust0');
{/ust1}  init(ust1,' ust1');
{/ust2}  init(ust2,' ust2');
{/ust}  init(ust,' ust',2);
{/G}  init(G,' G',G_,Exponential(int1));
{/G0}  init(G0,' G0',G0_,Exponential(int2));
{/G1}  init(G1,' G1',G1_,1000000);
  {~ie}end;
```

```
 {~mtb}procedure ModelTxt;begin case ActiveBlock of
{/G} ::G_   *:G.generate(Exponential(int1));
  *: ust0.queue ;
  *: ust.Enter ;
  *: ust0.depart ;
  *: ust1.queue ;
  *: advance(Exponential(Wint1))  ;
  *: ust1.depart ;
  *: ust.Leave ;
  *: terminate ;
{/G0} ::G0_   *:G0.generate(Exponential(int2));
  *: priority(1) ;
  *: ust0.queue ;
  *: ust.Enter(2) ;
  *: ust0.depart ;
  *: ust2.queue ;
  *: advance(Exponential(Wint2))  ;
  *: ust2.depart ;
  *: ust.Leave(2) ;
  *: terminate ;
{/G1} ::G1_   *:G1.generate(1000000);
    *: terminate(1) ;
 {~mte} end;end;
 {~mb} procedure Simulation;  begin
  start(GetTerm);
 {~me}end;
 {~rb} procedure Report;begin
 {~re} end;
 {~cab} procedure CloseAllObj;begin
 {~cae} end;
```

Здесь для сбора более подробной статистики, используются формальные очереди ust1 и ust2. Итоговая статистика модели имеет вид.

```
{/G} {::G_}   1:G.generate(Exponential(int1));
   2: ust0.queue ;
   3: ust.Enter ;
   4: ust0.depart ;
   5: ust1.queue ;
   6: advance(Exponential(Wint1))  ;
   7: ust1.depart ;
   8: ust.Leave ;
   9: terminate ;
{/G0} {::G0_}   10:G0.generate(Exponential(int2));
   11: priority(1) ;
   12: ust0.queue ;
   13: ust.Enter(2) ;
   14: ust0.depart ;
   15: ust2.queue ;
   16: advance(Exponential(Wint2))  ;
   17: ust2.depart ;
   18: ust.Leave(2) ;
   19: terminate ;
{/G1} {::G1_}   20:G1.generate(1000000);
      21: terminate(1) ;
```

```
StartTime         0.00000 EndTime    1000000.00000
          BLOCK Report
    Location   Entries   Current
          1      6811         0
          2      6811         0
          3      6811         0
          4      6811         0
          5      6811         0
          6      6811         0
          7      6811         0
          8      6811         0
          9      6811         0
         10      5901         0
         11      5901         0
         12      5901         0
         13      5901         0
         14      5901         0
         15      5901         0
         16      5901         0
         17      5901         0
         18      5901         0
         19      5901         0
         20         1         0
         21         1         0
           Report FUTURE LIST
      TRAN     Assem        Pr   Current      Next       TimeStamp         EndTime
Interrupt
     12715     12715         0         0         1   1000033.32834   1000033.32834
0
     12713     12713         0         0        10   1000130.17036   1000130.17036
0
     12716     12716         0         0        20   2000000.00000   2000000.00000
0
     Queue   Entries   Current       Max      Zero       AverQueue        AverTime
AverTime(-Ze)    NumObj
      ust0     12712         0        53      4080         3.97094       312.37699
460.02505          3
     Queue   Entries   Current       Max      Zero       AverQueue        AverTime
AverTime(-Ze)    NumObj
      ust1      6811         0         2         0         0.16965        24.90754
24.90754           4
     Queue   Entries   Current       Max      Zero       AverQueue        AverTime
AverTime(-Ze)    NumObj
      ust2      5901         0         1         0         0.59939       101.57399
101.57399          5
   Storage   Entries   Current       Max       Min        AverStor        AverTime
Utility
       ust     18613         0         2         0         1.36842        73.51966
0.68421
    NextGo        TRAN             AC1    ACapac  Capacity    NumObj
         0     12712    999994.36111     2.000         2         6
```

В GPSS имеются параметры заявки, которые, как правило, доступны только самой заявке (ее P – параметры). Для присваивания значений этим параметрам (RP(Num,ATran), IP(Num,ATran), SP(Num,ATran), BP(Num,ATran)) используются блоки

*:RPlet(Nums,Values,ATran);

Это блок присваивания списка вещественных значений Values - RP -параметрам с номерами из списка Nums. По умолчанию, значения присваиваются активной заявке. Иначе, заявке с указанным номером ATran .

*:IPlet(Nums,Values,ATran);

Это блок присваивания списка целых значений Values- IP –параметрам с номерами из списка Nums. По умолчанию, значения присваиваются активной заявке. Иначе, заявке с указанным номером ATran .

*:SPlet(Nums,Values,ATran);

Это блок присваивания списка строковых значений Values - SP -параметрам с номерами из Nums. По умолчанию, значения присваивается активной заявке. Иначе, заявке с указанным номером ATran . Учтите, что строки сохраняются в SP – параметрах без хвостовых пробелов.

*:BPlet(Nums,Values,ATran);

Это блок присваивания списка булевских значений Values BP -параметрам с номерами из Nums. По умолчанию, значения присваиваются активной заявке. Иначе, заявке с указанным номером ATran .

*:RPAdd(Nums,Values,ATran);

Это блок увеличивает на значения Values - RP -параметры с номерами Nums. По умолчанию, значения увеличиваются для активной заявки. Иначе, для заявки с указанным номером ATran .

*:IPAdd(Nums,Values,ATran);

Это блок увеличивает на значения Values - IP -параметры с номерами Nums. По умолчанию, значения увеличиваются для активной заявки. Иначе, для заявки с указанным номером ATran .

Для работы с P – параметрами вместо номера можно использовать переменную, которой присвоено значение функции Init(st) , где St – строка для идентификации параметра. Вместо первого списка можно указывать начальный номер параметров.

Пример блока.

*:IPLET(5,[och.L+30])

Означает, что нужно вычислить величину och.L+30 и записать результат в IP(5).

Примечание: не существующий RP,IP - параметр в выражениях имеет нулевое значение, SP – параметр имеет значение пустой строки, BP - параметр имеет значение False.

Общая форма обращения к параметрам заявок имеет вид.

RP(Num,ATran), IP(Num,ATran), SP(Num,ATran), BP(Num,ATran) .

Здесь Num – это номер параметра, а ATran – номер заявки. То есть, имеется возможность получить значение не только активной заявки, но и любой другой. По умолчанию, вы получаете значение параметра активной заявки.

Примечание: следует избегать использования параметров с номером ' -1'.

Кроме параметров заявки, в GPSS имеются параметры системы, так называемые ячейки. На самом деле, эти параметры принадлежат не существующим заявкам с отрицательным номером.

Для работы с ними можно использовать объект типа Tparam, который позволяет иметь набор именованных P – параметров. В этом классе определена единственная функция N, которая численно равна –NumObj, то есть номеру объекта с минусом.

Пусть вы создали переменную SYS класса Tparam. Тогда для присваивания значений ячейкам (RP(Num, SYS.N), IP(Num, SYS.N), SP(Num, SYS.N), BP(Num, SYS.N)) используются блоки

*:RPlet(Nums,Values, SYS.N);

*:IPlet(Nums,Values, SYS.N);

*:SPlet(Nums,Values, SYS.N);

*:BPlet(Nums,Values, SYS.N);

Это блоки присваивания списка значений Values - P -параметрам с номерами из Nums.

Все блоки, процедуры, и функции, определенные для P- параметров заявок, доступны и для ячеек, если вместо номера заявки указать функцию, в данном случае, SYS.N, или её численное значение.

Например, пусть описана переменная ISYS :Integer; тогда после инициализации переменной SYS класса Tparam, можно написать

ISYS := SYS.N; и впоследствии вместо SYS.N писать везде ISYS.

Рассмотрим примеры использования блоков PLET .

Пусть в систему поступают заявки со средним интервалом 100 мс., распределенным по экспоненциальному закону. Заявка может равновероятно занять любое из 5 устройств, где она будет обслуживаться за время, равное 490+-90 мс.

Промоделировать работу системы, в течение 1000000 мс. Определить суммарное время задержки заявок в очереди, к каждому устройству.

Для задания номера устройства и очереди, которые выбраны для обслуживания, будем использовать IP – параметры.

Тогда можно получить следующую модель.

```
  Const
num=5;
   {~vb} Var
{/gen0} gen0: Tgenerate;
{/fac0} fac0:array[1..num] of Tfacility;
{/que0} que0:array[1..num] of Tqueue;
{/Gen} Gen:Tgenerate;
{\Sys} Sys:Tparam;
  {~ve}
  {~pfe}
  {~ib} procedure Initial; var i:Integer; begin
    setstart(1);
  {/gen0}  init(gen0,' Gen0',gen0_,Exponential(100));
{/fac0}  for i:=1 to num do init(fac0[i] , 'fac0'+inttostr(i));
{/que0}  for i:=1 to num do init(que0[i] ,' que0'+inttostr(i));
{/Gen}  init(Gen,' Gen',Gen_,1000000);
{\Sys} Init(Sys,'Sys');
  {~ie}end;
  {~mtb}procedure ModelTxt;begin case ActiveBlock of
{/gen0} ::gen0_   *:gen0.generate(Exponential(100));
   *: IPlet(1,[duniform(1,num)]);
   *: que0[IP(1)].Queue;
   *: fac0[IP(1)].Seize;
   *: Advance(490,90);
   *: que0[IP(1)].Depart;
   *: fac0[IP(1)].Release;
   *: RPlet(IP(1),[RP(IP(1), sys.N)+TranAge] , sys.N) ;
   *: Terminate();
{/Gen} ::Gen_   *:Gen.generate(1000000);
 *: Terminate(1);
```

{~mte} end;end;
{~mb} procedure Simulation; begin
start(GetTerm);
{~me}end;
{~rb} procedure Report; begin
{~re} end;
{~cab} procedure CloseAllObj;begin
{~cae} end;

Обратите внимание, что очереди и устройства описаны как массивы, с номерами элементов от 1 до Num. В этой связи, в процедуре Initial, вызовы процедур для очередей и устройств выполнены в циклах. Обратите также внимание на то, что индексы объектов записываются в квадратных скобках.

Техника построения модели остается прежней. Нужно вначале вставить описания всех нужных объектов, а затем учесть, что часть из них является массивами. Делается это в соответствии с образцом.

Было

{/fac0} fac0: Tfacility;

Исправили и получили.

{/fac0} fac0:array[1..num] of Tfacility;

Аналогичным образом исправляются строки с вызовами процедур объекта

Было

{/fac0} init(fac0 , 'fac0');

Исправили и получили.

{/fac0} for i:=1 to num do init(fac0[i] , 'fac0'+inttostr(i));

Кроме этого, после заголовка процедуры Initial, добавлено описание переменной i, то есть, добавлен текст.

var i:integer;

В данном случае, удобнее задавать число элементов массива с помощью константы Num, а поэтому добавлен текст

num=5; в разделе Const.

Итоговая статистика этой модели будет иметь вид.

```
{/gen0} {::gen0_}   1:gen0.generate(Exponential(100));
   2: IPlet(1,[duniform(1,num)]);
   3: que0[IP(1)].Queue;
   4: fac0[IP(1)].Seize;
   5: Advance(490,90);
   6: que0[IP(1)].Depart;
   7: fac0[IP(1)].Release;
   8: sys.RPlet(IP(1),[sys.RP(IP(1))+TranAge]) ;
   9: Terminate();
{/Gen} {::Gen_}   10:Gen.generate(1000000);
 11: Terminate(1);

StartTime         0.00000 EndTime    1000000.00000
          BLOCK Report
    Location   Entries   Current
         1      9965         0
         2      9965         0
         3      9965        94
         4      9871         0
         5      9871         5
         6      9866         0
         7      9866         0
         8      9866         0
```

```
                  9      9866         0
                 10         1         0
                 11         1         0
                   Report FUTURE LIST
                TRAN     Assem        Pr   Current      Next       TimeStamp         EndTime
Interrupt
                9759      9759         0         5         6    978857.65498   1000128.75141
0
                9967      9967         0         0         1   1000130.99649   1000130.99649
0
                9900      9900         0         5         6    993846.57915   1000209.50859
0
                9858      9858         0         5         6    989303.37275   1000330.53029
0
                9891      9891         0         5         6    992885.50544   1000365.22346
0
                9939      9939         0         5         6    997587.88084   1000476.98850
0
                9968      9968         0         0        10   2000000.00000   2000000.00000
0
         SYS          Report RP LIST ( DOUBLE ) Count= 5
                Num          double
                  1  16691407.28571
                  2  11050932.31342
                  3  28981254.49613
                  4   8202208.96644
                  5  11733764.48587
             Facility   Entries   Current         Utility        AverTime    NextGo       TRAN
TRANP    NumObj
                fac01      1992         1         0.98025       492.09300         0      9858
0         4
             Facility   Entries   Current         Utility        AverTime    NextGo       TRAN
TRANP    NumObj
                fac02      1944         1         0.95404       490.76117         0      9900
0         5
             Facility   Entries   Current         Utility        AverTime    NextGo       TRAN
TRANP    NumObj
                fac03      2014         1         0.98394       488.55171         0      9759
0         6
             Facility   Entries   Current         Utility        AverTime    NextGo       TRAN
TRANP    NumObj
                fac04      1941         1         0.94895       488.89561         0      9891
0         7
             Facility   Entries   Current         Utility        AverTime    NextGo       TRAN
TRANP    NumObj
                fac05      1980         1         0.97013       489.96604         0      9939
0         8
                Queue   Entries   Current       Max      Zero       AverQueue        AverTime
AverTime(-Ze)    NumObj
                que01      2014        23        42         0        16.81893      8351.00698
8351.00698         9
                Queue   Entries   Current       Max      Zero       AverQueue        AverTime
AverTime(-Ze)    NumObj
                que02      1952         9        33         0        11.07675      5674.56555
5674.56555        10
                Queue   Entries   Current       Max      Zero       AverQueue        AverTime
AverTime(-Ze)    NumObj
                que03      2055        42        61         0        29.43498     14323.59136
14323.59136        11
                Queue   Entries   Current       Max      Zero       AverQueue        AverTime
AverTime(-Ze)    NumObj
                que04      1955        15        28         0         8.26436      4227.29528
4227.29528        12
                Queue   Entries   Current       Max      Zero       AverQueue        AverTime
AverTime(-Ze)    NumObj
                que05      1989        10        34         0        11.74649      5905.72720
5905.72720        13
```

Здесь опущена статистика по списку текущих событий, и по спискам P – параметров.

Видоизменим модель, с тем, чтобы получить среднеквадратичную величину времени в очереди для каждого устройства.

Тогда модель приобретет следующий вид.

```
  Const
num=5;
   {~vb} Var
{/gen0} gen0: Tgenerate;
{/fac0} fac0:array[1..num] of Tfacility;
{/que0} que0:array[1..num] of Tqueue;
{/Gen} Gen:Tgenerate;
{/BSigm} BSigm:Tparam;
{/BSum} BSum:Tparam;
{/BSsqr} BSsqr:Tparam;
{\Sys} Sys:Tparam;
  {~ve}
  {~pfe}
  {~ib} procedure Initial; var i:integer; begin
    setstart(1);
  {/gen0}  init(gen0,' gen0',gen0_,Exponential(100));
{/fac0}  for i:=1 to num do init(fac0[i], 'fac0'+inttostr(i));
{/que0}  for i:=1 to num do init(que0[i], 'que0'+inttostr(i));
{/Gen}  init(Gen,' Gen',Gen_,1000000);
{/BSigm}  init(BSigm,' BSigm');
{/BSum}  init(BSum,' BSum');
{/BSsqr}  init(BSsqr,' BSsqr');
{\Sys} Init(Sys,'Sys');
  {~ie}end;
  {~mtb}procedure ModelTxt;begin case ActiveBlock of
{/gen0} ::gen0_   *:gen0.generate(Exponential(100));
   *: IPlet(1,[duniform(1,num)]);
   *: que0[IP(1)].Queue;
   *: fac0[IP(1)].Seize;
   *: RPlet(IP(1),[RP(IP(1) ,bssqr.N)+sqr(TranAge)],bssqr.N) ;
   *: RPlet(IP(1),[RP(IP(1) , bsum.N)+1], bsum.N) ;
   *: RPlet(IP(1),[sqrt(RP(IP(1) ,bssqr.N)/RP(IP(1),bsum.N))], bsigm.N) ;
   *: Advance(490,90);
   *: que0[IP(1)].Depart;
   *: fac0[IP(1)].Release;
   *: RPlet(IP(1),[RP(IP(1), sys.N)+TranAge], sys.N) ;
   *: Terminate();
{/Gen} ::Gen_   *:Gen.generate(1000000);
 *: Terminate(1);
  {~mte} end;end;
  {~mb} procedure Simulation;  begin
    start(GetTerm);
```

{~me}end;
{~rb} procedure Report; begin
{~re} end;
{~cab} procedure CloseAllObj;begin
{~cae} end;

Итоговая статистика по модели будет иметь вид:

```
      {/gen0} {::gen0_}   1:gen0.generate(Exponential(100));
   2: IPlet(1,[duniform(1,num)]);
   3: que0[IP(1)].Queue;
   4: fac0[IP(1)].Seize;
   5: bssqr.RPlet(IP(1),[bssqr.RP(IP(1))+sqr(TranAge)]) ;
   6: bsum.RPlet(IP(1),[bsum.RP(IP(1))+1]) ;
   7: bsigm.RPlet(IP(1),[sqrt(bssqr.RP(IP(1))/bsum.RP(IP(1)))]) ;
   8: Advance(490,90);
   9: que0[IP(1)].Depart;
   10: fac0[IP(1)].Release;
   11: sys.RPlet(sys.IP(1),sys.RP(IP(1))+TranAge) ;
   12: Terminate();
{/Gen} {::Gen_}   13:Gen.generate(1000000);
 14: Terminate(1);

StartTime         0.00000 EndTime    1000000.00000
          BLOCK Report
    Location   Entries   Current
         1      9933         0
         2      9933         0
         3      9933        96
         4      9837         0
         5      9837         0
         6      9837         0
         7      9837         0
         8      9837         5
         9      9832         0
        10      9832         0
        11      9832         0
        12      9832         0
        13         1         0
        14         1         0
          Report FUTURE LIST
       TRAN      Assem        Pr   Current      Next       TimeStamp         EndTime Interrupt
       9847      9847         0         8         9    990315.46221   1000106.65680         0
       9884      9884         0         8         9    994201.45027   1000291.41292         0
       9935      9935         0         0         1   1000323.61351   1000323.61351         0
       9907      9907         0         8         9    997076.73477   1000375.89438         0
       9763      9763         0         8         9    983965.34743   1000392.65539         0
       9804      9804         0         8         9    987712.43607   1000448.56004         0
       9936      9936         0         0        13   2000000.00000   2000000.00000         0
SYS          Report RP LIST ( DOUBLE ) Count= 5
       Num          double
         1  27506913.88011
         2  22733232.67256
         3   9212776.57916
         4   7450468.28033
         5  13688736.03817
  Facility   Entries   Current         Utility        AverTime    NextGo       TRAN      TRANP
NumObj
     fac01      1996         1         0.98073       491.35019         0      9763      0
4
  Facility   Entries   Current         Utility        AverTime    NextGo       TRAN      TRANP
NumObj
     fac02      1981         1         0.97003       489.66555         0      9884      0
5
  Facility   Entries   Current         Utility        AverTime    NextGo       TRAN      TRANP
NumObj
     fac03      1924         1         0.94197       489.58986         0      9847      0
6
```

```
  Facility   Entries   Current         Utility        AverTime    NextGo       TRAN      TRANP
NumObj
     fac04      1941         1         0.95155       490.23812         0      9907      0
7
  Facility   Entries   Current         Utility        AverTime    NextGo       TRAN      TRANP
NumObj
     fac05      1995         1         0.97338       487.91087         0      9804      0
8
     Queue   Entries   Current       Max      Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
     que01      2026        31        61         0        27.78996     13716.66386
13716.66386         9
     Queue   Entries   Current       Max      Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
     que02      1991        11        53         0        22.77309     11438.01649
11438.01649        10
     Queue   Entries   Current       Max      Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
     que03      1939        16        29         0         9.28954      4790.89242
4790.89242        11
     Queue   Entries   Current       Max      Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
     que04      1948         8        25         0         7.46354      3831.38522
3831.38522        12
     Queue   Entries   Current       Max      Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
     que05      2029        35        39         0        13.97072      6885.51775
6885.51775        13
BSigm          Report RPLIST ( DOUBLE ) Count= 5
       Num          double
         1     15963.48753
         2     13276.98631
         3      5511.37081
         4      4189.50175
         5      7511.29495
BSum           Report RPLIST ( DOUBLE ) Count= 5
       Num          double
         1      1996.00000
         2      1981.00000
         3      1924.00000
         4      1941.00000
         5      1995.00000
BSsqr          Report RPLIST ( DOUBLE ) Count= 5
       Num          double
         1 508646536204.28046
         2 349207442188.41632
         3 58441900619.62623
         4 34068286252.28333
         5 112557005900.05280
```

В этой модели объект Bsigm используется для накопления суммы квадратов времен в очереди для каждой из очередей. Объект Bsum – используется для определения числа заявок, покинувших каждую из очередей. Объект BSsqr - используется для определения среднеквадратичных значений времени в каждой из очередей. Очевидно, что такая модель выглядит относительно логичной и понятной за счет использования объектов класса Tparam.

Для обнуления текущего времени жизни заявки, используется блок *:Mark(ATran);

Этот блок обнуляет время жизни заявки с данным номером, по умолчанию - активной.

Чаще всего блок используется для фиксации интервалов времени, прожитых заявками после прохождения определенного места в модели. Это позволяет фиксировать такие величины, как время обслуживания, время в очереди, и так далее.

Возможности данного блока очень легко перекрыть блоком RPlet.

Для этого, в нужном месте модели помещается блок, вида.
*:RPlet(Num,[AClock]);

Тогда в RP(Num) - параметр заявки заносится текущее время моделирования. Если впоследствии вычислить величину AClock - RP(Num) – то мы получим значение нужного нам интервала.

Чаще всего значение времени жизни заявки используется для табулирования времени обслуживания заявок в устройствах.

Примером использования блока MARK могла бы служить следующая задача.

Заявка проходит последовательно 3 устройства, протабулировать время, проведенное заявкой в очереди к каждому устройству.

Пусть интервал поступления заявок определяется величиной 100+-90 единиц времени. Время обслуживания в каждом из устройств составляет 90+-80, 75+-70, 60+-55.

Тогда модель этой системы могла бы выглядеть следующим образом.

```
  {~vb} Var
{/T} T:Ttable;
{/T0} T0:Ttable;
{/T1} T1:Ttable;
{/Q} Q:Tqueue;
{/Q0} Q0:Tqueue;
{/Q1} Q1:Tqueue;
{/F} F:Tfacility;
{/F0} F0:Tfacility;
{/F1} F1:Tfacility;
{/G0} G0:Tgenerate;
 {~ve}
 {~pfe}
 {~ib} procedure Initial;begin
  SetStart(1000);
{/T}  init(T,'t',0,100,100);
{/T0}  init(T0,'t0',0,50,100);
{/T1}  init(T1,'t1',0,20,100);
{/Q}  init(Q,'q');
{/Q0}  init(Q0,'q0');
{/Q1}  init(Q1,'q1');
{/F}  init(F,'f');
{/F0}  init(F0,'f0');
{/F1}  init(F1,'f1');
{/G0}  init(G0,'g0',G0_,100,90);
 {~ie}end;
 {~mtb}procedure ModelTxt;begin case ActiveBlock of
{/G0} ::G0_   *:G0.generate(100,90);
      *: RPlet(1,[AClock]) ;
      *: Q.Queue ;
      *: F.Seize ;
      *: Q.Depart ;
```

```
      *: T.Tabulate(AClock-RP(1)) ;
      *: Advance(90,80) ;
      *: F.Release ;
      *: RPlet(2,[AClock]) ;
      *: Q0.Queue ;
      *: F0.Seize ;
      *: Q0.Depart ;
      *: T0.Tabulate(AClock-RP(2)) ;
      *: Advance(75,70) ;
      *: F0.Release ;
      *: RPlet(3,[AClock]) ;
      *: Q1.Queue ;
      *: F1.Seize ;
      *: Q1.Depart ;
      *: T1.Tabulate(AClock-RP(3)) ;
      *: Advance(60,55) ;
      *: F1.Release ;
      *: Terminate(1) ;
 {~mte} end;end;
 {~mb} procedure Simulation;  begin
  BarReset;
  start(GetTerm);
 Show(T,1);
 Show(T0,2);
 Show(T1,3);
 {~me}end;
 {~rb} procedure Report;begin
 {~re} end;
 {~cab} procedure CloseAllObj;begin
 {~cae} end;
```

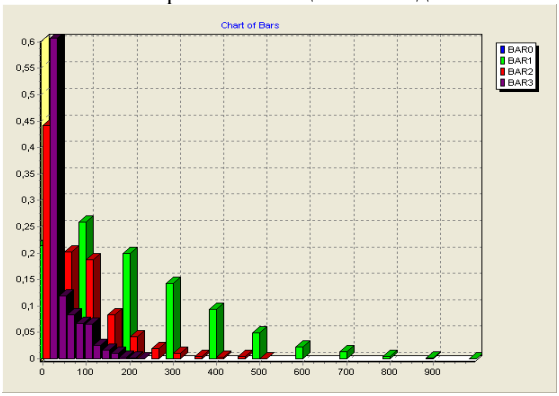
Тогда итоговая статистика будет иметь вид:

```
{/G0} {::G0_}   1:G0.generate(100,90);
         2: RPlet(1,[ACick]) ;
         3: Q.Queue ;
         4: F.Seize ;
         5: Q.Depart ;
         6: T.Tabulate(AClock-RP(1)) ;
         7: Advance(90,80) ;
         8: F.Release ;
         9: RPlet(2,[AClock]) ;
         10: Q0.Queue ;
         11: F0.Seize ;
         12: Q0.Depart ;
         13: T0.Tabulate(AClock-RP(2)) ;
         14: Advance(75,70) ;
         15: F0.Release ;
         16: RPlet(3,[AClock]) ;
         17: Q1.Queue ;
         18: F1.Seize ;
         19: Q1.Depart ;
         20: T1.Tabulate(AClock-RP(3)) ;
         21: Advance(60,55) ;
         22: F1.Release ;
```

```
                23: Terminate(1) ;

          StartTime         0.00000 EndTime      99633.96089
                    BLOCK Report
              Location   Entries   Current
                     1      1004         0
                     2      1004         0
                     3      1004         2
                     4      1002         0
                     5      1002         0
                     6      1002         0
                     7      1002         1
                     8      1001         0
                     9      1001         0
                    10      1001         0
                    11      1001         0
                    12      1001         0
                    13      1001         0
                    14      1001         1
                    15      1000         0
                    16      1000         0
                    17      1000         0
                    18      1000         0
                    19      1000         0
                    20      1000         0
                    21      1000         0
                    22      1000         0
                    23      1000         0
                      Report CURRENT LIST
                     TRAN     Assem        Pr   Current      Next       TimeStamp       TimeEnter
Interrupt
                     1003      1003         0         3         4     99494.80220     99494.80220
0
                     1004      1004         0         3         4     99603.64216     99603.64216
0
                      Report FUTURE LIST
                     TRAN     Assem        Pr   Current      Next       TimeStamp         EndTime
Interrupt
                     1001      1001         0        14        15     99305.13187     99648.68845
0
                     1002      1002         0         7         8     99403.00082     99697.86621
0
                     1005      1005         0         0         1     99701.21085     99701.21085
0
                        RP LIST ( Double )
                     TRAN       Num          Double
                     1001         1     99305.13187
                     1001         2     99622.20574
                     1002         1     99403.00082
                     1003         1     99494.80220
                     1004         1     99603.64216
          Table          T  Entries      1002.00000
          Mean       156.48705 StdDev       163.10273    NumObj          3
                     Range       Frequency
                   0.00000       214.00000
                 100.00000       260.00000
                 200.00000       200.00000
                 300.00000       143.00000
                 400.00000        94.00000
                 500.00000        49.00000
                 600.00000        22.00000
                 700.00000        14.00000
                 800.00000         4.00000
                 900.00000         2.00000
                1000.00000         0.00000
          Table         T0  Entries      1001.00000
          Mean        47.83552 StdDev        69.80704    NumObj          4
                     Range       Frequency
                   0.00000       442.00000
                  50.00000       203.00000
```

```
                100.00000       188.00000
                150.00000        83.00000
                200.00000        42.00000
                250.00000        20.00000
                300.00000        11.00000
                350.00000         5.00000
                400.00000         3.00000
                450.00000         4.00000
                500.00000         0.00000
          Table         T1  Entries      1000.00000
          Mean       17.82238 StdDev        31.08009    NumObj         5
                    Range       Frequency
                     0.00000       607.00000
                    20.00000       119.00000
                    40.00000        83.00000
                    60.00000        67.00000
                    80.00000        65.00000
                   100.00000        26.00000
                   120.00000        17.00000
                   140.00000        11.00000
                   160.00000         3.00000
                   180.00000         2.00000
                   200.00000         0.00000
                  Queue   Entries   Current       Max       Zero       AverQueue         AverTime
AverTime(-Ze)     NumObj
                    Q       1004         2         9        214         1.57546        156.34413
198.69557          6
                  Queue   Entries   Current       Max       Zero       AverQueue         AverTime
AverTime(-Ze)     NumObj
                   Q0       1001         0         5        442         0.48059         47.83552
85.65895           7
                  Queue   Entries   Current       Max       Zero       AverQueue         AverTime
AverTime(-Ze)     NumObj
                   Q1       1000         0         3        607         0.17888         17.82238
45.34956           8
               Facility   Entries   Current         Utility        AverTime    NextGo        TRAN
TRANP     NumObj
                    F       1002         1         0.88319        87.82038          0       1002
0          9
               Facility   Entries   Current         Utility        AverTime    NextGo        TRAN
TRANP     NumObj
                   F0       1001         1         0.74047        73.70229          0       1001
0         10
               Facility   Entries   Current         Utility        AverTime    NextGo        TRAN
TRANP     NumObj
                   F1       1000         0         0.60387        60.16611          0          0
0         11
```

Гистограммы таблиц имеют вид.



Для присваивания значений переменным используются блок.

*:Let(Varia,Value);

Этот блок присваивает значения Value - переменной Varia для вещественных, целых, строковых и логических значений соответственно. Если параметр Value отсутствует, то его значение будет 0, 0, ‘’, False.

*:Add(Varia,Value);

Этот блок увеличивает значение Value - переменной на значение Varia для вещественных и целых значений. Если параметр Value отсутствует, то его значение будет 1.

Если нужно задать значения переменным вне процедуры ModelTxt, то следует использовать оператор присваивания, вида Varia:=Value; .

В принципе, использование переменных или X – параметров – это дело вкуса автора модели, так как обычно все равно, что именно использовать.

Для перемещения активной заявки в конец списка текущих событий для данного приоритетного класса, используют блок

*:Buffer;

Иначе говоря, после него активная заявка пропускает вперед все заявки с данным приоритетом. Пример применения блока можно найти в разделе, где описываются блоки проверки условий. Фактически, блок Buffer эквивалентен блоку *:Priority(Prior); .

23 Перенаправление потоков заявок

Безусловное перенаправление заявок выполняет блок

*:Transfer(NumBl,NumIP);

Этот блок обеспечивает переход активной заявки на блок с меткой NumBl.

В параметр IP с номером NumIP - записывается номер следующего блока для возможного возврата. По умолчанию, номер следующего блока никуда не записывается. В полной форме этот блок напоминает переход на подпрограмму. А в сокращенной форме – он подобен оператору GoTO известных языков программирования.

При работе с фрагментами модели как с подпрограммами, числовые, строковые и логические данные могут передаваться как P – параметры заявки, или даже как временные наборы P – параметров.

Вероятностное перенаправление заявок выполняет блок.

*:Transfer(Ver,NumBl1,NumBl2); Здесь Numbl1 и Numbl2 - метки перехода, а Ver - вероятность. Вероятность соответствует второй метке.

Для создания временных наборов P – параметров можно использовать блок

*:IncTran, который увеличивает номер последней выпущенной заявки на 1, однако с сама заявка при этом не создается. Далее, с помощью функции MaxTran можно узнать этот номер и сохранить его в IP- параметре заявки, а затем, при работе с P – параметрами можно будет в качестве номера фиктивной заявки использовать этот номер. Когда временный набор P – параметров вам будет не нужен, то его можно удалить с помощью блоков

*: RPDelete(ATran); этот блок удаляет все RP -параметры данной заявки. По умолчанию – удаляются параметры активной заявки.

*: IPDelete(ATran); этот блок удаляет все IP -параметры данной заявки. По умолчанию - удаляются параметры активной заявки.

*: SPDelete(ATran); этот блок удаляет все SP -параметры данной заявки. По умолчанию - удаляются параметры активной заявки.

*: BPDelete(ATran); этот блок удаляет все BP -параметры данной заявки. По умолчанию - удаляются параметры активной заявки.

*:Loop(NumIP,NumBl);

Этот блок обеспечивает организацию цикла. Он уменьшает значение IP(NumIp) на 1.

Если оно не нулевое, то заявка уходит по метке Numbl, иначе она идет дальше по модели.

Примеры использования выше рассмотренных блоков.

10 сборщиков собирают изделия. Каждый собирает свое изделие за 30+-5 мин. Затем изделие подвергается напылению. На установке можно напылять сразу 2 изделия в течение 7+-3 мин. Затем проверяется качество каждого изделия по очереди в течение 3 мин. на установке контроля.

Промоделировать работу системы в течение 400 часов. Выполнить табулирование времени изготовления изделий.

```
  {~vb} Var
{/qq} qq:Tqueue;
{/dd} dd:Tqueue;
{/ss} ss:Tstorage;
{/un} un:Tfacility;
{/tab} tab:Ttable;
{/ng1} ng1:Tgenerate;
{/ng2} ng2:Tgenerate;
 {~ve}
 {~pfe}
 {~ib} procedure Initial;begin
 setstart(1);
{/qq}  init(qq,' qq');
{/dd}  init(dd,'dd');
{/ss}  init(ss,'ss',2);
{/un}  init(un,'un');
{/tab}  init(tab,' tab',0,1,100);
{/ng1}     init(ng1,' ng1',ng1_,0,0,0,10);
{/ng2}     init(ng2,' ng2',ng2_,10000);
 {~ie}end;
 {~mtb}procedure ModelTxt;begin case ActiveBlock of
{/ng1} ::ng1_   *:ng1.generate(0);
::ret   *:mark;
      *:advance(30,5);
      *:qq.queue;
      *:ss.enter(1);
      *:qq.depart;
```

```
      *:advance(7,3);
      *:ss.leave(1);
      *:dd.queue;
      *:un.seize;
      *:dd.depart;
      *:advance(3);
      *:un.release;
      *:tab.tabulate(TranAge);
      *:transfer(ret);
{/ng2} ::ng2_   *:ng2.generate(10000);
      *:terminate(1);
  {~mte} end;end;
  {~mb} procedure Simulation;  begin
  start(GetTerm);
  {~me}end;
  {~rb} procedure Report;begin
  {~re} end;
  {~cab} procedure CloseAllObj;begin
  {~cae} end;
```

Более сложный пример.

Заявки двух типов поступают в систему с интервалами 200+-150 и 300+-250. Они обслуживаются в устройстве за среднее время 90 и 110 соответственно. Протабулировать время, проведенное в устройстве заявками первого и второго типов. Промоделировать обслуживание 5000 заявок.

Текст такой модели может выглядеть следующим образом.

```
   {~vb} Var
{/gener0} gener0:Tgenerate;
{/gener1} gener1:Tgenerate;
{/table0} table0:Ttable;
{/table1} table1:Ttable;
{/facil0} facil0:Tfacility;
{/qu} qu:Tqueue;
  {~ve}
  {~pfe}
  {~ib} procedure Initial;begin
  SetStart(5000);
{/gener0}  init(gener0,' gener0',gener0_,200,150);
{/gener1}  init(gener1,' gener1',gener1_,300,250);
{/table0}  init(table0,' table0',0,10,100);
{/table1}  init(table1,' table1',0,10,100);
{/facil0}  init(facil0,' facil0');
{/qu}  init(qu,' qu');
  {~ie}end;
  {~mtb}procedure ModelTxt;begin case ActiveBlock of
{/gener0} ::gener0_   *:gener0.generate(200,150);
```

```
      *: RPlet(2,[90])  ;
      *: transfer(lab,2) ;
      *: table0.tabulate(TranAge) ;
      *: terminate(1) ;
{/gener1} ::gener1_   *:gener1.generate(300,250);
      *: RPlet(2,[110])  ;
         *: transfer(lab,2) ;
         *: table1.tabulate(TranAge) ;
          *: terminate(1) ;
::lab *: qu.queue ;
       *: facil0.seize  ;
        *: qu.depart ;
       *: mark ;
       *: advance(exponential(Rp(2))) ;
      *: facil0.release  ;
      *: topoint(0,aclock,qu.a) ;
      *: transfer(Ip(2))  ;
  {~mte} end;end;
  {~mb} procedure Simulation;  begin
   start(GetTerm);
  BARReset (1);
  BARReset (2);
 show(table0,2);
 show(table1,1);
  {~me}end;
  {~rb} procedure Report;begin
  {~re} end;
  {~cab} procedure CloseAllObj;begin
  {~cae} end;
```

Здесь блоки Transfer используются в двух вариантах.

Пример использования блока Loop:

Заявки обрабатываются устройством случайное число раз, от 1до 10. Кратность обработки определяется для заявки заранее и вероятность каждого из этих чисел одинакова. Интервал поступления заявок – 100+-80, интервал обслуживания – 17+-7.

Промоделировать работу системы по обслуживанию 1000 заявок.

Модель такой системы может выглядеть следующим образом.

```
  {~vb} Var
{/QKass} QKass:Tqueue;
{/Kass} Kass:Tfacility;
{/Gen} Gen:Tgenerate;
  {~ve}
  {~pfe}
  {~ib} procedure Initial;begin
  SetStart(1000);
{/QKass}  init(QKass,' QKass');
```

```
{/Kass}  init(Kass,' Kass');
{/Gen}  init(Gen,' Gen',Gen_,100,80);
  {~ie}end;
  {~mtb}procedure ModelTxt;begin case ActiveBlock of
{/Gen} ::Gen_   *:Gen.generate(100,80);
     *: IPlet(5,[Duniform(1,10)]) ;
::mmm     *: Qkass.Queue ;
     *: Kass.Seize ;
    *: Qkass.Depart ;
    *: Advance(17,7) ;
    *: Kass.release ;
    *: Loop(5,mmm) ;
    *: terminate(1) ;
  {~mte} end;end;
  {~mb} procedure Simulation;  begin
   start(GetTerm);
  {~me}end;
  {~rb} procedure Report;begin
  {~re} end;
  {~cab} procedure CloseAllObj;begin
  {~cae} end;
```

Результирующая статистика модели имеет вид.

```
{/Gen} {::Gen_}   1:Gen.generate(100,80);
      2: IPlet(5,[Duniform(1,10)]) ;
{::mmm}     3: Qkass.Queue ;
     4: Kass.Seize ;
    5: Qkass.Depart ;
    6: Advance(17,7) ;
    7: Kass.release ;
    8: Loop(5,mmm) ;
    9: terminate(1) ;

StartTime         0.00000 EndTime     100444.75089
          BLOCK Report
    Location   Entries   Current
         1      1005         0
         2      1005         0
         3      5570         5
         4      5565         0
         5      5565         0
         6      5565         0
         7      5565         0
         8      5565         0
         9      1000         0
          Report CURRENT LIST
        TRAN     Assem        Pr   Current      Next       TimeStamp       TimeEnter Interrupt
       1001      1001         0         3         4     99969.94682     99969.94682         0
       1002      1002         0         3         4    100115.41695    100115.41695         0
       1003      1003         0         3         4    100204.81148    100204.81148         0
       1004      1004         0         3         4    100370.85213    100370.85213         0
       1005      1005         0         3         4    100438.78236    100438.78236         0
          Report FUTURE LIST
        TRAN     Assem        Pr   Current      Next       TimeStamp         EndTime Interrupt
       1006      1006         0         0         1    100555.18388    100555.18388         0
          Report  IP LIST (  Integer  )
        TRAN       Num   Integer
       1001         5         9
       1002         5         6
       1003         5         4
```

```
      1004          5          8
      1005          5          2
     Queue    Entries    Current        Max       Zero        AverQueue         AverTime    AverTime(-
Ze)     NumObj
     QKass       5570          5         14       4683          3.55483         64.10479
402.55209          3
  Facility    Entries    Current          Utility         AverTime     NextGo        TRAN       TRANP
NumObj
      Kass       5565          0          0.94139         16.99149          0           0       0
4
```

Пусть в систему поступают заявки со средним интервалом 120, распределенным по экспоненциальному закону. В системе имеется два устройства. Первое устройство выбирается с вероятностью 0.4, а второе – с вероятностью 0.6 . Время обслуживания заявок в любом из устройств одинаково, и равно 150+-100. Промоделировать обслуживание 1000 заявок. Вывести графики занятости устройств, и средней очереди к ним.

Текст модели такой системы, с использованием обращения к объектам по индексу может выглядит следующим образом.

```
  {~vb} Var
{/QKass} QKass:Tqueue;
{/Kass} Kass:Tfacility;
{/Gen} Gen:Tgenerate;
{/kass1} kass1:Tfacility;
K:array[1..2] of Tfacility;
  {~ve}
  {~pfe}
  {~ib} procedure Initial;begin
  SetStart(1000);
{/QKass}  init(QKass,' QKass');
{/Kass}  init(Kass,' Kass');
{/Gen}  init(Gen,' Gen',Gen_,Exponential(120));
{/kass1}  init(kass1,' kass1');
k[1]:=kass;
k[2]:=kass1;
  {~ie}end;
  {~mtb}procedure ModelTxt;begin case ActiveBlock of
{/Gen} ::Gen_   *:Gen.generate(Exponential(120));
    *: Queue(Qkass) ;
    *: IPlet(1,[IByBool([random<0.4,true],[1,2])]) ;
    *: k[Ip(1)].Seize ;
    *: Depart(Qkass) ;
    *: Advance(150,100) ;
    *: k[Ip(1)].release ;
    *: ToPoint(2,aclock,Qkass.A) ;
    *: ToPoint(1,aclock,Kass1.A) ;
    *: ToPoint(0,aclock,Kass.A) ;
    *: terminate(1) ;
  {~mte} end;end;
  {~mb} procedure Simulation;  begin
```

```
   start(Getterm);
//   start(Gettg1);
  {~me}end;
  {~rb} procedure Report;begin
  {~re} end;
  {~cab} procedure CloseAllObj;begin
  {~cae} end;
```

Выходная статистика такой модели выглядит следующим образом. По этой статистике видно, что устройство kass1 действительно занималось чаще, чем kass. А обращение к устройствам в по индексу действительно работает.

```
{/Gen} {::Gen_}   1:Gen.generate(Exponential(120));
     2: Queue(Qkass) ;
     3: IPlet(1,[IByBool([random<0.4,true],[1,2])]) ;
     4: k[Ip(1)].Seize ;
     5: Depart(Qkass) ;
     6: Advance(150,100) ;
     7: k[Ip(1)].release ;
     8: ToPoint(2,aclock,Qkass.A) ;
     9: ToPoint(1,aclock,Kass1.A) ;
     10: ToPoint(0,aclock,Kass.A) ;
     11: terminate(1) ;

StartTime         0.00000 EndTime     114494.77313
Count0
          BLOCK Report
    Location   Entries   Current
         1      1007         0
         2      1007         0
         3      1007         6
         4      1001         0
         5      1001         0
         6      1001         1
         7      1000         0
         8      1000         0
         9      1000         0
        10      1000         0
        11      1000         0
          Report CURRENT LIST  Count= 6
       TRAN     Assem        Pr   Current      Next       TimeStamp       TimeEnter Interrupt
      1001      1001         0         3         4    114107.88656    114107.88656         0
      1002      1002         0         3         4    114116.15700    114116.15700         0
      1003      1003         0         3         4    114173.61162    114173.61162         0
      1004      1004         0         3         4    114197.84242    114197.84242         0
      1005      1005         0         3         4    114389.29356    114389.29356         0
      1006      1006         0         3         4    114391.28823    114391.28823         0
          Report FUTURE LIST   Count=  2
       TRAN     Assem        Pr   Current      Next       TimeStamp         EndTime Interrupt
      1008      1008         0         0         1    114553.70155    114553.70155         0
      1007      1007         0         6         7    114396.82311    114597.73990         0
         Report  IP LIST ( Integer ) Count= 7
       TRAN       Num   Integer
      1001         1         2
      1002         1         2
      1003         1         2
      1004         1         2
      1005         1         2
      1006         1         2
      1007         1         1
     Queue   Entries   Current       Max      Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
     QKass      1007         6        14       314         2.39465       272.26900
395.63475         3
  Facility   Entries   Current         Utility        AverTime    NextGo       TRAN      TRANP
NumObj
```

```
      Kass        421          1          0.53488        145.46597          0       1007       0
4
  Facility    Entries    Current          Utility         AverTime     NextGo        TRAN       TRANP
NumObj
      kass1       580          0          0.76752        151.51275          0          0       0
6
```

24 Блок проверки условий

Для явной проверки условий в GPSS используется блок TEST, который имеет следующий формат:

*:Test(Logs,NumBls);

Здесь Logs,NumBls – списки логических выражений и меток. Оба списка записываются через запятую в квадратных скобках. Просматриваются логические выражения, и для первого выражения со значением True происходит переход на метку с таким же порядковым номером. Если в списке все значения False- то заявка не входит в блок Test, а остается на выходе блока, в котором она находилась.

Блоки Test очень мощные, но они могут повлечь значительные расходы процессорного времени на безуспешные попытки заявки войти в блок. Чтобы уменьшить частоту безуспешных попыток вхождения в блок, можно поместить заявки в список пользователя, используя блоки Link и Unlink.

Пример:

Пусть имеется железнодорожный переезд со шлагбаумом. Автомобили с каждой стороны шлагбаума подъезжают со средним интервалом в 20 секунд, распределенным по экспоненциальному закону. Через шлагбаум одновременно могут переезжать два автомобиля. Время переезда – 15+-3 секунды. В среднем через 3600+- 900 секунд проходит поезд, и шлагбаум закрывается на 300+-60 секунд. Промоделировать работу переезда в течение 10 суток.

Текст модели может выглядеть следующим образом.

```
  {~vb} Var
{/PERE} PERE:Tstorage;
{/Tb} Tb:Ttable;
{/Tb0} Tb0:Ttable;
{/QPere} QPere:Tqueue;
{/G} G:Tgenerate;
{/G0} G0:Tgenerate;
{/G1} G1:Tgenerate;
{/G2} G2:Tgenerate;
{\Sys} Sys:Tparam;
  {~ve}
  {~pfe}
  {~ib} procedure Initial;begin
   SetStart(10);
{/PERE}  init(PERE,' PERE',2);
{/Tb}  init(Tb,' Tb',0,10,100);
{/Tb0}  init(Tb0,' Tb0',0,3,100);
{/QPere}  init(QPere,' QPere');
```

```
{/G}  init(G, ' G2',G_,Exponential(20));
{/G0}  init(G0,' G0',G0_,Exponential(20));
{/G1}  init(G1,' G1',G1_,3600,900);
{/G2}  init(G2,' G2',G2_,3600*24);
{\Sys} Init(Sys,'Sys');
  {~ie}end;
  {~mtb}procedure ModelTxt;begin case ActiveBlock of
{/G} ::G_   *:G.generate(Exponential(20));
::mm1    *: QPere.Queue ;
   *: test([not BP(1, sys.N)],[!*]) ;
       *: Pere.Enter ;
       *: QPere.Depart ;
       *: Advance(15,3) ;
       *: Pere.Leave ;
        *: Tb.Tabulate(TranAge);
        *: Terminate ;
{/G0} ::G0_   *:G0.generate(Exponential(20));
         *: Transfer(mm1);
{/G1} ::G1_   *:G1.generate(3600,900);
          *: BPlet(1,[True], sys.N) ;
          *: Advance(300,60) ;
       *: Tb0.tabulate(Qpere.L) ;
          *: BPlet(1,[False], sys.N) ;
            *: Terminate ;
{/G2} ::G2_   *:G2.generate(3600*24);
     *:toString(0,inttostr(term));
      *: Terminate(1)  ;
  {~mte} end;end;
  {~mb} procedure Simulation;  begin
   start(GetTerm);
  {~me}end;
  {~rb} procedure Report;begin
  {~re} end;
  {~cab} procedure CloseAllObj;begin
  {~cae} end;
```

Результирующая статистика может выглядеть следующим образом.

```
{/G} {::G_}   1:G.generate(Exponential(20));
{::mm1}    2: QPere.Queue ;
   3: test([not BP(1)],[4]) ;
        4: Pere.Enter ;
        5: QPere.Depart ;
        6: Advance(15,3) ;
        7: Pere.Leave ;
        8: Tb.Tabulate(TranAge);
        9: Terminate ;
{/G0} {::G0_}   10:G0.generate(Exponential(20));
        11: Transfer(mm1);

{/G1} {::G1_}   12:G1.generate(3600,900);
           13: BPlet(1,[True]) ;
           14: Advance(300,60) ;
           15: Tb0.tabulate(Qpere.L) ;
           16: BPlet(1,[False]) ;
```

```
              17: Terminate ;
          {/G2} {::G2_}   18:G2.generate(3600*24);
                19:toString(0,inttostr(term));
                20: Terminate(1)  ;

          StartTime         0.00000 EndTime     864000.00000
                  BLOCK Report
              Location   Entries   Current
                   1     43620         0
                   2     86808         0
                   3     86808        26
                   4     86782         0
                   5     86782         0
                   6     86782         2
                   7     86780         0
                   8     86780         0
                   9     86780         0
                  10     43188         0
                  11     43188         0
                  12       239         0
                  13       239         0
                  14       239         0
                  15       239         0
                  16       239         0
                  17       239         0
                  18        10         0
                  19        10         0
                  20        10         0
                    Report CURRENT LIST
                    TRAN     Assem        Pr   Current      Next       TimeStamp       TimeEnter
Interrupt
                   87031     87031         0         3         4    863732.40253    863732.40253
0
                   87035     87035         0         3         4    863740.18505    863740.18505
0
                   87036     87036         0         3         4    863744.55160    863744.55160
0
                   87037     87037         0         3         4    863753.16673    863753.16673
0
                   87038     87038         0         3         4    863768.72582    863768.72582
0
                   87034     87034         0         3         4    863769.91238    863769.91238
0
                   87039     87039         0         3         4    863775.91877    863775.91877
0
                   87040     87040         0         3         4    863795.69184    863795.69184
0
                   87042     87042         0         3         4    863801.61799    863801.61799
0
                   87043     87043         0         3         4    863807.48325    863807.48325
0
                   87044     87044         0         3         4    863869.58284    863869.58284
0
                   87041     87041         0         3         4    863869.71125    863869.71125
0
                   87046     87046         0         3         4    863874.68583    863874.68583
0

                    Report FUTURE LIST
                    TRAN     Assem        Pr   Current      Next       TimeStamp         EndTime
Interrupt
                   87032     87032         0         6         7    863729.27938    864001.65606
0
                   87059     87059         0         0        10    864002.81552    864002.81552
0
                   87033     87033         0         6         7    863731.43833    864010.30693
0
                   87060     87060         0         0         1    864020.20695    864020.20695
0
```

```
               87015     87015         0         0        12    867020.44932    867020.44932
0
               87061     87061         0         0        18    950400.00000    950400.00000
0
          SYS          Report BX LIST ( Boolean )  Count= 1
                Num          Boolean
                  1            FALSE
              Storage   Entries   Current       Max       Min        AverStor        AverTime
Utility
                PERE     86782         2         2         0         1.50694        15.00303
0.75347
                NextGo       TRAN            AC1    ACapac  Capacity    NumObj
                     0      87033    863994.00692     2.000         2         3
              Table        Tb  Entries     86780.00000
              Mean         76.97093 StdDev        90.37161    NumObj         4
                        Range       Frequency
                       10.00000         0.00000
                       20.00000     28726.00000
                       30.00000     15039.00000
                       40.00000      8212.00000
                       50.00000      4319.00000
                       60.00000      2659.00000
                       70.00000      1850.00000
                       80.00000      1374.00000
                       90.00000      1094.00000
                      100.00000      1072.00000
                      110.00000      1060.00000
                      120.00000      1052.00000
                      130.00000      1037.00000
                      140.00000       977.00000
                      150.00000       872.00000
                      160.00000       911.00000
                      170.00000       941.00000
                      180.00000      1044.00000
                      190.00000      1069.00000
                      200.00000      1157.00000
                      210.00000      1073.00000
                      220.00000       979.00000
                      230.00000       958.00000
                      240.00000       916.00000
                      250.00000      1009.00000
                      260.00000      1039.00000
                      270.00000       956.00000
                      280.00000       793.00000
                      290.00000       768.00000
                      300.00000       677.00000
                      310.00000       706.00000
                      320.00000       609.00000
                      330.00000       520.00000
                      340.00000       423.00000
                      350.00000       297.00000
                      360.00000       248.00000
                      370.00000       153.00000
                      380.00000        91.00000
                      390.00000        57.00000
                      400.00000        30.00000
                      410.00000        10.00000
                      420.00000         2.00000
                      430.00000         1.00000
                      440.00000         0.00000
              Table       Tb0  Entries       239.00000
              Mean        30.52301 StdDev         6.88306    NumObj         5
                        Range       Frequency
                        9.00000         0.00000
                       12.00000         1.00000
                       15.00000         4.00000
                       18.00000         5.00000
                       21.00000         7.00000
                       24.00000        26.00000
                       27.00000        45.00000
```

```
                30.00000        32.00000
                33.00000        43.00000
                36.00000        29.00000
                39.00000        23.00000
                42.00000        15.00000
                45.00000         5.00000
                48.00000         2.00000
                51.00000         2.00000
                54.00000         0.00000
              Queue   Entries   Current       Max       Zero       AverQueue        AverTime
AverTime(-Ze)    NumObj
              QPere     86808        26        51     21210         6.22865        61.99372
82.03834          6
```

Список текущих событий приводится не полностью.

Пример. Пусть заявки поступают в систему с интервалом 100+-50 и должны обслуживаться одним из устройств, 1 или 2. В устройстве 1 обслуживание занимает 150+-50 единиц времени, а в устройстве 2 – 170+-40. В первую очередь делается попытка занять устройство 1. После обслуживания в 1 или 2 устройстве, заявка обслуживается в устройстве 3 за время 90+-30 и покидает систему. Промоделировать обслуживание 1000 заявок.

Текст такой модели может выглядеть следующим образом:

```
  {~vb} Var
{/Fac1} Fac1:Tfacility;
{/Fac2} Fac2:Tfacility;
{/Fac3} Fac3:Tfacility;
{/Que} Que:Tqueue;
{/Que3} Que3:Tqueue;
{/Gen} Gen:Tgenerate;
  {~ve}
  {~pfe}
  {~ib} procedure Initial;begin
  setstart(1000);
{/Fac1}  init(Fac1,' Fac1');
{/Fac2}  init(Fac2,' Fac2');
{/Fac3}  init(Fac3,' Fac3');
{/Que}  init(Que,' Que');
{/Que3}  init(Que3,' Que3');
{/Gen}  init(Gen,' Gen',Gen_,100,50);
  {~ie}end;
  {~mtb}procedure ModelTxt;begin case ActiveBlock of
{/Gen} ::Gen_   *:Gen.generate(100,50);
    *: que.Queue ;
    *: test([fac1.L=0,fac2.L=0],[!*,tofac2]) ;
    *: fac1.seize ;
    *: que.depart ;
    *: advance(150,50)   ;
    *: fac1.release ;
    *: transfer(tofac3) ;
::tofac2   *: fac2.seize ;
    *: que.depart ;
```

```
    *: advance(170,40)   ;
    *: fac2.release ;
::tofac3    *: que3.Queue ;
    *: fac3.seize ;
    *: que3.depart ;
    *: advance(90,30)   ;
    *: fac3.release ;
    *: terminate(1) ;
  {~mte} end;end;
  {~mb} procedure Simulation;  begin
   start(GetTerm);
  {~me}end;
  {~rb} procedure Report;begin
  {~re} end;
  {~cab} procedure CloseAllObj;begin
  {~cae} end;
```

Результат моделирования имеет следующий вид:

```
{/Gen} {::Gen_}   1:Gen.generate(100,50);
     2: que.Queue ;
     3: test([fac1.L=0,fac2.L=0],[4,tofac2]) ;
     4: fac1.seize ;
     5: que.depart ;
     6: advance(150,50)   ;
     7: fac1.release ;
     8: transfer(tofac3) ;
{::tofac2}   9: fac2.seize ;
    10: que.depart ;
    11: advance(170,40)   ;
    12: fac2.release ;
{::tofac3}    13: que3.Queue ;
    14: fac3.seize ;
    15: que3.depart ;
    16: advance(90,30)   ;
    17: fac3.release ;
    18: terminate(1) ;

StartTime         0.00000 EndTime      98761.13929
          BLOCK Report
    Location   Entries   Current
         1      1003         0
         2      1003         0
         3      1003         0
         4       531         0
         5       531         0
         6       531         1
         7       530         0
         8       530         0
         9       472         0
        10       472         0
        11       472         1
        12       471         0
        13      1001         1
        14      1000         0
        15      1000         0
        16      1000         0
        17      1000         0
        18      1000         0
          Report CURRENT LIST
      TRAN     Assem        Pr   Current      Next       TimeStamp       TimeEnter Interrupt
     1001      1001         0        13        14     98560.69989     98675.20595         0
          Report FUTURE LIST
```

```
       TRAN     Assem        Pr   Current      Next       TimeStamp         EndTime Interrupt
       1003      1003         0         6         7     98684.22900     98787.34082         0
       1004      1004         0         0         1     98830.63372     98830.63372         0
       1002      1002         0        11        12     98631.51843     98857.58902         0
  Facility   Entries   Current         Utility        AverTime    NextGo       TRAN      TRANP
NumObj
      Fac1       531         1         0.81067       150.77688         0      1003      0
3
  Facility   Entries   Current         Utility        AverTime    NextGo       TRAN      TRANP
NumObj
      Fac2       472         1         0.80687       168.82951         0      1002      0
4
  Facility   Entries   Current         Utility        AverTime    NextGo       TRAN      TRANP
NumObj
      Fac3      1000         0         0.91514        90.38036         0         0      0
5
     Queue   Entries   Current       Max      Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
      Que       1003         0         2       712         0.09355         9.21188
31.75091         6
     Queue   Entries   Current       Max      Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
      Que3      1001         1         4       262         0.45489        44.88060
60.79227         7
```

По итоговой статистике видно, что первое устройство занято больше второго, так как заявки пытаются вначале войти именно в него.

Пример:

Пусть заявки поступают в систему с интервалом 100+-70 и обслуживаются одним из двух устройств. В первом устройстве – за 150+-90, а во втором – за 200+-80. вероятность попасть в первое устройство - 0.6 – а во второе – 0.4 . Промоделировать обслуживание 1000 заявок.

Тогда модель может выглядеть следующим образом.

```
  {~vb} Var
{/Fac1} Fac1:Tfacility;
{/Fac2} Fac2:Tfacility;
{/Gen} Gen:Tgenerate;
{/Que2} Que2:Tqueue;
{/Que1} Que1:Tqueue;
  {~ve}
  {~pfe}
  {~ib} procedure Initial;begin
  setstart(1000);
{/Fac1}  init(Fac1,' Fac1');
{/Fac2}  init(Fac2,' Fac2');
{/Gen}  init(Gen,' Gen',Gen_,100,70);
{/Que2}  init(Que2,' Que2');
{/Que1}  init(Que1,' Que1');
  {~ie}end;
  {~mtb}procedure ModelTxt;begin case ActiveBlock of
{/Gen} ::Gen_   *:Gen.generate(100,70);
    *: test([Random<=0.6,true],[!*,tofac2]) ;
     *: que1.Queue ;
      *: fac1.seize ;
```

```
    *: que1.depart ;
    *: advance(150,90)   ;
    *: fac1.release ;
    *: terminate(1) ;
::tofac2  *: que2.Queue ;
    *: fac2.seize ;
    *: que2.depart ;
    *: advance(200,80)   ;
    *: fac2.release ;
    *: terminate(1) ;
  {~mte} end;end;
  {~mb} procedure Simulation;  begin
   start(GetTerm);
  {~me}end;
  {~rb} procedure Report;begin
  {~re} end;
  {~cab} procedure CloseAllObj;begin
  {~cae} end;
```

Выходная статистика этой модели имеет вид:

```
{/Gen} {::Gen_}   1:Gen.generate(100,70);
     2: test([Random<=0.6,true],[3,tofac2]) ;
     3: que1.Queue ;
     4: fac1.seize ;
     5: que1.depart ;
     6: advance(150,90)   ;
     7: fac1.release ;
     8: terminate(1) ;
{::tofac2}  9: que2.Queue ;
    10: fac2.seize ;
    11: que2.depart ;
    12: advance(200,80)   ;
    13: fac2.release ;
    14: terminate(1) ;

StartTime         0.00000 EndTime     102920.46211
          BLOCK Report
    Location   Entries   Current
         1      1008         0
         2      1008         0
         3       616         8
         4       608         0
         5       608         0
         6       608         0
         7       608         0
         8       608         0
         9       392         0
        10       392         0
        11       392         0
        12       392         0
        13       392         0
        14       392         0
          Report CURRENT LIST
       TRAN     Assem        Pr   Current      Next       TimeStamp       TimeEnter Interrupt
       997       997         0         3         4    101780.74780    101780.74780         0
      1000      1000         0         3         4    102180.96983    102180.96983         0
      1002      1002         0         3         4    102352.99614    102352.99614         0
      1003      1003         0         3         4    102407.74908    102407.74908         0
      1004      1004         0         3         4    102463.98295    102463.98295         0
      1006      1006         0         3         4    102633.80366    102633.80366         0
      1007      1007         0         3         4    102781.42048    102781.42048         0
```

```
      1008      1008         0         3         4    102851.38858    102851.38858         0
          Report FUTURE LIST
       TRAN     Assem        Pr   Current      Next       TimeStamp         EndTime Interrupt
      1009      1009         0         0         1    103018.35305    103018.35305         0
  Facility   Entries   Current         Utility        AverTime    NextGo       TRAN      TRANP
NumObj
      Fac1       608         0         0.90266       152.80024         0          0      0
3
  Facility   Entries   Current         Utility        AverTime    NextGo       TRAN      TRANP
NumObj
      Fac2       392         0         0.75592       198.46763         0          0      0
4
     Queue   Entries   Current       Max      Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
      Que2       392         0         6       122         0.69226       181.75496
263.88127         6
     Queue   Entries   Current       Max      Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
      Que1       616         8        10        97         2.08426       348.23546
413.31993         7
```

Здесь можно видеть, что первое устройство действительно более занято, чем первое.

Пусть заявки поступают в одноканальное устройство с интервалом 100 единиц распределенным по экспоненциальному закону. Заявки обслуживаются в нем с интервалом 80+-30. Пусть обслуживание 15% заявок оказывается неудачным, и заявки нуждаются в повторном обслуживании. Промоделировать обслуживание 5000 заявок. Тогда, модель системы может выглядеть следующим образом.

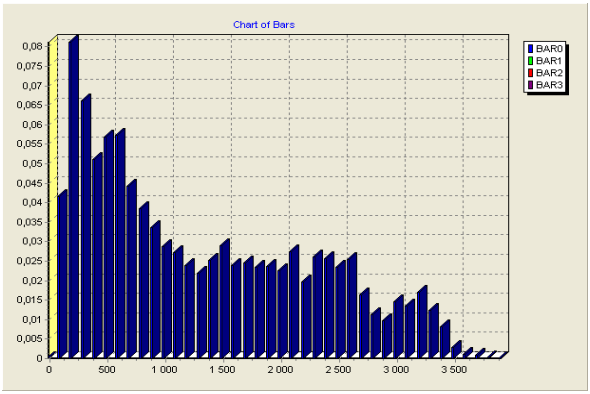
```
  {~vb} Var
{/Fac1} Fac1:Tfacility;
{/Que1} Que1:Tqueue;
{/Tb} Tb:Ttable;
{/G} G:Tgenerate;
  {~ve}
  {~pfe}
  {~ib} procedure Initial;begin
  setstart(5000);
{/Fac1}  init(Fac1,' Fac1');
{/Que1}  init(Que1,' Que1');
{/Tb}  init(Tb,' Tb',0,100,50);
{/G}  init(G,' G',G_,exponential(100));
  {~ie}end;
  {~mtb}procedure ModelTxt;begin case ActiveBlock of
{/G} ::G_   *:G.generate(exponential(100));
::ret     *: que1.Queue ;
      *: fac1.seize ;
     *: que1.depart ;
     *: advance(80,30)   ;
     *: fac1.release ;
    *: test([Random<=0.15,true],[ret,!*]) ;
     *: tb.tabulate(TranAge) ;
     *: terminate(1) ;
  {~mte} end;end;
```

```
{~mb} procedure Simulation;  begin
  start(GetTerm);
show(tb);
{~me}end;
{~rb} procedure Report;begin
{~re} end;
{~cab} procedure CloseAllObj;begin
{~cae} end;
```

Итоговая статистика имеет вид:

```
{/G} {::G_}   1:G.generate(exponential(100));
{::ret}     2: que1.Queue ;
       3: fac1.seize ;
      4: que1.depart ;
      5: advance(80,30)   ;
      6: fac1.release ;
      7: test([Random<=0.15,true],[ret,8]) ;
      8: tb.tabulate(TranAge) ;
      9: terminate(1) ;

StartTime         0.00000 EndTime     499500.50241
              BLOCK Report
          Location   Entries   Current
               1      5004         0
               2      5930         4
               3      5926         0
               4      5926         0
               5      5926         0
               6      5926         0
               7      5926         0
               8      5000         0
               9      5000         0
                Report CURRENT LIST
             TRAN      Assem         Pr    Current       Next        TimeStamp        TimeEnter
Interrupt
             5001       5001          0          2          3     499133.97061     499133.97061
0
             5002       5002          0          2          3     499258.75662     499258.75662
0
             5003       5003          0          2          3     499377.05088     499377.05088
0
             5004       5004          0          2          3     499487.66114     499487.66114
0
                Report FUTURE LIST
             TRAN      Assem         Pr    Current       Next        TimeStamp          EndTime
Interrupt
             5005       5005          0          0          1     499605.80435     499605.80435
0
          Facility    Entries    Current          Utility         AverTime     NextGo         TRAN
TRANP    NumObj
              Fac1       5926          0          0.95337         80.35950          0          0
0         3
            Queue    Entries    Current        Max       Zero        AverQueue          AverTime
AverTime(-Ze)    NumObj
             Que1       5930          4         39       1161         11.56101        973.81624
1210.88914         4
         Table        Tb  Entries      5000.00000
         Mean      1250.03923 StdDev       950.35433    NumObj         5
                  Range       Frequency
                0.00000         0.00000
              100.00000       208.00000
              200.00000       406.00000
              300.00000       331.00000
              400.00000       255.00000
              500.00000       284.00000
              600.00000       286.00000
```

700.00000	221.00000
800.00000	192.00000
900.00000	168.00000
1000.00000	144.00000
1100.00000	136.00000
1200.00000	120.00000
1300.00000	109.00000
1400.00000	126.00000
1500.00000	145.00000
1600.00000	120.00000
1700.00000	123.00000
1800.00000	117.00000
1900.00000	118.00000
2000.00000	113.00000
2100.00000	137.00000
2200.00000	99.00000
2300.00000	131.00000
2400.00000	128.00000
2500.00000	117.00000
2600.00000	127.00000
2700.00000	82.00000
2800.00000	56.00000
2900.00000	49.00000
3000.00000	73.00000
3100.00000	67.00000
3200.00000	85.00000
3300.00000	62.00000
3400.00000	41.00000
3500.00000	14.00000
3600.00000	5.00000
3700.00000	4.00000
3800.00000	1.00000
3900.0	0.00000



По результатам моделирования можно заметить, что некоторые заявки действительно многократно обслуживаются устройством.

Пример: пусть имеется 2 устройства обслуживающих заявки . Одно обслуживает за 11+-5 с, другое - за 11+-7с. Заявки становятся в очередь к тому устройству, где очередь короче.

Тогда модель может выглядеть следующим образом.

{~vb} Var
{/Fac1} Fac1:Tfacility;
{/Fac2} Fac2:Tfacility;

```
{/Gen} Gen:Tgenerate;
{/Que2} Que2:Tqueue;
{/Que1} Que1:Tqueue;
  {~ve}
  {~pfe}
  {~ib} procedure Initial;begin
  setstart(1000);
{/Fac1}  init(Fac1,' Fac1');
{/Fac2}  init(Fac2,' Fac2');
{/Gen}  init(Gen,' Gen',Gen_,5.6,5);
{/Que2}  init(Que2,' Que2');
{/Que1}  init(Que1,' Que1');
  {~ie}end;
  {~mtb}procedure ModelTxt;begin case ActiveBlock of
{/Gen} ::Gen_   *:Gen.generate(5.6,5);
    *: test([que1.L<=que2.L,true],[!*,tofac2]) ;
     *: que1.Queue ;
      *: fac1.seize ;
     *: que1.depart ;
     *: advance(11,5)   ;
     *: fac1.release ;
     *: terminate(1) ;
::tofac2  *: que2.Queue ;
     *: fac2.seize ;
     *: que2.depart ;
     *: advance(11,7)   ;
     *: fac2.release ;
     *: terminate(1) ;
  {~mte} end;end;
  {~mb} procedure Simulation;  begin
   start(GetTerm);
  {~me}end;
  {~rb} procedure Report;begin
  {~re} end;
  {~cab} procedure CloseAllObj;begin
  {~cae} end;
```

Пример: система должна обслуживать заявки, если длина очереди меньше 10. Если она равна 10,то последующие заявки теряются.

```
  {~vb} Var
{/Fac1} Fac1:Tfacility;
{/Que1} Que1:Tqueue;
{/G} G:Tgenerate;
  {~ve}
  {~pfe}
  {~ib} procedure Initial;begin
```

```
  setstart(5000);
{/Fac1}  init(Fac1,' Fac1');
{/Que1}  init(Que1,' Que1');
{/G}  init(G,' G',G_,exponential(100));
  {~ie}end;
  {~mtb}procedure ModelTxt;begin case ActiveBlock of
{/G} ::G_   *:G.generate(exponential(100));
    *: test([que1.L<10,true],[!*,exit]) ;
    *: que1.Queue ;
    *: fac1.seize ;
    *: que1.depart ;
    *: advance(110,30)   ;
    *: fac1.release ;
    *: terminate(1) ;
::exit     *: terminate ;
  {~mte} end;end;
  {~mb} procedure Simulation;  begin
   start(GetTerm);
  {~me}end;
  {~rb} procedure Report;begin
  {~re} end;
  {~cab} procedure CloseAllObj;begin
  {~cae} end;
```

Пример:

Пусть в систему поступают заявки с интервалом 100, распределенным по экспоненциальному закону. Пусть обслуживание заявок ведется набором устройств, каждое из которых имеет время обслуживания, равное в среднем 500 + 40#номер устройства. Отклонение времени обслуживания от среднего составляет 30%. Для обслуживания выбирается первое попавшееся свободное устройство из 15 имеющихся. Если таковое устройство не найдется, то использовать последнее. Определить общее и среднее время обслуживания 2000 заявок.

При решении задач, требующих массивов, разработчик модели непременно должен ответить себе на вопрос, как следует поступать, если в массиве не хватит элементов. Иначе модель может вести себя совершенно непредсказуемым образом. Впрочем, эта проблема есть в любых языках программирования.

Здесь понадобится использовать массивы устройств и очередей. Для создания массивов объектов следует вначале вставить простой объект. Затем в разделе констант (**Const**) следует определить размер массива, по образцу.

```
 num=15;
```

Здесь num – это имя константы, а 15 - количество элементов массива. Далее, в описании следует найти нужный объект и изменить его по образцу.

{/fac0} fac0 Tfacility; - заменить на

{/fac0} fac0:array[1..num] of Tfacility;

Аналогичным образом нужно изменить и описание очереди, получив.

{/que0} que0:array[1..num] of Tqueue;

Таким образом, мы получим описание 15 устройств и очередей.

Затем в процедуре initial строку
{/sh} procedure initial; begin - заменить на
{/sh} procedure initial; var i:integer; begin
а строки
{/fac0} .init(fac0);
{/que0} init(que0); - заменить на
{/fac0} for i:=1 to num do init(fac0[i], 'fac0'+inttostr(i));
{/que0} for i:=1 to num do init(que0[i], 'que0'+inttostr(i));

Таким образом, мы построим циклы создания 15 устройств и очередей.

Подобным образом нужно поступать всегда, когда вы создаете массивы объектов.

Доступ к элементам массива мы получаем, используя конструкцию NameObj[Num].Name

Здесь NameObj – имя объекта, Num – его номер, Name – имя процедуры или функции. Именно так мы работали с массивами в вышеуказанных процедурах.

В итоге, текст модели может выглядеть следующим образом.

```
Const
num=15;
  {~vb} Var
{/gen0} gen0: Tgenerate;
{/fac0} fac0:array[1..num] of Tfacility;
{/que0} que0:array[1..num] of Tqueue;
  {~ve}
  {~pfe}
  {~ib} procedure Initial;  var  i:integer; begin
    setstart(2000);
{/gen0}  init(gen0,' gen0',gen0_,exponential(100));
{/fac0}  for i:=1 to num do init(fac0[i], 'fac0'+inttostr(i));
{/que0}  for i:=1 to num do init(que0[i], 'que0'+inttostr(i));
  {~ie}end;
  {~mtb}procedure ModelTxt;begin case ActiveBlock of
{/gen0} ::gen0_   *:gen0.generate(exponential(100));
   *: IPlet(1,[0]);
::rret  *: IPlet(1,[ip(1)+1]);
      *: test([(fac0[IP(1)].L=0)or(IP(1)=num),true],[!*,rret]) ;
   *: que0[IP(1)].Queue;
   *: fac0[IP(1)].Seize;
   *: que0[IP(1)].Depart;
   *: Advance(500+40*ip(1),0.3*(500+40*ip(1)));
   *: fac0[IP(1)].Release;
   *: Terminate(1);
  {~mte} end;end;
  {~mb} procedure Simulation;  begin
    start(GetTerm);
  {~me}end;
```

{~rb} procedure Report; begin
{~re} end;
{~cab} procedure CloseAllObj;begin
{~cae} end;

В данном случае, выходная статистика представляет интерес. Она, в частности, свидетельствует, что разные устройства имеют разное время задержки, и что в данном случае последнее устройство – единственное, которое имеет непустую очередь.

```
 /gen0} {::gen0_}   1:gen0.generate(exponential(100));
   2: IPlet(1,[0]);
{::rret}  3: IPlet(1,[ip(1)+1]);
      4: test([(fac0[IP(1)].L=0)or(IP(1)=num),true],[5,rret]) ;
   5: que0[IP(1)].Queue;
   6: fac0[IP(1)].Seize;
   7: que0[IP(1)].Depart;
   8: Advance(500+40*ip(1),0.3*(500+40*ip(1)));
   9: fac0[IP(1)].Release;
   10: Terminate(1);

StartTime         0.00000 EndTime     199433.79704
          BLOCK Report
    Location   Entries   Current
         1      2004         0
         2      2004         0
         3      9702         0
         4      9702         0
         5      2004         0
         6      2004         0
         7      2004         0
         8      2004         4
         9      2000         0
        10      2000         0
          Report FUTURE LIST
       TRAN     Assem        Pr   Current      Next       TimeStamp         EndTime Interrupt
      2001      2001         0         8         9    198978.22111    199601.53134         0
      2005      2005         0         0         1    199659.20079    199659.20079         0
      2002      2002         0         8         9    199021.80918    199705.68081         0
      2003      2003         0         8         9    199211.09104    199744.82502         0
      2004      2004         0         8         9    199413.51028    199842.16001         0
          Report  IP LIST (  Integer  )
       TRAN       Num   Integer
      2001         1         5
      2002         1         6
      2003         1         1
      2004         1         2
  Facility   Entries   Current         Utility        AverTime    NextGo       TRAN      TRANP
NumObj
     fac01       312         1         0.83815       535.75369         0      2003      0
4
  Facility   Entries   Current         Utility        AverTime    NextGo       TRAN      TRANP
NumObj
     fac02       285         1         0.81734       571.94535         0      2004      0
5
  Facility   Entries   Current         Utility        AverTime    NextGo       TRAN      TRANP
NumObj
     fac03       248         0         0.77887       626.34361         0         0      0
6
  Facility   Entries   Current         Utility        AverTime    NextGo       TRAN      TRANP
NumObj
     fac04       225         0         0.73883       654.88017         0         0      0
7
  Facility   Entries   Current         Utility        AverTime    NextGo       TRAN      TRANP
NumObj
     fac05       199         1         0.68808       689.57614         0      2001      0
8
  Facility   Entries   Current         Utility        AverTime    NextGo       TRAN      TRANP
NumObj
```

```
     fac06       175         1         0.64784       738.29018         0      2002      0
9
  Facility   Entries   Current         Utility        AverTime    NextGo       TRAN      TRANP
NumObj
     fac07       139         0         0.54920       787.98323         0         0      0
10
  Facility   Entries   Current         Utility        AverTime    NextGo       TRAN      TRANP
NumObj
     fac08       120         0         0.49227       818.13465         0         0      0
11
  Facility   Entries   Current         Utility        AverTime    NextGo       TRAN      TRANP
NumObj
     fac09        94         0         0.39545       838.99719         0         0      0
12
  Facility   Entries   Current         Utility        AverTime    NextGo       TRAN      TRANP
NumObj
    fac010        75         0         0.33591       893.23227         0         0      0
13
  Facility   Entries   Current         Utility        AverTime    NextGo       TRAN      TRANP
NumObj
    fac011        53         0         0.24690       929.04338         0         0      0
14
  Facility   Entries   Current         Utility        AverTime    NextGo       TRAN      TRANP
NumObj
    fac012        35         0         0.17145       976.92086         0         0      0
15
  Facility   Entries   Current         Utility        AverTime    NextGo       TRAN      TRANP
NumObj
    fac013        24         0         0.11796       980.23869         0         0      0
16
  Facility   Entries   Current         Utility        AverTime    NextGo       TRAN      TRANP
NumObj
    fac014        13         0         0.06851      1051.00233         0         0      0
17
  Facility   Entries   Current         Utility        AverTime    NextGo       TRAN      TRANP
NumObj
    fac015         7         0         0.04141      1179.77596         0         0      0
18
     Queue   Entries   Current       Max      Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
     que01       312         0         1       312         0.00000         0.00000
0.00000        19
     Queue   Entries   Current       Max      Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
     que02       285         0         1       285         0.00000         0.00000
0.00000        20
     Queue   Entries   Current       Max      Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
     que03       248         0         1       248         0.00000         0.00000
0.00000        21
     Queue   Entries   Current       Max      Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
     que04       225         0         1       225         0.00000         0.00000
0.00000        22
     Queue   Entries   Current       Max      Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
     que05       199         0         1       199         0.00000         0.00000
0.00000        23
     Queue   Entries   Current       Max      Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
     que06       175         0         1       175         0.00000         0.00000
0.00000        24
     Queue   Entries   Current       Max      Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
     que07       139         0         1       139         0.00000         0.00000
0.00000        25
     Queue   Entries   Current       Max      Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
     que08       120         0         1       120         0.00000         0.00000
0.00000        26
```

```
     Queue   Entries   Current       Max      Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
     que09        94         0         1        94         0.00000         0.00000
0.00000        27
     Queue   Entries   Current       Max      Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
    que010        75         0         1        75         0.00000         0.00000
0.00000        28
     Queue   Entries   Current       Max      Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
    que011        53         0         1        53         0.00000         0.00000
0.00000        29
     Queue   Entries   Current       Max      Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
    que012        35         0         1        35         0.00000         0.00000
0.00000        30
     Queue   Entries   Current       Max      Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
    que013        24         0         1        24         0.00000         0.00000
0.00000        31
     Queue   Entries   Current       Max      Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
    que014        13         0         1        13         0.00000         0.00000
0.00000        32
     Queue   Entries   Current       Max      Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
    que015         7         0         2         3         0.02031       578.51905
1012.40833        33
```

Построим модель аналогичной системы, у которой допускаются очереди до 4-х заявок. Такая модель может иметь вид:

```
Const
num=15;
  {~vb} Var
{/gen0} gen0: Tgenerate;
{/fac0} fac0:array[1..num] of Tfacility;
{/que0} que0:array[1..num] of Tqueue;
  {~ve}
  {~pfe}
  {~ib} procedure Initial;  var  i:integer; begin
    setstart(2000);
{/gen0}  init(gen0,' gen0',gen0_,exponential(100));
{/fac0}  for i:=1 to num do init(fac0[i], 'fac0'+inttostr(i));
{/que0}  for i:=1 to num do init(que0[i], 'que0'+inttostr(i));
  {~ie}end;
  {~mtb}procedure ModelTxt;begin case ActiveBlock of
{/gen0} ::gen0_   *:gen0.generate(exponential(100));
   *: IPlet(1,[0]);
::rret  *: IPlet(1,[ip(1)+1]);
      *: test([(que0[IP(1)].L<4)or(IP(1)=num),true],[!*,rret]) ;
   *: que0[IP(1)].Queue;
   *: fac0[IP(1)].Seize;
   *: que0[IP(1)].Depart;
   *: Advance(500+40*ip(1),0.3*(500+40*ip(1)));
   *: fac0[IP(1)].Release;
   *: Terminate(1);
```

{~mte} end;end;
{~mb} procedure Simulation; begin
start(GetTerm);
{~me}end;
{~rb} procedure Report; begin
{~re} end;
{~cab} procedure CloseAllObj;begin
{~cae} end;

В итоговой статистике, число устройств, необходимых для этого случая уменьшается до 9.

```
{/gen0} {::gen0_}   1:gen0.generate(exponential(100));
   2: IPlet(1,[0]);
{::rret}  3: IPlet(1,[ip(1)+1]);
      4: test([(que0[IP(1)].q<4)or(IP(1)=num),true],[5,rret]) ;
   5: que0[IP(1)].Queue;
   6: fac0[IP(1)].Seize;
   7: que0[IP(1)].Depart;
   8: Advance(500+40*ip(1),0.3*(500+40*ip(1)));
   9: fac0[IP(1)].Release;
   10: Terminate(1);

StartTime         0.00000 EndTime     206294.31065
          BLOCK Report
    Location   Entries   Current
         1      2021         0
         2      2021         0
         3      7080         0
         4      7080         0
         5      2021        17
         6      2004         0
         7      2004         0
         8      2004         4
         9      2000         0
        10      2000         0
          Report CURRENT LIST
       TRAN     Assem        Pr   Current      Next       TimeStamp       TimeEnter Interrupt
      1999      1999         0         5         6    203639.39808    203639.39808         0
      2002      2002         0         5         6    203897.34082    203897.34082         0
      2003      2003         0         5         6    204026.00420    204026.00420         0
      2005      2005         0         5         6    204312.68093    204312.68093         0
      2006      2006         0         5         6    204597.70202    204597.70202         0
      2007      2007         0         5         6    204837.29949    204837.29949         0
      2008      2008         0         5         6    204976.38973    204976.38973         0
      2009      2009         0         5         6    205031.62440    205031.62440         0
      2010      2010         0         5         6    205032.57865    205032.57865         0
      2013      2013         0         5         6    205502.24972    205502.24972         0
      2014      2014         0         5         6    205560.64904    205560.64904         0
      2015      2015         0         5         6    205597.22028    205597.22028         0
      2016      2016         0         5         6    205713.26229    205713.26229         0
      2017      2017         0         5         6    205816.82673    205816.82673         0
      2018      2018         0         5         6    206003.05128    206003.05128         0
      2020      2020         0         5         6    206127.30821    206127.30821         0
      2021      2021         0         5         6    206135.89383    206135.89383         0
          Report FUTURE LIST
       TRAN     Assem        Pr   Current      Next       TimeStamp         EndTime Interrupt
      2022      2022         0         0         1    206363.27848    206363.27848         0
      1996      1996         0         8         9    203513.37390    206487.53302         0
      1997      1997         0         8         9    203580.47109    206569.69724         0
      2001      2001         0         8         9    203740.78578    206789.90783         0
      2019      2019         0         8         9    206031.00813    206976.56367         0
          Report  IP LIST (  Integer  )
       TRAN       Num   Integer
      1996         1         3
      1997         1         1
      1999         1         3
```

2001	1	4
2002	1	4
2003	1	1
2005	1	2
2006	1	1
2007	1	2
2008	1	3
2009	1	3
2010	1	4
2013	1	1
2014	1	2
2015	1	4
2016	1	3
2017	1	2
2018	1	1
2019	1	5
2020	1	4
2021	1	5

Facility	Entries	Current	Utility	AverTime	NextGo	TRAN	TRANP	NumObj
fac01	376	1	0.99967	548.47310	0	1997	0	4
fac02	354	0	0.99825	581.73347	0	0	0	5
fac03	328	1	0.99502	625.81679	0	1996	0	6
fac04	314	1	0.99134	651.30199	0	2001	0	7
fac05	279	1	0.94844	701.28573	0	2019	0	8
fac06	206	0	0.74246	743.52278	0	0	0	9
fac07	104	0	0.38485	763.38374	0	0	0	10
fac08	35	0	0.13890	818.66685	0	0	0	11
fac09	8	0	0.03467	894.09381	0	0	0	12
fac010	0	0	0.00000	0.00000	0	0	0	13
fac011	0	0	0.00000	0.00000	0	0	0	14
fac012	0	0	0.00000	0.00000	0	0	0	15
fac013	0	0	0.00000	0.00000	0	0	0	16

```
  Facility   Entries   Current         Utility        AverTime    NextGo       TRAN      TRANP
NumObj
    fac014         0         0         0.00000         0.00000         0          0      0
17
  Facility   Entries   Current         Utility        AverTime    NextGo       TRAN      TRANP
NumObj
    fac015         0         0         0.00000         0.00000         0          0      0
18
     Queue   Entries   Current       Max      Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
     que01       380         4         4         1         3.80097      2063.46835
2068.91285        19
     Queue   Entries   Current       Max      Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
     que02       358         4         4         1         3.73654      2153.14993
2159.18116        20
     Queue   Entries   Current       Max      Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
     que03       332         4         4         1         3.61875      2248.57774
2255.37103        21
     Queue   Entries   Current       Max      Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
     que04       318         4         4         2         3.35830      2178.61323
2192.40192        22
     Queue   Entries   Current       Max      Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
     que05       280         1         4         7         2.79314      2057.88541
2110.65170        23
     Queue   Entries   Current       Max      Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
     que06       206         0         4        22         1.73830      1740.77939
1948.91606        24
     Queue   Entries   Current       Max      Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
     que07       104         0         4        23         0.67125      1331.49715
1709.57658        25
     Queue   Entries   Current       Max      Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
     que08        35         0         4        10         0.20480      1207.10047
1689.94066        26
     Queue   Entries   Current       Max      Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
     que09         8         0         4         2         0.04442      1145.34413
1527.12551        27
     Queue   Entries   Current       Max      Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
    que010         0         0         0         0         0.00000         0.00000
0.00000        28
     Queue   Entries   Current       Max      Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
    que011         0         0         0         0         0.00000         0.00000
0.00000        29
     Queue   Entries   Current       Max      Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
    que012         0         0         0         0         0.00000         0.00000
0.00000        30
     Queue   Entries   Current       Max      Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
    que013         0         0         0         0         0.00000         0.00000
0.00000        31
     Queue   Entries   Current       Max      Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
    que014         0         0         0         0         0.00000         0.00000
0.00000        32
     Queue   Entries   Current       Max      Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
    que015         0         0         0         0         0.00000         0.00000
0.00000        33
```

Пример применения блока BUFFER в сочетании с GATE и TEST.

Пусть в систему поступают заявки, со средним интервалом 100 единиц, распределенным по экспоненциальному закону. Они должны группироваться по 10, и поступать на обслуживание 11 канальное устройство. Заявка, которая обнаружила 10 заявок, должна впустить их в устройство всех вместе, а сама остаться для формирования новой группы. Время обслуживания в многоканальном устройстве равно 900+-300. Если в момент попытки войти в многоканальное устройство, в нем останутся заявки, то все 10 заявок группы, должны быть удалены из системы. Промоделировать успешное обслуживание 1000 заявок.

Текст этой модели может быть, например, таким.

```
  {~vb} Var
  {/stor} stor:Tstorage;
  {/ng1} ng1:Tgenerate;
   num:Integer;
   key:Boolean;
{/ge} ge:Tgenerate;
  {~ve}
  {~pfe}
  {~ib} procedure Initial;begin
  setstart( 1000);
  {/stor}  init(stor,' stor',11);
  {/ng1}      init(ng1,' ng1',ng1_,exponential(100));
  Num:=0;
   Key:=false;
{/ge}  init(ge,' ge',ge_,10000);
  {~ie}end;
  {~mtb}procedure ModelTxt;begin  case ActiveBlock of
{/ng1} ::ng1_   *:ng1.generate(exponential(100));
    *:test([num=10,true],[!*,next]);
    *:let(key,true);
    *:buffer;
    *:let(key,false);
::next *:let( num,num+1);
    *:test([key],[!*]);
    *:test([stor.L=0,true],[!*,outl]);
    *:buffer;
    *:stor.enter;
    *:let( num,num-1);
    *:advance( 900,300);
    *:stor.leave;
    *:terminate( 1);
::outl *:let( num,num-1);
     *:tovalue(1,Total(!*-1));
     *: terminate;
{/ge} ::ge_   *:ge.generate(10000);
     *:currentblocks;
```

```
   *: terminate;
  {~mte} end;end;
  {~mb} procedure Simulation;  begin
 {/sh}    start(GetTerm);
  valueTitle(1,'stor');
  {~me}end;
  {~rb} procedure Report;begin
{/num}  reportval(num,'num = ');
{/key}  reportval(key,'key = ');
  {~re} end;
  {~cab} procedure CloseAllObj;begin
  {~cae} end;
```

В данной задаче блоки BUFFER нужно применить как для заявки, которая является инициатором впуска других заявок, так и для впущенных заявок, для корректной работы блока ***:test([stor.L=0,true],[!*,outl]);** . Когда заявка - инициатор впуска обрабатывается, то вначале она устанавливает ключ KEY в положение включено. Затем она становится последней в списке подвижных заявок, и поэтому задержанные заявки имеют возможность пройти через два блока Test. Они также ставятся в конец списка потенциально подвижных заявок.

Поэтому дальше движется заявка – инициатор, и выключает ключ. Она в результате оказывается первой задержанной заявкой в новой группе. А заявки прошедшие через два Test, получают возможность войти все одновременно в многоканальное устройство. Остальные детали работы модели представляются достаточно очевидными, и поэтому не обсуждаются.

Выходная статистика этой достаточно не простой задачи, имеет вид:

```
{/ng1} {::ng1_}   1:ng1.generate(exponential(100));
    2:test([num=10,true],[3,next]);
    3:let(key,true);
    4:buffer;
    5:let(key,false);
{::next} 6:let( num,num+1);
    7:test([key],[8]);
    8:test([stor.L=0,true],[9,outl]);
    9:buffer;
    10:stor.enter;
    11:let( num,num-1);
    12:advance( 900,300);
    13:stor.leave;
    14:terminate( 1);
{::outl} 15:let( num,num-1);
     16:tovalue(1,Total(17-1));
     17: terminate;
{/ge} {::ge_}   18:ge.generate(10000);
     19:currentblocks;
     20: terminate;

StartTime         0.00000 EndTime     164345.97771
         BLOCK Report
    Location   Entries   Current
         1      1674         0
         2      1674         0
         3       167         0
         4       167         0
         5       167         0
         6      1674         4
         7      1670         0
```

```
         8      1670         0
         9      1000         0
        10      1000         0
        11      1000         0
        12      1000         0
        13      1000         0
        14      1000         0
        15       670         0
        16       670         0
        17       670         0
        18        16         0
        19        16         0
        20        16         0
          Report CURRENT LIST
       TRAN     Assem        Pr   Current      Next       TimeStamp       TimeEnter Interrupt
      1688      1688         0         6         7    164103.21501    164103.21501         0
      1689      1689         0         6         7    164133.45789    164133.45789         0
      1690      1690         0         6         7    164261.90788    164261.90788         0
      1691      1691         0         6         7    164293.53983    164293.53983         0
          Report FUTURE LIST
       TRAN     Assem        Pr   Current      Next       TimeStamp         EndTime Interrupt
      1692      1692         0         0         1    164349.49341    164349.49341         0
      1648      1648         0         0        18    170000.00000    170000.00000         0
   Storage   Entries   Current       Max       Min        AverStor        AverTime
Utility
      stor      1000         0        10         0         5.48041       900.68289
0.49822
    NextGo       TRAN             AC1    ACapac  Capacity    NumObj
         0      1671    164345.97771    11.000        11         3
num  =           4
key  =  FALSE
```

Более сложный пример использования блока Test, представляет задача о работе порта. Здесь используются функции, результат которых зависит от номера, или от условия.

Модель порта, постановка задачи.

В порту есть три причала: 1, 2 и 3 (Berth1, Berth2 и Berth3).

В любой заданный момент времени к причалу 1 могут пришвартовываться 2 маленьких корабля или один средний.

И причал 2, и причал 3 могут обслуживать один большой корабль, два средних или четыре маленьких.

Время между приходами кораблей равно 26 часам, оно распределено экспоненциально, маленькие, средние и большие корабли приходят в порт в пропорции 5:3:2 соответственно. Очереди для причалов основываются на том, что первый прибывший обслуживается первым, за исключением того, что ни маленький, ни средний корабль не может пришвартовываться к причалу, который уже ждет большой корабль, у средних кораблей больший приоритет, чем у маленьких кораблей.

Время разгрузки экспоненциально распределено со средним временем для маленьких кораблей - 15 часов, для средних - 30 часов, для больших - 45 часов. Время погрузки следующее:

Время маленьких кораблей равномерно распределено в течение 24±6 часов.

Время средних кораблей равномерно распределено в течение 36±10 часов. Время больших кораблей равномерно распределено в течение 56±12 часов.

Большие корабли могут причалить или отчалить от причалов

2 и 3 только во время прилива. 2 и 3. Отлив длится 3 часа,
прилив - 10 часов. Необходимо:
1. Запустить процесс моделирования для 500 дней.
2. Определить распределение транзитного времени для
каждого типа кораблей.
3. Определить коэффициенты использования трех причалов. Решение данной задачи достаточно сложно, и поэтому текст модели пересыщен комментариями.
Листинг модели достаточно близок к решению на классическом GPSS, и, тем не менее, он заметно проще него. Решение основано на модели Джерарда Каммингса (Gerard F. Cummings)

При решении этой задачи причалы фигурируют в двух качествах. Как требования на занятие или освобождение причала, и как реальные причалы.

Причалы как требования занимаются до реальной возможности причалить в связи с возможным отливом. А освобождаются – только после реального ухода корабля с учетом возможного отлива. Остальные подробности построения модели более или менее очевидны.

```
 Const
// Константы нужны для того, чтобы использовать их имена в качестве
 // аргумента для P- параметров.
 Size=1;
 Capacity=2;
 Quenum=3;
 M_Unload=4;
 M_Load=5;
 Loadsp=6;
Berth_Num=7;
  {~vb} Var
{/Berth} Berth:ARRAY[1..3] OF Tstorage; // Реальный причал.
  Tide:Boolean;  // прилив есть.
{/tab} tab:ARRAY[1..3]  OF  Ttable;
{/gener0} gener0:Tgenerate;
{/ge1} ge1:Tgenerate;
{/ge2} ge2:Tgenerate;
{/que} que:ARRAY[1..3]  OF  Tqueue;
{/Berth1} Berth1:ARRAY[1..3] of Tstorage; // Причал, как заказ.
I:integer;
  {~ve}
  {~pfe}
  {~ib} procedure Initial;begin
   SETSTART(5000);
{/Berth} for i:=1 to 3 do  init(Berth[i], 'Berth '+inttostr(i),ibynum(i,[2,4,4])); // Задание
емкости реального причала.
  Tide:=false;
{/tab}  for i:=1 to 3 do init(tab[i], 'tab '+inttostr(i),0,10,100);
{/gener0}  init(gener0,' gener0',gener0_,24);
```

```
{/ge1}  init(ge1,' ge1',ge1_,0,0,0,1);
{/ge2}  init(ge2,' ge2',ge2_,Exponential(26));
{/que}  for i:=1 to 3 do  init(que[i], ' que '+inttostr(i));
{/Berth1}  for i:=1 to 3 do  init(Berth1[i] , 'Berth1 '+inttostr(i),ibynum(i,[2,4,4])); //
Задание емкости причала как заказа.
  {~ie}end;
  {~mtb}procedure ModelTxt;begin do case ActiveBlock of
// * Таймер дня
{/gener0} ::gener0_   *:gener0.generate(24); //;Одна заявка каждый день.
        *:TERMINATE( 1) ; //Часы работают один раз в день.
{/ge1} ::ge1_   *:ge1.generate(0);
::Again  *: LET(Tide,FALSE) ; // ;Циклическая заявка моделирует прилив.
       *:ADVANCE( 3) ; // Отлив длится 3 часа.
       *: LET(Tide,TRUE) ; // Прилив начинается.
       *:ADVANCE( 10) ; // Полная вода длится 10 часов.
       *:TRANSFER(Again) ; // Заявка возвращается к 'Again'.
{/ge2} ::ge2_   *:ge2.generate(Exponential(26));  //Каждые 26 часов появляется корабль.
       *:IPLET( Size,[IBYBOOL([RANDOM<=0.5,RANDOM<=0.6,true],[1,2,3])]); // тип
корабля в соответствии с вероятностями.
      *:IPLET( Capacity,[ibynum(Ip(size),[1,2,4] )]);  // емкость корабля
      *:PRIORITY(Ip(size)); // приоритет
      *:IPLET( Quenum,[Ip(size)]) ; // Очередь  для  корабля.
      *:RPLET( M_Unload,[ibynum(Ip(size),[15,30,45])]) ; //Среднее время разгрузки.
      *:RPLET( M_Load,[ibynum(Ip(size),[24,36,56])]) ; //Среднее время погрузки.
      *:RPLET( Loadsp,[ibynum(Ip(size),[6,10,12])]); // Разброс времени погрузки.
      *:que[ IP(Quenum)].QUEUE ;
*:test([(berth1[1].r>=IP(Capacity))or(berth1[2].r>=IP(Capacity))or(berth1[3].r>=IP(Capa
city))],[!*]); // Проверка, есть ли место для корабля на причале?
      *:IPLET( Berth_Num,[IBYBOOL([berth1[1].r>=IP(Capacity),
berth1[2].r>=IP(Capacity),berth1[3].r>=IP(Capacity)],[1,2,3])])  ; // Определяется
свободный причал.
      *:Berth1[IP(Berth_num)].ENTER (IP(Capacity)); // Занимаем причал с помощью
емкости корабля, как заказ.
        *:TEST([Ip(size)=3,true],[!*,Skipit1]) ; // Большой корабль?  Если
переключатель установлен, то прилив
      *:TEST([TIDE],[!*]) ; //Ожидается прилив.
::skipit1         *:Berth[IP(Berth_num)].ENTER (IP(Capacity)); // Занимаем реальный
причал с помощью емкости корабля.
       *:que[ IP(Quenum)].DEPART; // Выйти из очереди.
       *:ADVANCE(Exponential( RP(M_Unload))) ; // Время разгрузки.
       *:ADVANCE( RP(M_Load),RP(Loadsp)) ; // Время погрузки.
       *:Berth[IP(Berth_num)].LEAVE(IP(Capacity)) ;// Реальный причал
освобождается на емкость корабля.
       *:TEST([Ip(size)=3,true],[!*,Skipit]) ; // Большой корабль?  Если переключатель
установлен, то прилив
       *:TEST([TIDE],[!*]) ; //Ожидается прилив.
```

::Skipit *:Berth1[IP(Berth_num)].LEAVE(IP(Capacity)) ;// Причал, как заказ, освобождается на емкость корабля.

***: TAB[IP(Quenum)].TABULATE(M1);//Транзитное время сводится в таблицу по типу корабля.**

***:topoint(Ip(size)+3,aclock,que[Ip(size)].t); // график среднего времени в очереди.**

***:TERMINATE ;// Корабль отплывает.**

{~mte} end;end;

{~mb} procedure Simulation; begin

Line3d(false);

BarReset;

start(GetTerm);

show(tab[1],2);

show(tab[2],1);

show(tab[3],0);

ReportSave ('RPort.txt');

{~me}end;

{~rb} procedure Report; begin

{/Tide} reportval(Tide,'Tide = ');

{~re} end;

{~cab} procedure CloseAllObj;begin

{~cae} end;

Результирующая статистика по модели выглядит следующим образом.

```
// * Таймер дня
{/gener0} {::gener0_}   1:gener0.generate(24); //;Одна заявка каждый день.
          2:TERMINATE( 1) ; //Часы работают один раз в день.
{/ge1} {::ge1_}   3:ge1.generate(0);
{::Again}  4: LET(Tide,FALSE) ; // ;Циклическая заявка моделирует прилив.
        5:ADVANCE( 3) ; // Отлив длится 3 часа.
        6: LET(Tide,TRUE) ; // Прилив начинается.
        7:ADVANCE( 10) ; // Полная вода длится 10 часов.
        8:TRANSFER(Again) ; // Заявка возвращается к 'Again'.
{/ge2} {::ge2_}   9:ge2.generate(Exponential(26));  //Каждые 26 часов появляется корабль.
        10:IPLET( Size,[IBYBOOL([RANDOM<=0.5,RANDOM<=0.6,true],[1,2,3])]); // тип корабля в
соответствии с вероятностями.
       11:IPLET( Capacity,[ibynum(Ip(size),[1,2,4] )]);  // емкость корабля
       12:PRIORITY(Ip(size)); // приоритет
       13:IPLET( Quenum,[Ip(size)]) ; // Очередь  для  корабля.
       14:RPLET( M_Unload,[ibynum(Ip(size),[15,30,45])]) ; //Среднее время разгрузки.
       15:RPLET( M_Load,[ibynum(Ip(size),[24,36,56])]) ; //Среднее время погрузки.
       16:RPLET( Loadsp,[ibynum(Ip(size),[6,10,12])]); // Разброс времени погрузки.
       17:que[ IP(Quenum)].QUEUE ;
18:test([(berth1[1].r>=IP(Capacity))or(berth1[2].r>=IP(Capacity))or(berth1[3].r>=IP(Capacity))],
[19]); // Проверка,есть ли место для корабля на причале?
       19:IPLET(
Berth_Num,[IBYBOOL([berth1[1].r>=IP(Capacity),berth1[2].r>=IP(Capacity),berth1[3].r>=IP(Capacity
)],[1,2,3])])  ; // Определяется свободный причал.
     20:Berth1[IP(Berth_num)].ENTER (IP(Capacity)); // Занимаем причал с помощью емкости
корабля, как заказ.
        21:TEST([Ip(size)=3,true],[22,Skipit1]) ; // Большой корабль?  Если переключатель
установлен, то прилив
        22:TEST([TIDE],[23]) ; //Ожидается прилив.
{::skipit1}         23:Berth[IP(Berth_num)].ENTER (IP(Capacity)); // Занимаем реальный причал с
помощью емкости корабля.
        24:que[ IP(Quenum)].DEPART; // Выйти из очереди.
        25:ADVANCE(Exponential( RP(M_Unload))) ; // Время разгрузки.
        26:ADVANCE( RP(M_Load),RP(Loadsp)) ; // Время погрузки.
        27:Berth[IP(Berth_num)].LEAVE(IP(Capacity)) ;// Реальный причал освобождается на емкость
корабля.
```

```
        28:TEST([Ip(size)=3,true],[29,Skipit]) ; // Большой корабль?  Если переключатель
установлен, то прилив
        29:TEST([TIDE],[30]) ; //Ожидается прилив.
{::Skipit}  30:Berth1[IP(Berth_num)].LEAVE(IP(Capacity)) ;// Причал, как заказ, освобождается на
емкость корабля.
       31: TAB[IP(Quenum)].TABULATE(M1);//Транзитное время сводится в таблицу по типу корабля.
        32:topoint(Ip(size)+3,aclock,que[Ip(size)].t); // график среднего времени в очереди.
       33:TERMINATE ;// Корабль отплывает.

StartTime         0.00000 EndTime     120000.00000
          BLOCK Report
    Location   Entries   Current
         1      5000         0
         2      5000         0
         3         1         0
         4      9231         0
         5      9231         0
         6      9231         0
         7      9231         1
         8      9230         0
         9      4678         0
        10      4678         0
        11      4678         0
        12      4678         0
        13      4678         0
        14      4678         0
        15      4678         0
        16      4678         0
        17      4678         0
        18      4678         0
        19      4678         0
        20      4678         0
        21      4678         0
        22      1003         0
        23      4678         0
        24      4678         0
        25      4678         1
        26      4677         1
        27      4676         0
        28      4676         0
        29      1002         0
        30      4676         0
        31      4676         0
        32      4676         0
        33      4676         0
           Report FUTURE LIST
       TRAN     Assem        Pr   Current      Next       TimeStamp         EndTime Interrupt
       9676      9676         2        26        27    119955.61414    120002.13834         0
          2         2         0         7         8         0.00000    120003.00000         0
       9675      9675         3        25        26    119923.83613    120011.35741         0
       9681      9681         0         0         1    120024.00000    120024.00000         0
       9679      9679         0         0         9    120043.47498    120043.47498         0
            RP LIST ( Double )
       TRAN       Num          Double
       9675         4        45.00000
       9675         5        56.00000
       9675         6        12.00000
       9676         4        30.00000

       9676         5        36.00000
       9676         6        10.00000
           Report  IP LIST (  Integer  )
       TRAN       Num   Integer
       9675         1         3
       9675         2         4
       9675         3         3
       9675         7         2
       9676         1         2
       9676         2         2
       9676         3         2
```

```
      9676         7         3
   Storage   Entries   Current       Max       Min        AverStor        AverTime
Utility
    Berth1      2653         0         2         0         1.13576        51.37238
0.56788
    NextGo       TRAN             AC1    ACapac  Capacity    NumObj
         0      9672    119978.65924     2.000         2         3
   Storage   Entries   Current       Max       Min        AverStor        AverTime
Utility
    Berth2      3559         4         4         0         2.50259        84.38071
0.62565
    NextGo       TRAN             AC1    ACapac  Capacity    NumObj
         0      9675    119997.29152     4.000         4         4
   Storage   Entries   Current       Max       Min        AverStor        AverTime
Utility
    Berth3      2845         2         4         0         2.02643        85.47345
0.50661
    NextGo       TRAN             AC1    ACapac  Capacity    NumObj
         0      9670    119992.00270     4.000         4         5
Tide  =  TRUE
 Table      tab1  Entries      2305.00000
 Mean        46.18012 StdDev        33.06803    NumObj         6
           Range       Frequency
        10.00000         0.00000
        20.00000        25.00000
        30.00000       603.00000
        40.00000       705.00000
        50.00000       365.00000
        60.00000       213.00000
        70.00000       146.00000
        80.00000        58.00000
        90.00000        54.00000
       100.00000        27.00000
       110.00000        24.00000
       120.00000        25.00000
       130.00000         6.00000
       140.00000         9.00000
       150.00000         9.00000
       160.00000         5.00000
       170.00000         4.00000
       180.00000         6.00000
       190.00000         1.00000
       200.00000         0.00000
       210.00000         1.00000
       220.00000         0.00000
       230.00000         4.00000
       240.00000         2.00000
       250.00000         2.00000
       260.00000         1.00000
       270.00000         0.00000
       280.00000         1.00000
       290.00000         1.00000
       300.00000         1.00000
       310.00000         1.00000
       320.00000         0.00000
       330.00000         1.00000
       340.00000         0.00000
       350.00000         0.00000
       360.00000         0.00000
       370.00000         0.00000
       380.00000         2.00000
       390.00000         1.00000
       400.00000         1.00000
       410.00000         0.00000
       420.00000         1.00000
       430.00000         0.00000
 Table      tab2  Entries      1369.00000
 Mean        76.76202 StdDev        44.33237    NumObj         7
           Range       Frequency
        20.00000         0.00000
```

30.00000	11.00000
40.00000	129.00000
50.00000	259.00000
60.00000	207.00000
70.00000	173.00000
80.00000	133.00000
90.00000	114.00000
100.00000	74.00000
110.00000	71.00000
120.00000	40.00000
130.00000	37.00000
140.00000	14.00000
150.00000	22.00000
160.00000	10.00000
170.00000	12.00000
180.00000	14.00000
190.00000	8.00000
200.00000	7.00000
210.00000	4.00000
220.00000	2.00000
230.00000	5.00000
240.00000	4.00000
250.00000	3.00000
260.00000	4.00000
270.00000	1.00000
280.00000	1.00000
290.00000	1.00000
300.00000	0.00000
310.00000	1.00000
320.00000	4.00000
330.00000	0.00000
340.00000	2.00000
350.00000	1.00000
360.00000	0.00000
370.00000	0.00000
380.00000	1.00000
390.00000	0.00000

Table tab3 Entries 1002.00000
Mean 132.25468 StdDev 65.11521 NumObj 8

Range	Frequency
40.00000	0.00000
50.00000	4.00000
60.00000	42.00000
70.00000	96.00000
80.00000	84.00000
90.00000	83.00000
100.00000	68.00000
110.00000	75.00000
120.00000	76.00000
130.00000	67.00000
140.00000	50.00000
150.00000	61.00000
160.00000	33.00000
170.00000	29.00000
180.00000	38.00000
190.00000	21.00000
200.00000	29.00000
210.00000	24.00000
220.00000	15.00000
230.00000	17.00000
240.00000	8.00000
250.00000	19.00000
260.00000	12.00000
270.00000	11.00000
280.00000	7.00000
290.00000	4.00000
300.00000	5.00000
310.00000	3.00000
320.00000	3.00000
330.00000	6.00000

```
       340.00000         2.00000


       350.00000         0.00000
       360.00000         1.00000
       370.00000         2.00000
       380.00000         4.00000
       390.00000         0.00000
       400.00000         1.00000
       410.00000         0.00000
       420.00000         0.00000
       430.00000         1.00000
       440.00000         0.00000
       450.00000         1.00000
       460.00000         0.00000
     Queue   Entries   Current       Max      Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
      que1      2305         0         8      1938         0.14254         7.42080
46.60745        12
     Queue   Entries   Current       Max      Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
      que2      1370         0         6      1025         0.12083        10.58361
42.02768        13
     Queue   Entries   Current       Max      Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
      que3      1003         0         5       373         0.24963        29.86658
47.54949        14
   Storage   Entries   Current       Max       Min        AverStor        AverTime
Utility
   Berth_1      2653         0         2         0         1.13576        51.37238
0.56788
    NextGo       TRAN             AC1    ACapac  Capacity    NumObj
         0      9672    119978.65924     2.000         2        15
   Storage   Entries   Current       Max       Min        AverStor        AverTime
Utility
   Berth_2      3559         4         4         0         2.51120        84.67090
0.62780
    NextGo       TRAN             AC1    ACapac  Capacity    NumObj
         0      9675    119997.29152     4.000         4        16
   Storage   Entries   Current       Max       Min        AverStor        AverTime
Utility
   Berth_3      2845         2         4         0         2.03655        85.89999
0.50914
    NextGo       TRAN             AC1    ACapac  Capacity    NumObj
         0      9670    119992.00270     4.000         4        17
```

Если количество элементов в массивах объектов невелико, известно заранее и не может меняться, то можно «погрузить» одиночные объекты в массивы соответствующих типов. Как это делается на практике – демонстрирует пример модели порта, в которой эта техника используется. Ваша выгода при таком построении модели в том, что не нужно использовать циклы в процедуре **Initial.**

Пример модели с погружением объектов в массивы.

```
Const
Size=1;
Capacity=2;
Quenum=3;
M_Unload=4;
M_Load=5;
Loadsp=6;
Berth_Num=7;
 {~vb} Var
```

```
 Tide:Boolean;
{/gener0} gener0:Tgenerate;
{/ge1} ge1:Tgenerate;
{/ge2} ge2:Tgenerate;
{/BerthRe} BerthRe:Tstorage;
{/BerthRe0} BerthRe0:Tstorage;
{/BerthRe1} BerthRe1:Tstorage;
  BerthR:ARRAY[1..3] OF Tstorage;
{/BerthVir} BerthVir:Tstorage;
{/BerthVir0} BerthVir0:Tstorage;
{/BerthVir1} BerthVir1:Tstorage;
  BerthV:ARRAY[1..3] of Tstorage;
{/QBertch} QBertch:Tqueue;
{/QBertch0} QBertch0:Tqueue;
{/QBertch1} QBertch1:Tqueue;
 qBer:ARRAY[1..3]  OF  Tqueue;
{/Tabnum} Tabnum:Ttable;
{/Tabnum0} Tabnum0:Ttable;
{/Tabnum1} Tabnum1:Ttable;
  tab:ARRAY[1..3]  OF  Ttable;
  {~ve}
  {~pfe}
  {~ib} procedure Initial;begin
   SETSTART(5000);
   Tide:=false;
{/gener0}  init(gener0,' gener0',gener0_,24);
{/ge1}  init(ge1,' ge1',ge1_,0,0,0,1);
{/ge2}  init(ge2,' ge2',ge2_,Exponential(26));
{/BerthRe} init(BerthRe,' BerthRe',2);
{/BerthRe0} init(BerthRe0,' BerthRe0',4);
{/BerthRe1} init(BerthRe1,' BerthRe1',4);
{/BerthVir} init(BerthVir,' BerthVir',2);
{/BerthVir0} init(BerthVir0,' BerthVir0',4);
{/BerthVir1} init(BerthVir1,' BerthVir1',4);
{/QBertch} init(QBertch,' Berth');
{/QBertch0} init(QBertch0,' Berth0');
{/QBertch1} init(QBertch1,' Berth1');
{/Tabnum}  init(Tabnum,' Tabnum',0,10,100);
{/Tabnum0}  init(Tabnum0,' Tabnum0',0,10,100);
{/Tabnum1}  init(Tabnum1,' Tabnum1',0,10,100);
 BerthR[1]:=BerthRe;
 BerthR[2]:=BerthRe0;
 BerthR[3]:=BerthRe1;
 Berthv[1]:=Berthvir;
 Berthv[2]:=Berthvir0;
 Berthv[3]:=Berthvir1;
```

```
 qBer[1]:=QBertch;
 qBer[2]:=QBertch0;
 qBer[3]:=QBertch1;
 tab[1]:=Tabnum;
 tab[2]:=Tabnum0;
 tab[3]:=Tabnum1;
  {~ie}end;
  {~mtb}procedure ModelTxt;begin  case ActiveBlock of
// * Таймер дня
{/gener0} ::gener0_   *:gener0.generate(24); //;Один транзакт каждый день.
        *:TERMINATE( 1) ; //Часы работают один раз в день.
{/ge1} ::ge1_   *:ge1.generate(0);
::Again *: LET(Tide,FALSE) ; // ;Циклический транзакт моделирует прилив.
       *:ADVANCE( 3) ; // Отлив длится 3 часа.
       *: LET(Tide,TRUE) ; // Прилив начинается.
       *:ADVANCE( 10) ; // Полная вода длится 10 часов.
       *:TRANSFER(Again) ; // Транзакт возвращается к 'Again'.
{/ge2} ::ge2_   *:ge2.generate(Exponential(26));  //Каждые 26 часов появляется корабль.
       *:IPLET( Size,[IBYBOOL([RANDOM<=0.5,RANDOM<=0.6,true],[1,2,3])]); // тип
корабля
       *:IPLET( Capacity,[ibynum(Ip(size),[1,2,4] )]);  // емкость корабля
       *:PRIORITY(Ip(size)); // приоритет
       *:IPLET( Quenum,[Ip(size)) ; // Очередь  для  кораблей.
       *:RPLET( M_Unload,[ibynum(Ip(size),[15,30,45])]) ; //Среднее время разгрузки.
       *:RPLET( M_Load,[ibynum(Ip(size),[24,36,56])]) ; //Среднее время погрузки.
       *:RPLET( Loadsp,[ibynum(Ip(size),[6,10,12])]); // Разброс времени погрузки.
       *:qber[ IP(Quenum)].QUEUE ;
   *:test([(berthv[1].r>=IP(Capacity)) or(berthv[2].r>=IP(Capacity))
or(berthv[3].r>=IP(Capacity))],[!*]);
       *:IPLET( Berth_Num,[IBYBOOL([berthv[1].r>=IP(Capacity),berthv[2].r>=
IP(Capacity),berthv[3].r>=IP(Capacity)],[1,2,3])])  ;
       *:Berthv[IP(Berth_num)].ENTER (IP(Capacity)); // Занимаем причал с помощью
емкости корабля.
       *:TEST([Ip(size)=3,true],[!*,Skipit1]) ; // Большой корабль?  Если переключатель
установлен, то прилив
       *:TEST([TIDE],[!*]) ; //Ожидается прилив.
::skipit1 *:Berthr[IP(Berth_num)].ENTER (IP(Capacity)); // Занимаем причал с
помощью емкости корабля.
       *:qber[ IP(Quenum)].DEPART; // Выйти из очереди.
       *:ADVANCE(Exponential( RP(M_Unload))) ; // Время разгрузки.
       *:ADVANCE( RP(M_Load),RP(Loadsp)) ; // Время погрузки.
       *:Berthr[IP(Berth_num)].LEAVE(IP(Capacity)) ;// Причал освобождается на
емкость корабля.
       *:TEST([Ip(size)=3,true],[!*,Skipit]) ; // Большой корабль?  Если переключатель
установлен, то прилив
       *:TEST([TIDE],[!*]) ; //Ожидается прилив.
```

```
::Skipit  *:Berthv[IP(Berth_num)].LEAVE(IP(Capacity)) ;// Причал освобождается на емкость корабля.
      *: TAB[IP(Quenum)].TABULATE(M1);//Транзитное время сводится в таблицу по типу корабля.
      *:topoint(Ip(size),ac1,qber[Ip(size)].a*100);
 //   *:topoint(Ip(size)+3,aclock,qber[Ip(size)].t);
      *:TERMINATE ;// Корабль отплывает.
  {~mte} end;end;
  {~mb} procedure Simulation;  begin
//  Line3d(false);
  BarReset;
  start(GetTerm);
  show(tab[1],2);
  show(tab[2],1);
  show(tab[3],0);
  ReportSave ('RPort.txt');
  {~me}end;
  {~rb} procedure Report;begin
{/Tide}  reportval(Tide,'Tide = ');
  {~re} end;
  {~cab} procedure CloseAllObj;begin
  {~cae} end;
```

Обратите только внимание, что присваивания объектов элементам массивов – погружение должно вестись в конце процедуры **Initial,** например, как это сделано в данной модели, а описание массивов должно быть выполнено вручную и в дополнение к описаниям объектов.

```
BerthR[1]:=BerthRe;
BerthR[2]:=BerthRe0;
BerthR[3]:=BerthRe1;
Berthv[1]:=Berthvir;
Berthv[2]:=Berthvir0;
Berthv[3]:=Berthvir1;
qBer[1]:=QBertch;
qBer[2]:=QBertch0;
qBer[3]:=QBertch1;
tab[1]:=Tabnum;
tab[2]:=Tabnum0;
tab[3]:=Tabnum1;
```

25 Работа с прерываниями

Прерывания возможны только для одноканальных устройств. Если устройство в данный момент свободно, то занятие его по прерыванию ничем не лучше, чем обычное занятие устройства. Если устройство обслуживает заявку, то прерывание возможно, и тогда будет обслуживаться заявка, пришедшая по прерыванию.

Когда обслуживание прерывания завершено, то устройство может продолжить обслуживание прерванной заявки. Для моделирования прерываний используются следующие блоки.

*:Ustr.Extract(NumBl,NumRP);

Этот блок удаляет обслуживаемую заявку из устройства Ustr.

Здесь NumBl - номер или метка блока, куда удаляется заявка.

NumRP- номер RP - параметра для запоминания оставшегося времени задержки в блоке Advance для удаляемой заявки. По умолчанию, время не запоминается. Устройство остается незанятым, и, даже если были прерванные заявки, то они его не получают.

Поэтому после выполнения этого блока возможно занятие устройства по Seize или Preempt. Возможно также выполнение Return0, для передачи устройства прерванной ранее на устройстве заявке.

*:Ustr.Preempt;

Этот блок прерывает обслуживание заявки на устройстве.

Активная заявка в любом случае занимает устройство по прерыванию.

*:Ustr.Return;

Этот блок освобождает устройство, занятое по прерыванию. Ранее прерванная заявка, если она есть, получает устройство для продолжения обслуживания.

*:Ustr.Preempt0;

Этот блок прерывает обслуживание заявки на устройстве. Устройство остается незанятым. Чаще всего используется для моделирования недоступности.

*:Ustr.Return0;

Этот блок возвращает для обслуживания в устройстве, прерванную заявку. Это происходит после того, как устройство было оставлено незанятым. Чаще всего он используется для моделирования возврата устройства из недоступности.

Пример:

Пусть заявки поступают в систему со средним интервалом 100 единиц, распределенным по экспоненциальному закону. Приоритет заявок равномерно распределен в интервале 1.. 10. Время обработки любой заявки составляет 95+-15 единиц. Обслуживание заявок идет с прерываниями и учетом приоритета. Промоделировать обслуживание 1000 заявок. Протабулировать время обслуживания каждой заявки. Модель такой системы может выглядеть следующим образом:

```
  {~vb} Var
{/ng1} ng1:Tgenerate;
{/ust1} ust1:Tqueue;
{/ust} ust:Tfacility;
{/tt} tt:Ttable;
  {~ve}
  {~pfe}
  {~ib} procedure Initial;begin
{/ng1}  init(ng1,' ng1',ng1_,exponential(100));
{/ust1}  init(ust1,' ust1');
{/ust}  init(ust,' ust');
setstart(1000);
{/tt}  init(tt,'tt',0,100,100);
```

```
{~ie}end;
{~mtb}procedure ModelTxt;begin case ActiveBlock of
{/ng1} ::ng1_   *:ng1.generate(exponential(100));
*:priority (duniform(1,10));
*:ust1.queue;
*:test([prior(ust.Tran)<prior],[!*]);
*:ust.preempt;
*: Mark ;
*:ust1.depart;
*:advance( 95,15);
*:tt.tabulate(TranAge);
*:ust.return;
*:terminate(1);
{~mte} end;end;
{~mb} procedure Simulation;  begin
   start(GetTerm);
BARReset;
show(tt,2);
 {~me}end;
{~rb} procedure Report;begin
{~re} end;
{~cab} procedure CloseAllObj;begin
{~cae} end;
```

Итоговая статистика такой модели имеет следующий вид, который подтверждает наличие многоуровневых прерываний

```
{/ng1} {::ng1_}   1:ng1.generate(exponential(100));
  2:priority (duniform(1,10));
  3:ust1.queue;
  4:test([prior(ust.Tran)<prior],[5]);
  5:ust.preempt;
  6: Mark ;
  7:ust1.depart;
  8:advance( 95,15);
  9:tt.tabulate(TranAge);
  10:ust.return;
  11:terminate(1);

StartTime         0.00000 EndTime      99906.53211
          BLOCK Report
    Location   Entries   Current
         1      1004         0
         2      1004         0
         3      1004         3
         4      1001         0
         5      1001         0
         6      1001         0
         7      1001         0
         8      1001         1
         9      1000         0
        10      1000         0
        11      1000         0
          Report CURRENT LIST
      TRAN     Assem        Pr   Current      Next       TimeStamp       TimeEnter Interrupt
      1002      1002         5         3         4     99588.44878     99588.44878         0
      1003      1003         4         3         4     99802.45213     99802.45213         0
      1004      1004         2         3         4     99827.28511     99827.28511         0
          Report FUTURE LIST
```

```
       TRAN     Assem        Pr   Current      Next       TimeStamp         EndTime Interrupt
        989       989         1         8         9     99389.98670     99995.94076         0
       1005      1005         0         0         1    100264.19751    100264.19751         0
      Queue   Entries   Current       Max      Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
       ust1      1004         3        19       491         4.67291       464.99434
910.04741         4
  Facility   Entries   Current         Utility        AverTime    NextGo       TRAN      TRANP
NumObj
       ust      1001         1         0.94669        94.48575         0       989    989
5
 Table        tt  Entries      1000.00000
 Mean       228.40771 StdDev       452.81427    NumObj         6
          Range       Frequency
        0.00000         0.00000
      100.00000       471.00000
      200.00000       307.00000
      300.00000        88.00000
      400.00000        42.00000
      500.00000        20.00000
      600.00000        13.00000
      700.00000         7.00000
      800.00000         9.00000
      900.00000         6.00000
     1000.00000         4.00000
     1100.00000         4.00000
     1200.00000         3.00000
     1300.00000         5.00000
     1400.00000         2.00000
     1500.00000         1.00000
     1600.00000         1.00000
     1700.00000         2.00000
     1800.00000         1.00000
     1900.00000         0.00000
     2000.00000         1.00000
     2100.00000         1.00000
     2200.00000         2.00000
```

Информация о таблице приводится не полностью.

Другой пример:

Пусть основные заявки поступают в систему со средним интервалом 200 единиц, распределенным по экспоненциальному закону. Пусть также с интервалом 300+-200 в систему поступают заявки, которые занимают устройство по прерыванию. Обслуживаемые заявки, как основные, так и пришедшие по прерыванию в этом случае отправляются на обслуживание в другое устройство, которое работает втрое медленнее основного. Основное устройство обслуживает основные заявки за интервал времени равный 120+-20, а приоритетные заявки – за время 150+-100 единиц. Промоделировать обслуживание 1000 заявок, пришедших по прерыванию. Протабулировать времена обслуживания основных заявок, и заявок, пришедших по прерыванию. Модель такой системы может выглядеть следующим образом:

```
{~vb} Var
        {/ng1} ng1:Tgenerate;
        {/ust} ust:Tfacility;
        {/tt} tt:Ttable;
        {/Fac} Fac:Tfacility;
        {/Que} Que:Tqueue;
        {/Gen} Gen:Tgenerate;
        {/Que0} Que0:Tqueue;
```

```
{/Qust} Qust:Tqueue;
{/Tt1} Tt1:Ttable;
  {~ve}
  {~pfe}
  {~ib} procedure Initial;begin
{/ng1}  init(ng1,' (ng1',ng1_,exponential(200));
{/ust}  init(ust,' ust');
setstart(1000);
{/tt}  init(tt,'tt',0,80,100);
{/Fac}  init(Fac,' Fac');
{/Que}  init(Que,' Que');
{/Gen}  init(Gen,' gen',Gen_,300,200);
{/Que0}  init(Que0,' Que0');
{/Qust}  init(Qust,' Qust');
{/Tt1}  init(Tt1,' Tt1',0,10,100);
  {~ie}end;
  {~mtb}procedure ModelTxt;begin case ActiveBlock of
{/ng1} ::ng1_   *:ng1.generate(exponential(200));
  *:QUst.queue;
  *:ust.seize;
  *:QUst.depart;
  *: Mark ;
  *:advance( 120,20);
  *:ust.release;
  *:tt.tabulate(m1);
  *:terminate;
{/Gen} ::Gen_   *:Gen.generate(300,200);
  *:QUe.queue;
  *:ust.extract(next,1);
  *:ust.preempt;
  *: Mark ;
  *:QUe.depart;
  *:advance( 150,100);
  *:tt.tabulate(TranAge);
  *:ust.return;
 *:tt1.tabulate(TranAge );
  *:terminate(1);
::next *: Que0.queue ;
   *: fac.seize ;
   *: Que0.depart ;
   *: advance(3*RP(1)) ;
   *: Fac.Release ;
  *:tt.tabulate(TranAge);
  *:terminate;
  {~mte} end;end;
  {~mb} procedure Simulation;  begin
```

{/sh} start(GetTerm);
BARReset;
show(tt,2);
{~me}end;
{~rb} procedure Report;begin
{~re} end;
{~cab} procedure CloseAllObj;begin
{~cae} end;

Итоговая статистика этой модели имеет вид:

```
{/ng1} {::ng1_}   1:ng1.generate(exponential(200));
  2:QUst.queue;
  3:ust.seize;
  4:QUst.depart;
  5: Mark ;
  6:advance( 120,20);
  7:ust.release;
  8:tt.tabulate(TranAge);
  9:terminate;
{/Gen} {::Gen_}   10:Gen.generate(300,200);
  11:QUe.queue;
  12:ust.extract(next,1);
  13:ust.preempt;
  14: Mark ;
  15:QUe.depart;
  16:advance( 150,100);
  17:tt.tabulate(TranAge);
  18:ust.return;
 19:tt1.tabulate(TranAge);
  20:terminate(1);
{::next} 21: Que0.queue ;
   22: fac.seize ;
   23: Que0.depart ;
   24: advance(3*RP(1)) ;
   25: Fac.Release ;
  26:tt.tabulate(TranAge);
  27:terminate;

StartTime         0.00000 EndTime     349474.63939
          BLOCK Report
    Location   Entries   Current
         1      1732         0
         2      1732         0
         3      1732         0
         4      1732         0
         5      1732         0
         6      1732         0
         7       900         0
         8       900         0
         9       900         0
        10      1149         0
        11      1149         0
        12      1149         0
        13      1149         0
        14      1149         0
        15      1149         0
        16      1149         0
        17      1000         0
        18      1000         0
        19      1000         0
        20      1000         0
        21       981         0
        22       981         0
        23       981         0
        24       981         1
        25       980         0
        26       980         0
```

```
                 27       980         0
                   Report FUTURE LIST
                TRAN     Assem        Pr   Current      Next       TimeStamp          EndTime
Interrupt
                2881      2881         0        24        25    349293.00113    349515.68625
0
                2882      2882         0         0         1    349539.90319    349539.90319
0
                2883      2883         0         0        10    349656.38239    349656.38239
0
                    RP LIST ( Double )
                TRAN       Num          Double
                2881         1        48.56991
            Facility   Entries   Current         Utility        AverTime    NextGo       TRAN
TRANP    NumObj
                ust      2881         0         0.92059       111.67068         0          0
0         4
          Table        tt  Entries      2880.00000
          Mean       187.65110 StdDev       110.20851    NumObj         5
                     Range       Frequency
                   0.00000         0.00000
                  80.00000       186.00000
                 160.00000      1396.00000
                 240.00000       635.00000
                 320.00000       304.00000
                 400.00000       212.00000
                 480.00000        72.00000
                 560.00000        35.00000
                 640.00000        23.00000
                 720.00000        11.00000
                 800.00000         3.00000
                 880.00000         3.00000
                 960.00000         0.00000
            Facility   Entries   Current         Utility        AverTime    NextGo       TRAN
TRANP    NumObj
                Fac        981         1         0.51340       182.89371         0       2881
0         6
              Queue   Entries   Current       Max      Zero       AverQueue        AverTime
AverTime(-Ze)    NumObj
                Que       1149         0         1      1149         0.00000         0.00000
0.00000          7
              Queue   Entries   Current       Max      Zero       AverQueue        AverTime
AverTime(-Ze)    NumObj
               Que0        981         0         3       683         0.11295        40.23939
132.46592          9
              Queue   Entries   Current       Max      Zero       AverQueue        AverTime
AverTime(-Ze)    NumObj
               Qust       1732         0        22       137         3.59052       724.47797
786.70586         10
          Table       Tt1  Entries      1000.00000
          Mean       144.16466 StdDev        57.45456    NumObj        11
                     Range       Frequency
                  50.00000         0.00000
                  60.00000        58.00000
                  70.00000        58.00000
                  80.00000        70.00000
                  90.00000        45.00000
                 100.00000        49.00000
                 110.00000        62.00000
                 120.00000        42.00000
                 130.00000        48.00000
                 140.00000        59.00000
                 150.00000        49.00000
                 160.00000        45.00000
                 170.00000        50.00000
                 180.00000        50.00000
                 190.00000        55.00000
                 200.00000        48.00000
                 210.00000        52.00000
                 220.00000        32.00000
```

```
230.00000        41.00000
240.00000        51.00000
250.00000        36.00000
260.0             0.00000
```

По итоговой статистике, можно сделать вывод, что заявки, пришедшие по прерыванию, обслуживаются заметно быстрее.

26 Ввод и вывод данных при моделировании

По ходу моделирования возможен вывод данных. Это позволяет наблюдать ход моделирования. Наиболее наглядный вывод данных о ходе моделирования дают всевозможные графики, которых мы вскользь касались и раньше.

Для этих целей используются, в основном, блоки и процедуры из модуля AddModelUnit.

Прямой вывод очередной точки графика выполняет блок *:ToPoint, который имеет вид:

*:ToPoint(Num,X,Y);

Здесь Num- номер графика, X,Y – координаты очередной точки. Всего имеется 16 графиков, от 0 до 15.

Процедуры, описанные ниже, следует вызывать или в процедуре Simulation, или в процедуре Initial.

Перед выводом графиков полезно очистить содержимое вкладки Line, или хотя бы конкретного графика. Это выполняет процедура

LineClearProc (Num). Здесь Num- номер графика. Ее следует помещать, до выполнения процедуры Start. Если процедура вызвана без параметра, то она удаляет все графики.

Для задания цвета графика и его имени можно вызвать процедуры.

LineColor(Num,Col) Задает цвет графика с номером Num .

LineTitle(Num,Line); Задает имя графика с номером Num .

Для задания цвета можно воспользоваться функцией RGB(R,G,B). В этой функции R,G,B – это интенсивности красного, зеленого и голубого цветов соответственно. Интенсивность любого цвета должна лежать в диапазоне от 0 до 255. Так RGB(0,0,0) – определяет черный цвет, а RGB(255,255,255) – белый цвет.

Цвета также можно задавать с помощью предопределенных констант из следующего списка.

ClAqua – бирюзовый
clBlack - черный
clBlue - синий
clCream - кремовый
clDkGray Темно-серый
clFuchsia фуксиновый
clGray – темно серый
clGreen - зеленый
clLime – лимонно зеленый
clLtGray - светло-серый
clMaroon марон
clMedGray средне серый

clMoneyGreen цвет доллара
clNavy морской синий
clOlive оливковый
clPurple фиолетовый
clRed красный
clSilver серебряный
clSkyBlue голубой
clTeal темно бирюзовый
clWhite белый
clYellow желтый

Если вы хотите управлять видом графиков (плоские- объемные), то можно вызвать процедуру. Line3D(Yes3D);

Ее логический параметр Yes3D , переключает, плоские - объемные графики.

Если вы хотите сохранить графики в файле на диске, то вызовите процедуру. LineSave (FileName) .

PageColor(Page,Col); Задает цвет страницы. Список констант для SetPage, PageColor и ShowPage. spSimulation, ReportSetting, spReport, spLine, spValues, spBar, spAbout, spReserve ,spLicence

Процедура ToPointProc(Num,X,Y), подобна *:ToPoint(Num,X,Y), однако она не является блоком.

Аналогичным образом, блок *:LineClear (Num), очищающий график с данным номером, подобен процедуре LineClearProc (Num), но выполняется в ходе моделирования.

Для работы с гистограммами, в основном, используются процедуры, а не блоки.

Процедура BarClearProc (Num) выполняет сброс заданной гистограммы, по умолчанию, сбрасываются все гистограммы.

Блок *:BarClear (Num); выполняет ту же работу, что и процедура BarClearProc (Num), но выполняется в ходе моделирования.

Процедура Bar3D(Yes3D) переключает, плоские - объемные гистограммы.

Процедура BarColor(Num,Col) задает цвет гистограммы с номером Num.

Процедура BarTitle(Num,Line) задает имя гистограммы с номером Num.

Процедура BarSave (FileName) сохраняет гистограммы в файле на диске.

Процедура Bar(Num,X,Y) создает гистограмму, если X и Y – динамические массивы координат a, Num – номер гистограммы.

Для работы с выводом значений величин на вкладке Value, используются следующие процедуры и блоки.

Блок *:ToValue(Num,Value) выполняет вывод значения Value для строки Num в виде величины соответствующего столбика.

Процедура ValueColor(Num,Col) задает цвет столбика с номером Num.

Процедура ValueTitle(Num,Line) задает имя столбика с номером Num.

Процедура ToValueProc(Num,Value) , подобна *:ToValue(Num,Value), однако она не является блоком.

Для ввода и вывода текстов и манипуляций с файлами, используются следующие процедуры и блоки.

Блок *:ToString(Num,Line) выполняет вывод на AddMemo строки. Num – номер строки, Line - значение вещественного, целого, строкового или логического типа.

Блок *:CurrentBlocks выполняет вывод на AddMemo статистики по блокам.

Процедура AddMemo(Line) добавляет к AddMemo значение вещественного, целого, строкового или логического типа.

Процедура AddMemoClearProc очищает AddMemo.

Процедура AddLoad (FileName) загружает файл в AddMemo.

Процедура AddSave (FileName) сохраняет в файле содержимое AddMemo.

Процедура ToStringProc(Num,Line) , подобна *:ToString(Num,Line) , однако она не является блоком.

Процедура RepMemoAdd (Line) добавляет строку к RepMEMO.

Процедура ReportSave (FileName) сохраняет отчет по модели в файле.

Для ввода параметров из окна ввода, следует использовать функции.

RInput(Line) она вызывает диалог для ввода вещественного значения. Line - строка для заголовка.

IInput(Line) она вызывает диалог для ввода целого значения. Line - строка для заголовка.

SInput(Line) она вызывает диалог для ввода строкового значения. Line - строка для заголовка.

BInput(Line) она вызывает диалог для ввода логического значения. Line - строка для заголовка.

InputItem(Line) она вызывает диалог для выбора строки из выпадающего списка, Line - заголовок.

Для заполнения выпадающего списка используется процедура

NewItems(lines) она заполняет строками выпадающий список для InputItem. Список строк записывается в квадратных скобках через запятую.

Для ввода параметров из компонента AddMemo, следует использовать функции.

RGet(NumStr) она выполняет ввод вещественного значения из строки Numstr в AddMemo.

IGet(NumStr) она выполняет ввод целого значения из строки Numstr в AddMemo.

SGet(NumStr) она выполняет ввод строкового значения из строки Numstr в AddMemo.

BGet(NumStr) она выполняет ввод логического значения из строки Numstr в AddMemo.

FileNameGet Вызывает диалог открытия файла и возвращает его имя как строку.

FileNamePut Вызывает диалог закрытия файла и возвращает его имя как строку.

Приведем пример работы с рассмотреннными процедурами и блоками.

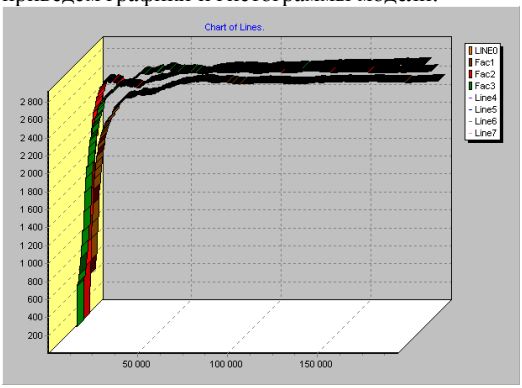
Пусть в систему поступают заявки с интервалом 100, распределенным по экспоненциальному закону. Пусть обслуживание заявок ведется набором устройств, каждое из которых имеет время обслуживания, равное в среднем 500 + 40*номер устройства. Отклонение времени обслуживания от среднего составляет 30%. Для обслуживания выбирается первое попавшееся устройство из 15 имеющихся устройств, у которого длина очереди меньше 5. Если таковое устройство не найдется, то использовать последнее. Определить общее и среднее время обслуживания 2000 заявок. Протабулировать время обслуживания в каждом из устройств. Вывести изменение значение среднего времени обслуживания для первых 3 устройств и гистограммы первых трех таблиц. В процессе моделирования отображать текущее состояние блоков.

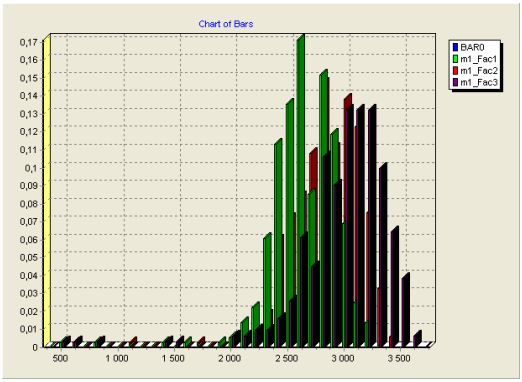
Модель в этом может иметь вид:

```
Const
num=15;
   {~vb} Var
{/gen0} gen0: Tgenerate;
{/fac0} fac0:array[1..num] of Tfacility;
{/que0} que0:array[1..num] of Tqueue;
{/Tab} Tab:array[1..num] of Ttable;
  {~ve}
  {~pfe}
  {~ib} procedure Initial;  var  i:integer; begin
    setstart(2000);
  {/gen0}  init(gen0,' gen0',gen0_,exponential(100));
{/fac0}  for i:=1 to num do init(fac0[i], 'fac0 '+inttostr(i));
{/que0}  for i:=1 to num do init(que0[i], ' que0 '+inttostr(i));
{/Tab}   for i:=1 to num do init(Tab[i], ' Tab '+inttostr(i),0,100,100);
  {~ie}end;
  {~mtb}procedure ModelTxt;begin case ActiveBlock of
{/gen0} ::gen0_   *:gen0.generate(exponential(100));
   *: IPlet(1,[0]);
::rret  *: IPlet(1,[ip(1)+1]);
      *: test([(que0[IP(1)].L<4)or(IP(1)=num),true],[!*,rret]) ;
   *: que0[IP(1)].Queue;
   *: fac0[IP(1)].Seize;
   *: que0[IP(1)].Depart;
   *: Advance(500+40*ip(1),0.3*(500+40*ip(1)));
   *: fac0[IP(1)].Release;
   *: tab[IP(1)].tabulate(TranAge) ;
   *: topoint(1,aclock, tab[1].a) ;
   *: topoint(2, aclock , tab[2].a) ;
   *: topoint(3, aclock, tab[3].a) ;
   *: currentblocks ;
   *: Terminate(1);
  {~mte} end;end;
  {~mb} procedure Simulation;  begin
    lineReset;
  lineTitle(1,'Fac1');
  lineTitle(2,'Fac2');
  lineTitle(3,'Fac3');
 barTitle(1,'m1_Fac1');
 barTitle(2,'m1_Fac2');
 barTitle(3,'m1_Fac3');
   barReset;
    start(GetTerm);
   show(tab[1],1);
   show(tab[2],2);
   show(tab[3],3);
```

{~me}end;
{~rb} procedure Report; begin
{~re} end;
{~cab} procedure CloseAllObj;begin
{~cae} end;

Вместо отчета, который в данном случае особого интереса не представляет, приведем графики и гистограммы модели.





27 Работа с семействами заявок

В Object GPSS имеются блоки, обеспечивающие работу с семействами.

Главный среди них - блок SPLIT, который имеет следующий формат:

*: Split(Num,NumBl,NumIP);

Блок создает Num копий заявок соответствующего семейства, и отправляет их по метке NumBl. NumIP – это номер IP - параметра для последовательной нумерации созданных заявок. Если параметр опущен, то заявки не нумеруются. При опущенном параметре Numbl – новые заявки переходят в следующий блок. После создания копий, заявка пытается перейти к следующему по номеру блоку.

Если, например, задан параметр NumIP как число 3, и в родительской заявке IP(3)=B, то параметру IP(3) копий будут присвоены значения B+1, B+2, B+3 и т.д.

Помимо значений всех P- параметров исходной заявки, в каждую копию записывается значение приоритета и отметка времени исходной заявки.

Каждая копия становится членом семейства заявок, порожденного исходной заявкой, которая была создана блоком GENERATE. Сама эта заявка также входит в семейство.

Если копия заявки входит в блок SPLIT, то вторичная копия становится членом того же семейства, что и первичная копия.

В модели одновременно может существовать произвольное число семейств, оно все время меняется, поскольку каждая генерируемая блоком GENERATE заявка создает новое семейство из одной заявки, а часть заявок удаляется из системы.

Узнать номер семейства активной заявки можно с помощью функции Assem.

Любую заявку можно принудительно включить в семейство с данным номером, с помощью блока Adopt, который имеет вид:

*: Adopt(NumA1,ATran); Блок устанавливает новый номер семейства NumA1 активной заявке (по умолчанию), или заявке с заданным номером ATran.

В блоке SPLIT в качестве номера семейства используется поле Assem, которое первоначально содержит номер TRAN родительской заявки.

Понятие семейства используется блокам Assemble, Gather, MATCH.

Блок Assemble имеет следующий формат:

*: Assemble(Num);

Он объединяет заданное число заявок, принадлежащих к одному семейству, в одну заявку (т.е. осуществляет сборку заданного числа заявок). После накопления, из блока Assemble выходит только одна заявка – инициатор сборки, которая переходит в следующий по номеру блок. В одном и том же блоке Assemble возможна одновременная сборка нескольких семейств.

Параметр Num - задает число заявок, участвующих в сборке.

Пример использования блоков SPLIT и ASSEMBLE

Пусть в систему передачи данных поступают информационные сообщения со средним интервалом 100, распределенным по экспоненциальному закону. Каждое сообщение разбивается на 3..17 блоков. Количество блоков распределено равномерно. Блок передается за 300+-290 единиц времени. На приемном конце заявки вновь собираются из блоков. Вывести в файл информацию о времени конца передачи сообщения, времени передачи, числе блоков, и номере семейства.

Тогда модель такой системы может иметь вид:

```
  {~vb} Var
{/G} G:Tgenerate;
{/G0} G0:Tgenerate;
{\Sys} Sys:Tparam;
  {~ve}
  {~pfe}
  {~ib} procedure Initial;begin
   setstart(1);
{/G}  init(G,'G',G_,exponential(100));
```

```
{/G0}  init(G0,'G0',G0_,2000);
{\Sys} Init(Sys,'Sys');
  {~ie}end;
  {~mtb}procedure ModelTxt;begin case ActiveBlock of
{/G} ::G_   *:G.generate(exponential(100));
    *:IPlet(1,[duniform(3,17)]) ;
    *: split(Ip(1)-1)  ;
    *: advance(300,290) ;
    *: assemble(Ip(1)) ;
   *: tostring(IP(1, sys.N),' ac1 '+Floattostr(aclock)+' M1 '+Floattostr(TranAge)
       +' P= '+inttostr(Ip(1))+' Assem '+inttostr(assem)) ;
   *: IPlet(1,[IP(1, sys.N)+1] , sys.N) ;
   *: terminate ;
{/G0} ::G0_   *:G0.generate(2000);
    *: terminate(1) ;
  {~mte} end;end;
  {~mb} procedure Simulation;  begin
   start(GetTerm);
  Addsave('test.txt');
  {~me}end;
  {~rb} procedure Report;begin
  {~re} end;
  {~cab} procedure CloseAllObj;begin
  {~cae} end;
```

Выходная статистика модели имеет вид:

```
{/G} {::G_}   1:G.generate(exponential(100));
     2:IPlet(1,[duniform(3,17)]) ;
     3: split(Ip(1)-1)  ;
     4: advance(300,290) ;
     5: assemble(Ip(1)) ;
     6: tostring(SYS.IP(1),' ac1 '+Floattostr(aclock)+' M1 '+Floattostr(TranAge)
         +' P= '+inttostr(Ip(1))+' Assem '+inttostr(assem)) ;
     7: SYS.IPlet(1,[IP(1)+1]) ;
     8: terminate ;
{/G0} {::G0_}   9:G0.generate(2000);
     10: terminate(1) ;
StartTime         0.00000 EndTime       2000.00000
          BLOCK Report
    Location   Entries   Current
         1        15         0
         2        15         0
         3        15         0
         4       152        25
         5       127         1
         6        12         0
         7        12         0
         8        12         0
         9         1         0
        10         1         0
          Report FUTURE LIST
       TRAN     Assem        Pr   Current      Next       TimeStamp         EndTime Interrupt
       151        133         0         4         5      1974.96368      2004.21867         0
       121        114         0         4         5      1440.19494      2004.69409         0
       122        114         0         4         5      1440.19494      2010.99943         0
       130        114         0         4         5      1440.19494      2012.41875         0
       152        133         0         4         5      1974.96368      2046.22612         0
       143        133         0         4         5      1974.96368      2065.73252         0
```

```
        138       138          0          0          1      2120.48197      2120.48197          0
        142       133          0          4          5      1974.96368      2135.81737          0
        135       118          0          4          5      1946.29034      2151.36096          0
        133       133          0          4          5      1974.96368      2166.67692          0
        140       133          0          4          5      1974.96368      2209.90289          0
        147       133          0          4          5      1974.96368      2266.87731          0
        145       133          0          4          5      1974.96368      2310.03752          0
        154       133          0          4          5      1974.96368      2328.03687          0
        150       133          0          4          5      1974.96368      2330.62835          0
        141       133          0          4          5      1974.96368      2380.37488          0
        149       133          0          4          5      1974.96368      2431.17076          0
        139       133          0          4          5      1974.96368      2446.55347          0
        134       118          0          4          5      1946.29034      2460.92422          0
        137       118          0          4          5      1946.29034      2476.56166          0
        136       118          0          4          5      1946.29034      2486.95230          0
        118       118          0          4          5      1946.29034      2497.93627          0
        144       133          0          4          5      1974.96368      2530.27187          0
        146       133          0          4          5      1974.96368      2530.46542          0
        148       133          0          4          5      1974.96368      2536.67099          0
        153       133          0          4          5      1974.96368      2554.26233          0
        155       155          0          0          9      4000.00000      4000.00000          0
           Report SINCHRONIZE LIST
        TRAN     Assem        Pr   Current      Next       TimeStamp       TimeEnter       Num
Interrupt
        120       114          0          5          6      1440.19494      1454.94825          3
0
           Report  IP LIST (  Integer  )
        TRAN       Num   Integer
        118          1         5
        120          1        15
        121          1        15
        122          1        15
        130          1        15
        133          1        17
        134          1         5
        135          1         5
        136          1         5
        137          1         5
        139          1        17
        140          1        17
        141          1        17
        142          1        17
        143          1        17
        144          1        17
        145          1        17
        146          1        17
        147          1        17
        148          1        17
        149          1        17
        150          1        17
        151          1        17
        152          1        17
        153          1        17
        154          1        17
SYS           Report IX LIST ( Integer ) Count= 1
        Num          Integer
          1             12
```

Информация в выходном файле имеет вид:

ac1 789,769981049556 M1 404,35008096043 P= 4 Assem 13
ac1 808,1871894559 M1 547,148886146024 P= 10 Assem 1
ac1 851,199071484527 M1 587,777505856939 P= 10 Assem 3
ac1 1003,65489975832 M1 569,149272418581 P= 13 Assem 23
ac1 1067,25797776677 M1 576,858882205561 P= 16 Assem 27
ac1 1129,99896060886 M1 510,118974801153 P= 11 Assem 40
ac1 1277,88119825782 M1 545,85183867719 P= 8 Assem 56
ac1 1279,70290431483 M1 432,21715667285 P= 6 Assem 75

ac1 1339,63332171602 M1 556,491720173508 P= 10 Assem 67
ac1 1434,84930499838 M1 577,870102417655 P= 15 Assem 85
ac1 1560,46875381844 M1 578,907188125886 P= 8 Assem 91
ac1 1639,93396763171 M1 548,462725789286 P= 4 Assem 106

Судя по информации из выходного файла, некоторые позднее поступившие на вход сообщения, на выходе появились раньше, что соответствует предполагаемым свойствам блока Assemble.

Рассмотрим модель системы с передачей информационных сообщений.

Пусть сообщения поступают в систему со средним интервалом времени в 750 единиц. Время распределено по экспоненциальному закону. Пусть число сегментов, на которые разбивается сообщение равномерно распределено в интервале от 3 до 12. Пусть на передачу одного сегмента по линии связи требуется 90+-20 единиц времени. После передачи каждого сегмента по линии связи, все они последовательно обрабатываются устройством, объединяясь в одно сообщение. Обработка в устройстве занимает 80+-50 единиц времени. Провести моделирование передачи и обработки 1000 сообщений. Текст такой модели может выглядеть следующим образом.

```
  {~vb} Var
{/G} G:Tgenerate;
{/QLine} QLine:Tqueue;
{/QUst} QUst:Tqueue;
{/Line} Line:Tfacility;
{/Ust} Ust:Tfacility;
  {~ve}
  {~pfe}
  {~ib} procedure Initial;begin
  setstart(1000);
{/G}  init(G,'G',G_,Exponential(750));
{/QLine}  init(QLine,' QLine');
{/QUst}  init(QUst,' QUst');
{/Line}  init(Line,' Line');
{/Ust}  init(Ust,' Ust');
  {~ie}end;
  {~mtb}procedure ModelTxt;begin case ActiveBlock of
{/G} ::G_   *:G.generate(Exponential(750));
        *: Qline.queue ;
        *: Line.seize ;
        *: Qline.Depart ;
        *: IPlet(1,[duniform(3,12)]) ;
        *: IPlet(2,[IP(1)]) ;
        *: advance(90,20) ;
        *: transfer(bbe2) ;
::bbc2   *: split(1,bg2)      ;
        *: advance(90,20) ;
::bbe2  *: loop(1,bbc2) ;
        *: Line.Release ;
```

```
::bg2   *: Qust.queue ;
        *: ust.seize ;
       *: Qust.Depart ;
        *: advance(80,50) ;
        *: Ust.Release ;
      *: Assemble(IP(2)) ;
      *: Terminate(1)  ;
  {~mte} end;end;
  {~mb} procedure Simulation;  begin
   start(GetTerm);
  {~me}end;
  {~rb} procedure Report;begin
  {~re} end;
  {~cab} procedure CloseAllObj;begin
  {~cae} end;
```

Итоговая статистика этой модели будет иметь вид:

```
  {/G} {::G_}   1:G.generate(Exponential(750));
           2: Qline.queue ;
           3: Line.seize ;
           4: Qline.Depart ;
           5: IPlet(1,[duniform(3,12)]) ;
           6: IPlet(2,[IP(1)]) ;
           7: advance(90,20) ;
           8: transfer(bbe2) ;
{::bbc2}    9: split(1,bg2)       ;
          10: advance(90,20) ;
{::bbe2}  11: loop(1,bbc2) ;
          12: Line.Release ;
{::bg2}   13: Qust.queue ;
         14: ust.seize ;
        15: Qust.Depart ;
         16: advance(80,50) ;
         17: Ust.Release ;
      18: Assemble(IP(2)) ;
      19: Terminate(1)  ;

StartTime         0.00000 EndTime     740114.06850
          BLOCK Report
    Location   Entries   Current
         1      1001         0
         2      1001         0
         3      1001         0
         4      1001         0
         5      1001         0
         6      1001         0
         7      1001         0
         8      1001         0
         9      6624         0
        10      6624         1
        11      7624         0
        12      1000         0
        13      7624         0
        14      7624         0
        15      7624         0
        16      7624         1
        17      7623         0
        18      7623         0
        19      1000         0
          Report FUTURE LIST
       TRAN     Assem        Pr   Current      Next       TimeStamp         EndTime Interrupt
      7618      7618         0        10        11    739827.47885    740122.92677         0
      7626      7618         0        16        17    739827.47885    740158.97073         0
```

```
      7624      7624         0         0         1    740850.84765    740850.84765         0
          Report  IP LIST (  Integer  )
       TRAN       Num   Integer
      7618         1         8
      7618         2         9
      7626         1         8
      7626         2         9
     Queue   Entries   Current       Max      Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
     QLine      1001         0        15        84         4.24945      3141.93382
3429.74455         4
     Queue   Entries   Current       Max      Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
      QUst      7624         0         4      2834         0.32901        31.93922
50.83603         5
  Facility   Entries   Current         Utility        AverTime    NextGo       TRAN      TRANP
NumObj
      Line      1001         1         0.92777       685.97179         0      7618      0
6
  Facility   Entries   Current         Utility        AverTime    NextGo       TRAN      TRANP
NumObj

       Ust      7624         1         0.82618        80.20290         0      7626      0
7
```

Блок Gather имеет следующий формат.

*: Gather(Num);

Здесь Num – число заявок семейства, которые необходимо собрать. После сборки заданного числа заявок, все они одновременно появляются на выходе блока. Блок во многом похож на блок ASSEMBLE.

Пример задачи на использование блока мог бы выглядеть следующим образом. Пусть в систему поступают электродвигатели для мойки и сушки в специальном аппарате с интервалом 70 минут , распределенном по экспоненциальному закону. Аппарат включают, только когда есть 5 электродвигателей. Процесс мойки и сушки занимает 280+-20 минут. Промоделировать процесс в течение 24000 минут.

Модель такой задачи может иметь вид.

```
  {~vb} Var
{/G} G:Tgenerate;
{/G0} G0:Tgenerate;
{/Qapp} Qapp:Tqueue;
{/app} app:Tstorage;
  {~ve}
  {~pfe}
  {~ib} procedure Initial;begin
    setstart(1);
{/G}  init(G,'G',G_,exponential(70));
{/G0}  init(G0,'G0',G0_,240000);
{/Qapp}  init(Qapp,' Qapp');
{/app}  init(app,' Qapp',5);
  {~ie}end;
  {~mtb}procedure ModelTxt;begin case ActiveBlock of
{/G} ::G_   *:G.generate(exponential(70));
        *: adopt(1) ;
       *: Qapp.Queue ;
```

```
        *: Gather(5) ;
       *: app.enter ;
       *: Qapp.depart ;
        *: Advance(280,20) ;
        *: Gather(5) ;
       *: app.Leave ;
       *: terminate()  ;
{/G0} ::G0_   *:G0.generate(240000);
       *: terminate(1)  ;
  {~mte} end;end;
  {~mb} procedure Simulation;  begin
   start(GetTerm);
  {~me}end;
  {~rb} procedure Report;begin
  {~re} end;
  {~cab} procedure CloseAllObj;begin
  {~cae} end;
```

Здесь блок Adopt позволяет все заявки считать принадлежащими одному (первому) семейству. Без него эта задача имела бы намного более громоздкое решение.

Выходная статистика имеет вид:

```
 {/G} {::G_}   1:G.generate(exponential(70));
          2: adopt(1) ;
         3: Qapp.Queue ;
          4: Gather(5) ;
         5: app.enter ;
         6: Qapp.depart ;
          7: Advance(280,20) ;
          8: Gather(5) ;
         9: app.Leave ;
         10: terminate()  ;
{/G0} {::G0_}   11:G0.generate(240000);
         12: terminate(1)  ;

StartTime         0.00000 EndTime     240000.00000
          BLOCK Report
    Location   Entries   Current
         1      3499         0
         2      3499         0
         3      3499         0
         4      3499         4
         5      3495         0
         6      3495         0
         7      3495         4
         8      3491         1
         9      3490         0
        10      3490         0
        11         1         0
        12         1         0
          Report FUTURE LIST
       TRAN     Assem        Pr   Current      Next       TimeStamp         EndTime Interrupt
      3492         1         0         7         8    239146.29976    240001.34231         0
      3494         1         0         7         8    239671.19660    240002.13641         0
      3493         1         0         7         8    239383.49182    240019.00937         0
      3496         1         0         7         8    239732.35898    240032.14865         0
      3501      3501         0         0         1    240041.03993    240041.03993         0
      3502      3502         0         0        11    480000.00000    480000.00000         0
          Report SINCHRONIZE LIST
       TRAN     Assem        Pr   Current      Next       TimeStamp       TimeEnter       Num
Interrupt
```

```
      3498          1          0          4          5    239841.48540    239841.48540          0
0
      3499          1          0          4          5    239945.41928    239945.41928          0
0
      3500          1          0          4          5    239948.00145    239948.00145          0
0
      3497          1          0          4          5    239812.08355    239812.08355          1
0
      3495          1          0          8          9    239689.85136    239998.77122          4
0
     Queue   Entries   Current       Max      Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
      Qapp      3499         4        23         0         4.15270       284.83827
284.83827         5
   Storage   Entries   Current       Max       Min        AverStor        AverTime
Utility
       app      3495         5         5         0         4.27377       293.47795
0.85475
    NextGo       TRAN             AC1    ACapac  Capacity    NumObj
         0      3496    239732.35898     5.000         5         6
```

Блок Match имеет следующий формат:

*: Match(NumBl);

Блок Match используется для синхронизации движения пар заявок, принадлежащих к одному семейству, без удаления этих заявок из модели.

Блоки Match используются парами, и, как правило, имеют метки. В поле NumBl указывается метка другого блока пары. То есть блоки MATCH ссылаются друг на друга. Такие блоки называются сопряженными. Когда заявка семейства входит в один из сопряженных блоков, а в другом нет заявки из этого семейства, то она задерживается в блоке. Если же в другом блоке есть заявка этого семейства, то обе они одновременно появляются на выходах сопряженных блоков. Одновременно в сопряженных блоках может синхронизироваться произвольное число пар заявок из различных семейств.

Пример использования блока Match.

Пусть в 8 часов утра должен начаться финальный конкурс лучших по профессии. Конкурсанты могут опоздать на 2.5+-2.5 минут, после чего начинается соревнование из 5 туров, которые выполняются подряд без подведения промежуточных итогов. Каждый тур выполняется первым участником за 10+-3 минуты, а вторым – за 9.8+-4 минуты. Когда оба участника завершат все 5 туров, происходит подведение итогов. Протабулировать время начала конкурса, и время ожидания каждого из конкурсантов, до завершения задания другим конкурсантом. Моделирование провести для 1000 таких конкурсов. Текст модели для этой задачи может иметь вид:

```
  {~vb} Var
{/beg} beg:Ttable;
{/wait1} wait1:Ttable;
{/wait2} wait2:Ttable;
{/ng1} ng1:Tgenerate;
  {~ve}
  {~pfe}
  {~ib} procedure Initial;begin
{/beg}  init(beg,' beg',0,0.8,30);
{/wait1}  init(wait1,' wait1',0,0.8,30);
{/wait2}  init(wait2,' wait2',0,0.8,30);
setstart( 1000);
```

```
{/ng1}     init(ng1,' ng1',ng1_,0);
  {~ie}end;
  {~mtb}procedure ModelTxt;begin case ActiveBlock of
{/ng1} ::ng1_   *:ng1.generate(0,0,0,1);
::mm5 *:split( 1,next);
  *:mark;
  *:advance( 2.5,2.5);
::mm1 *:match( mm2);
  *:beg.tabulate( TranAge);
  *:advance (10,3);
  *:advance (10,3);
  *:advance (10,3);
  *:advance (10,3);
  *:advance (10,3);
  *:mark;
::mm3 *:match (mm4);
  *:wait1.tabulate (TranAge);
  *:terminate( 1);
::next *:mark;
  *:advance (2.5,2.5);
::mm2 *:match( mm1);
  *:beg.tabulate( TranAge);
  *:advance (9.8,4);
  *:advance (9.8,4);
  *:advance (9.8,4);
  *:ADVANCE (9.8,4);
  *:advance( 9.8,4);
  *:mark;
::mm4 *:match( mm3);
  *:wait2.tabulate( TranAge);
  *:transfer (mm5);
  {~mte} end;end;
  {~mb} procedure Simulation;  begin
    start(GetTerm);
show(wait1,0);
show(wait2,1)
   {~me}end;
  {~rb} procedure Report;begin
  {~re} end;
  {~cab} procedure CloseAllObj;begin
  {~cae} end;
```

Итоговая статистика модели может иметь вид:

```
{/ng1} {::ng1_}   1:ng1.generate(0,0,0,1);
{::mm5} 2:split( 1,next);
3:mark;
4:advance( 2.5,2.5);
{::mm1} 5:match( mm2);
6:beg.tabulate( TranAge);
```

```
7:advance (10,3);
8:advance (10,3);
9:advance (10,3);
10:advance (10,3);
11:advance (10,3);
12:mark;
{::mm3} 13:match (mm4);
14:wait1.tabulate ( TranAge);
15:terminate( 1);
{::next} 16:mark;
17:advance (2.5,2.5);
{::mm2} 18:match( mm1);
19:beg.tabulate(  TranAge);
20:advance (9.8,4);
21:advance (9.8,4);
22:advance (9.8,4);
23:ADVANCE (9.8,4);
24:advance( 9.8,4);
25:mark;
{::mm4} 26:match( mm3);
27:wait2.tabulate(  TranAge);
28:transfer (mm5)

StartTime         0.00000 EndTime      55369.32051
          BLOCK Report
    Location   Entries   Current
         1         1         0
         2      1000         0
         3      1000         0
         4      1000         0
         5      1000         0
         6      1000         0
         7      1000         0
         8      1000         0
         9      1000         0
        10      1000         0
        11      1000         0
        12      1000         0
        13      1000         0
        14      1000         0
        15      1000         0
        16      1000         0
        17      1000         0
        18      1000         0
        19      1000         0
        20      1000         0
        21      1000         0
        22      1000         0
        23      1000         0
        24      1000         0
        25      1000         0
        26      1000         1
        27       999         0
        28       999         0
          Report CURRENT LIST
       TRAN     Assem        Pr   Current      Next       TimeStamp       TimeEnter Interrupt
      1001         1         0        26        27     55364.99570     55369.32051         0
 Table       beg  Entries      2000.00000
 Mean         3.33732 StdDev         1.19808    NumObj         3
           Range        Frequency
         0.00000          0.00000
         0.80000         50.00000
         1.60000        166.00000
         2.40000        236.00000
         3.20000        356.00000
         4.00000        468.00000
         4.80000        580.00000
         5.60000        144.00000
         6.40000          0.00000
 Table     wait1  Entries      1000.00000
```

```
Mean         2.06548 StdDev         3.33861    NumObj         4
          Range       Frequency
        0.00000       561.00000
        0.80000        43.00000
        1.60000        50.00000
        2.40000        48.00000
        3.20000        43.00000
        4.00000        32.00000
        4.80000        41.00000
        5.60000        30.00000
        6.40000        40.00000
        7.20000        21.00000
        8.00000        19.00000
        8.80000        16.00000
        9.60000        12.00000
       10.40000         8.00000
       11.20000         6.00000
       12.00000         8.00000
       12.80000         4.00000
       13.60000         7.00000
       14.40000         1.00000
       15.20000         1.00000
       16.00000         5.00000
       16.80000         3.00000
       17.60000         0.00000
       18.40000         0.00000
       19.20000         1.00000
       20.00000         0.00000
Table     wait2  Entries       999.00000
Mean         3.17154 StdDev         4.09404    NumObj         5
          Range       Frequency
        0.00000       439.00000
        0.80000        50.00000
        1.60000        46.00000
        2.40000        49.00000
        3.20000        42.00000
        4.00000        36.00000
        4.80000        39.00000
        5.60000        42.00000
        6.40000        38.00000
        7.20000        39.00000
        8.00000        29.00000
        8.80000        29.00000
        9.60000        28.00000
       10.40000        23.00000
       11.20000        17.00000
       12.00000        14.00000
       12.80000        12.00000
       13.60000         6.00000
       14.40000         3.00000
       15.20000         4.00000
       16.00000         6.00000
       16.80000         3.00000
       17.60000         2.00000
       18.40000         1.00000
       19.20000         0.00000
       20.00000         1.00000
       20.80000         0.00000
       21.60000         0.00000
       22.40000         1.00000
 1.000000E+0200         0.00000
```

Из выходной статистики видно, что первый конкурсант оказался вторым 561 раз и отставал от второго в среднем почти на 2 минуты. А второй конкурсант оказался вторым – 439 раз и отставал от первого в среднем на 3.337 минуты.

Среднее время начала конкурса оказалось на 3.342 смещенным относительно официального начала.

28 Списки пользователя

В GPSS имеется дополнительный тип списков заявок, названных списками пользователя, которые дают возможность удалять заявки из списков текущих событий и переводить их во временно неактивное состояние. Впоследствии эти заявки могут быть возвращены в список текущих событий.

Здесь, и далее ObjName – это имя списка.

Блок Link имеет вид.

*:ObjName.Link(Value,GR1); Блок помещает активную заявку в список. Выражение Value используется для упорядочения списка. Не обязательный целый параметр Gr1 также используется для упорядочения, причем он учитывается до Value. Это дает дополнительные возможности для формирования списка пользователя. Для удаления заявок из списка используется блок UnLink, он имеет вид.

*:ObjName.UnLink(NumBl,Num,Log,IPCount);

Здесь Numbl - метка, куда уйдут удаляемые заявки. Num- максимальное число удаляемых заявок. При Num=0 удаляются все заявки. При Num>0 заявки удаляются из начала списка, при Num<0 - с конца. Удалению подлежат только те заявки, для которых значение функции Log - истинно. Здесь Log – это отдельно описанная функция без параметров с результатом логического типа. Количество реально удаленных заявок, записывается в IP(IpCount). Если опущен параметр Log, то выбирать можно любые заявки. Если опущен IPCount, то количество реально удаленных заявок никуда не записывается.

*:ObjName.Select(Log ,Values) ; Блок записывает в динамический массив номера заявок из списка пользователя, для которых условие - функция Log - истинна.

*: ObjName.UnLink(NumBl, Num,Gr1, Log,<IPCount); Блок удаляет заявки из списка с Gr1=Gr. Количество удаленных заявок сохраняется в IP(IPCount) активной заявки.

Для эффективной работы со списками можно использовать следующие функции.

ObjName.A Определяет среднее число заявок в списке.

ObjName.V Определяет временной интеграл списка.

ObjName.T Определяет среднее время, проведенное заявкой в списке.

ObjName.C Определяет общее число занятий списка.

ObjName.M Определяет максимальное число заявок в списке.

ObjName.L Выдает текущее число заявок в списке.

ObjName.I Выдает минимальное число заявок в списке.

А вот более сложные функции. Их не рекомендуется вычислять внутри UnLink, а если нужно, то вычислять заранее и записывать в RP - параметры.

ObjName.Sum(Ssum,Log)

Определяет значение суммы значений функции SSUM для заявок из списка.

Учитываются только заявки удовлетворяющие условию - функции Log. Сами Ssum и Log – это параметры - функции и они должны быть без параметров. Их следует описать отдельно. Они должны быть вещественного и логического типа соответственно. Если параметр Log опущен, то учитываются все заявки.

ObjName.Avg(Ssum,Log)

Аналогична предыдущей функции, но вычисляет среднее значение.

ObjName.Min(Ssum,Log)

Аналогична предыдущей функции, но вычисляет минимальное значение.

ObjName.Max(Ssum,Log)

Аналогична предыдущей функции, но вычисляет максимальное значение.

Внутри функции Log для UnLink можно использовать следующие функции.

ObjName.Assem Выдает номер семейства для заявки из списка.

ObjName.TranAge Выдает время жизни для заявки из списка.

ObjName.Prior Выдает значение приоритета для заявки из списка.

ObjName.Tran Выдает значение номера заявки для заявки из списка.

ObjName.ScanNum Выдает порядковый номер заявки в списке в процессе просмотра с помощью UnLink. В начале просмотра значение функции равно 0.

ObjName.RP(Num) Выдает значение RP(Num) для заявки из списка.

ObjName.IP(Num) Выдает значение IP(Num) для заявки из списка.

ObjName.SP(Num) Выдает значение SP(Num) для заявки из списка.

ObjName.BP(Num) { Выдает значение BP(Num) для заявки из списка.

ObjName.Val Выдает значение параметра упорядочения заявок для заявки из списка.

Пример применения блоков Link и Unlink.

Рассмотрим модель работы одноканального устройства, с обслуживанием очереди по алгоритму LIFO:

Заявки поступают в систему с интервалом 1000, распределенным по экспоненциальному закону. Заявки обслуживаются в одноканальном устройстве с интервалом 800, распределенным по экспоненциальному закону. Порядок обслуживания - последним пришел – первым обслужен.

```
  {~vb} Var
{/tt} tt:Ttable;
{/och1} och1:Tqueue;
{/fac1} fac1:Tfacility;
{/list} list:Tuser;
{/ng1} ng1:Tgenerate;
  {~ve}
  {~pfe}
  {~ib} procedure Initial;begin
{/tt}  init(tt,'tt',0,2000,100);
{/och1}  init(och1,' och1');
{/fac1}  init(fac1,' fac1');
{/list}  init(list,' list');
{/ng1}      init(ng1,' ng1',ng1_,exponential(1000));
         setSTART( 1000 );
  {~ie}end;
  {~mtb}procedure ModelTxt;begin case ActiveBlock of
{/ng1} ::ng1_   *:ng1.generate(exponential(1000));
            *:och1.QUEUE;
            *:TEST([FAC1.L=0,TRUE],[!*+1,!*]);
             *:LIST.LINK(-AClock);
       ::next     *:FAC1.SEIZE;
```

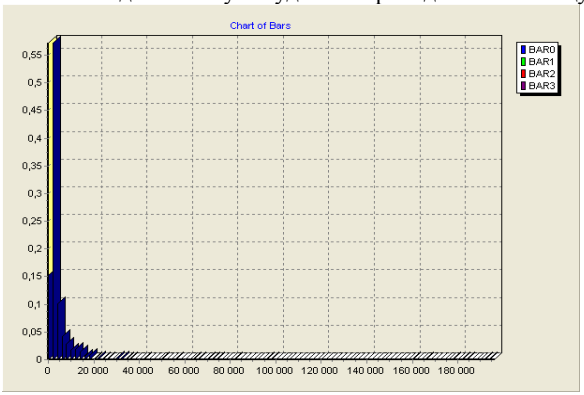
```
          *: OCH1.DEPART;
          *:tt.tabulate(TranAge);
          *: ADVANCE (exponential(800));
           *:FAC1.RELEASE;
          *:LIST. UNLINK(next,1);
         *: TERMINATE( 1) ;
  {~mte} end;end;
  {~mb} procedure Simulation;  begin
 {/sh}    start(GetTerm);
show(TT,0);
  {~me}end;
  {~rb} procedure Report;begin
  {~re} end;
  {~cab} procedure CloseAllObj;begin
  {~cae} end;
```

Выходная статистика этой модели - следующая.

```
Date Time Stamp= 29.04.2007 14:12:11
REPORT    AnatoliyGk@Sti.Lg.Ua : Product Num 816920
E:\work\ObjectClient\Object No Sys\models\book\tmp35.rtf
StartTime         0.00000 EndTime    1018195.01514
   Num      NameObject
     2              tt
     3            och1
     4            fac1
     5            list
     6             ng1

 {/ng1} ::ng1_   1:ng1.generate(exponential(1000));
              2:och1.QUEUE;
               3:test([Fac1.l=0,true],[next,4]);
              4:LIST.LINK(-AClock);
         ::next 5:FAC1.SEIZE;
              6: OCH1.DEPART;
              7:tt.tabulate(TranAge);
              8: ADVANCE (exponential(800));
               9:FAC1.RELEASE;
              10:LIST. UNLINK(next,1);
             11: TERMINATE( 1) ;
            BLOCK Report
    Location     Total   Current
          1      1010         0
          2      1010         0
          3      1010         0
          4       751        10
          5      1000         0
          6      1000         0
          7      1000         0
          8      1000         0
          9      1000         0
         10      1000         0
         11      1000         0
           Report  Current Transactions List  Count= 1
      Tran     Assem     Prior   Current      Next       TimeStamp       TimeEnter Inter
GenName
      1006      1006         0         4         5   1013549.68384   1013549.68384     0
ng1
           Report  Future Transactions List  Count= 1
      Tran     Assem     Prior   Current      Next       TimeStamp         EndTime Inter
GenName
      1011      1011         0         0         1   1019166.96247   1019166.96247     0
ng1
           Report  User List Count= 9
```

```
      Tran     Assem     Prior   Current      Next       TimeStamp       SortValue Inter
GenName          NameObj
      1003      1003         0         4         0   1010964.81422  -1010964.81422     0
ng1              list
       995       995         0         4         0   1006511.73831  -1006511.73831     0
ng1              list
       994       994         0         4         0   1006218.21101  -1006218.21101     0
ng1              list
       990       990         0         4         0   1002719.92202  -1002719.92202     0
ng1              list
       989       989         0         4         0   1002649.87746  -1002649.87746     0
ng1              list
       988       988         0         4         0   1002236.14001  -1002236.14001     0
ng1              list
       987       987         0         4         0   1002052.12819  -1002052.12819     0
ng1              list
       986       986         0         4         0   1001384.51100  -1001384.51100     0
ng1              list
       985       985         0         4         0   1001204.45182  -1001204.45182     0
ng1              list
     Queue   Entries   Current       Max       Min      Zero       AverQueue        AverTime
AverTime(-Ze)
      och1      1010        10        17         0       259         2.25818      2276.50077
3061.60556
  Facility   Entries   Current         Utility        AverTime    NextGo      Tran   IntTran
      fac1      1000         0         0.75389       767.60356         0         0         0
      User   Entries   Current       Max       Min        AverUser        AverTime
      list       751         9        17         0         2.25818      3061.60556
```

В данном случае удобнее приводить не таблицу из отчета, а гистограмму.



Анализируя выходную статистику, нетрудно видеть, что время ожидания в очереди некоторых заявок намного превосходит, ожидаемое время исходя из максимальной длины очереди. Это в свою очередь свидетельствует, что заявки действительно обслуживаются по дисциплине LIFO.

Хотя эта модель и не сложна, на ней можно проиллюстрировать несколько следующих важных моментов:

1) в этой системе активными, т.е. находящимися в списках текущих и будущих событий, списках прерываний или списках задержки, могут быть только те заявки, которые выходят из блока GENERATE, и та заявка, которая в данный момент занимает устройство. Все остальные заявки находятся в списке пользователя LIST.

2) поскольку все задержанные заявки, т.е. заявки, находящиеся в очереди к устройству FAC1, будут помещены в список пользователя LIST, интерпретатор не будет

тратить время на обработку этих заявок. Величина экономии времени зависит от длины очереди. Чем длиннее очередь, тем больше времени будет сэкономлено при помощи блоков LINK\UNLINK, применяемых для управления очередями к различным объектам.

3) пользователь имеет возможность формировать свои списки в динамике вне зависимости от списков задержки, которые управляются системой GPSS.

Списки пользователя моделируют специфические очереди, не подчиняющиеся принципу FIFO. Использование списков пользователя ускорит процесс моделирования при наличии очередей в системе.

Пример. Модель системы оперативной обработки данных в системе с разделением времени и кольцевым алгоритмом обслуживания и с учетом приоритетов. Объем памяти для заявки, ее приоритет и длительность обслуживания являются случайными величинами, рассчитываемыми по соответствующим формулам.

```
Const
Num_st=1;
prio=2;
num_ret=3;
Kvant=1;
  {~vb} Var
{/tt} tt:Ttable;
{/fac1} fac1:Tfacility;
{/list} list:Tuser;
{/ng1} ng1:Tgenerate;
{/Stor} Stor:Tstorage;
{/List1} List1:Tuser;
{\Sys} Sys:Tparam;
  {~ve}
  {~pfe}
  {~ib} procedure Initial;begin
{/tt}  init(tt,'tt',0,2000,100);
{/fac1}  init(fac1,' fac1');
{/list}  init(list,' list');
{/ng1}  init(ng1,' ng1',ng1_,exponential(150));
            setSTART( 1000 );
{/Stor}  init(Stor,' Stor',1024);
{/List1}  init(List1,' List1');
{\Sys} Init(Sys,'Sys');
rPLetProc(Kvant,[50], sys.N);
  {~ie}end;
  {~mtb}procedure ModelTxt;begin case ActiveBlock of
{/ng1} ::ng1_   *:ng1.generate(exponential(150));
          *: IPlet(num_st,[duniform(200,440)])  ;
          *: IPlet(prio,[duniform(0,4)])  ;
          *: IPlet(num_ret,[duniform(1,5)])  ;
::M8   *: TEST([ip(num_st)<=STOR.R,true],[!*,m4]) ;
        *: stor.Enter(ip(num_st)) ;
```

```
            *: mark ;
          ::m2   *:TEST([FAC1.L=0,TRUE],[!*+1,!*]);
             *: List.Link(IP(prio)) ;
          ::m3    *:FAC1.SEIZE;
                   *: ADVANCE( RP(Kvant), sys.N);
                    *:FAC1.RELEASE;
                 *: List.UnLink(m3,1) ;
                *: Loop(num_ret,m2) ;
                *: stor.Leave(ip(num_st)) ;
                    *:LIST1. UNLINK(m8,1);
                    *:tt.tabulate(TranAge);
                   *: TERMINATE( 1) ;
          ::m4 *:LIST1. LINK(ac1);
            {~mte} end;end;
            {~mb} procedure Simulation;  begin
           {/sh}    start(GetTerm);
          show(TT,0);
            {~me}end;
            {~rb} procedure Report;begin
            {~re} end;
            {~cab} procedure CloseAllObj;begin
            {~cae} end;
```

Выходная статистика модели выглядит следующим образом.

```
Date Time Stamp= 29.04.2007 9:39:19
REPORT    AnatoliyGk@Sti.Lg.Ua : Product Num 816920
E:\work\ObjectClient\Object No Sys\models\book\tmp36.rtf
StartTime         0.00000 EndTime     301654.18709
   Num      NameObject
     2              tt
     3            fac1
     4            list
     5             ng1
     6            Stor
     7           List1
     8             sys

 {/ng1} ::ng1_   1:ng1.generate(exponential(150));
            2: IPLet(num_st,[duniform(200,440)])  ;
            3: IPLet(prio,[duniform(0,4)])  ;
            4: IPLet(num_ret,[duniform(1,5)])  ;
::M8   5: nop    ;
          6: TEST([ip(num_st)<=STOR.R,true],[7,m4]) ;
           7: stor.Enter(ip(num_st)) ;
           8: mark ;
::m2    9:test([Fac1.l=0,true],[m3,10]);
         10: List.Link(-IP(prio)) ;
::m3    11: nop    ;
          12:FAC1.SEIZE;
               13: ADVANCE( sys.RP(Kvant));
                14:FAC1.RELEASE;
           15: List.UnLink(m3,1) ;
         16: Loop(num_ret,m2) ;
         17: stor.Leave(ip(num_st)) ;
               18:LIST1. UNLINK(m8,1);
               19:tt.tabulate(TranAge);
             20: priority(-1);
              21: TERMINATE( 1) ;
::m4 22:LIST1. LINK(aclock);
```

```
          BLOCK Report
    Location     Total   Current
         1      2011         0
         2      2011         0
         3      2011         0
         4      2011         0
         5      3950         0
         6      3950         0
         7      2003         0
         8      2003         0
         9      5974         0
        10      1045         0
        11      5974         2
        12      5972         0
        13      5972         1
        14      5971         0
        15      5971         0
        16      5971         0
        17      2000         0
        18      2000         0
        19      2000         0
        20      2000         0
        21      2000         0
        22      1947         8
          Report  Current Transactions List  Count= 2
      Tran     Assem     Prior   Current      Next       TimeStamp       TimeEnter Inter
GenName
      2004      2004         0        11        12    300418.21761    300418.21761     0
ng1
      2008      2008         0        11        12    301162.58116    301162.58116     0
ng1
          Report  Future Transactions List  Count= 2
      Tran     Assem     Prior   Current      Next       TimeStamp         EndTime Inter
GenName
      2003      2003         0        13        14    301654.18709    301704.18709     0
ng1
      2012      2012         0         0         1    301714.10245    301714.10245     0
ng1
          Report  User List Count= 8
      Tran     Assem     Prior   Current      Next       TimeStamp       SortValue Inter
GenName          NameObj
      1994      1994         0        22         0    298700.84605    300354.18709     0
ng1           List1
      2005      2005         0        22         0    300761.41150    300761.41150     0
ng1           List1
      2006      2006         0        22         0    300907.09573    300907.09573     0
ng1           List1
      2007      2007         0        22         0    300964.71788    300964.71788     0
ng1           List1
      1998      1998         0        22         0    299247.53531    301054.18709     0
ng1           List1
      2009      2009         0        22         0    301258.57155    301258.57155     0
ng1           List1
      2010      2010         0        22         0    301287.51104    301287.51104     0
ng1           List1
      2011      2011         0        22         0    301431.76495    301431.76495     0
ng1           List1
         Report  IP LIST ( Integer ) Count= 33
        Tran            Num/Name        Integer
        1994                   1            416
        1994                   2              0
        1994                   3              3
        1998                   1            410
        1998                   2              4
        1998                   3              1
        2003                   1            343
        2003                   2              1
        2003                   3              1
        2004                   1            299
        2004                   2              1
```

```
        2004                  3             2
        2005                  1           417
        2005                  2             1
        2005                  3             2
        2006                  1           385
        2006                  2             1
        2006                  3             4
        2007                  1           410
        2007                  2             2
        2007                  3             1
        2008                  1           207
        2008                  2             0
        2008                  3             5
        2009                  1           338
        2009                  2             2
        2009                  3             3
        2010                  1           259
        2010                  2             4
        2010                  3             2
        2011                  1           277
        2011                  2             1
        2011                  3             3
 Table tt  Entries      2000.00000 Mean       426.76560 StdDev       657.05801
           Range       Frequency        Integral
         0.00000         0.00000         0.00000
       200.00000       882.00000         0.44100
       400.00000       628.00000         0.75500
       600.00000       242.00000         0.87600
       800.00000        49.00000         0.90050
      1000.00000        24.00000         0.91250
      1200.00000        31.00000         0.92800
      1400.00000        19.00000         0.93750
      1600.00000        20.00000         0.94750
      1800.00000        16.00000         0.95550
      2000.00000        16.00000         0.96350
      2200.00000        11.00000         0.96900
      2400.00000         8.00000         0.97300
      2600.00000         9.00000         0.97750
      2800.00000         6.00000         0.98050
      3000.00000         5.00000         0.98300
      3200.00000         6.00000         0.98600
      3400.00000         5.00000         0.98850
      3600.00000         5.00000         0.99100
      3800.00000         4.00000         0.99300
      4000.00000         1.00000         0.99350
      4200.00000         0.00000         0.99350
      4400.00000         5.00000         0.99600
      4600.00000         1.00000         0.99650
      4800.00000         2.00000         0.99750
      5000.00000         1.00000         0.99800
      5200.00000         0.00000         0.99800
      5400.00000         0.00000         0.99800
      5600.00000         0.00000         0.99800
      5800.00000         0.00000         0.99800
      6000.00000         1.00000         0.99850
      6200.00000         0.00000         0.99850
      6400.00000         0.00000         0.99850
      6600.00000         0.00000         0.99850
      6800.00000         1.00000         0.99900
      7000.00000         2.00000         1.00000
      7200.00000         0.00000         1.00000
  Facility   Entries   Current         Utility        AverTime    NextGo      Tran   IntTran
      fac1      5972         1         0.98971        49.99163         0      2003         0
      User   Entries   Current       Max       Min        AverUser        AverTime
      list      1045         0         2         0         0.15567        44.93662
   Storage   Entries   Current       Max       Min        AverStor        AverTime
Utility
      Stor    639700       849      1024         0       835.28879       393.88520
0.81571
    NextGo      Tran          AClock    ACapac  Capacity
```

```
         0      2003    301654.18709  1024.000      1024
      User   Entries   Current       Max       Min        AverUser        AverTime
      List1     1947         8        46         0        14.91324      2310.55064
            Num/Name          Double sys          Report RP LIST ( DOUBLE ) Count= 1
R                1       50.00000
```

29 Блоки задания доступности и недоступности устройств

Для моделирования доступности и недоступности одноканальных и многоканальных устройств в самом простом случае, могут использоваться блоки

*:ObjName.SetGoto (NumBl);

Смысла этих блоков мы касались вскользь ранее. Положительное значение NumBl –определяет номер блока, отрицательное – запрещает вход в блок. По умолчанию – восстанавливается возможность входа в блок.

Это позволяет, задав в *:ObjName.SetGoto (NumBl) параметр равным –1, заблокировать вход в устройство, а вызвав его без параметра, вновь разрешить вход в устройство.

Пример использования блока.

Пусть в систему поступают заявки со средним интервалом 100 единиц, распределенным по экспоненциальному закону. Время обслуживания заявок составляет 48+-20 единиц. Устройство имеет интервалы доступности и недоступности, распределенные по нормальному закону со средним значением 2000, и отклонением в 200 единиц. Промоделировать работу системы за 20 циклов доступности и недоступности. Текст такой модели может иметь вид:

```
  {~vb} Var
{/gen0} gen0:Tgenerate;
{/fac0} fac0:Tfacility;
{/que0} que0:Tqueue;
{/g} g:Tgenerate;
  {~ve}
  {~pfe}
  {~ib} procedure Initial;begin
 setstart(20);
{/gen0}  init(gen0,' gen0',gen0_,Exponential(100));
{/fac0}  init(fac0,' fac0');
{/que0}  init(que0,' que0');
{/g}  init(g,'g',g_,0,0,0,1);
  {~ie}end;
  {~mtb}procedure ModelTxt;begin case ActiveBlock of
{/gen0} ::gen0_   *:gen0.generate(Exponential(100));
   *: que0.Queue;
   *: fac0.Seize;
   *: que0.Depart;
   *: Advance(48,20);
   *: fac0.Release;
   *: Terminate;
{/g} ::g_   *:g.generate(0,0,0,1);
```

```
::ret     *: fac0.Setgoto;
     *: advance(normal(2000,200)) ;
     *: fac0.Setgoto(-1);
     *: advance(normal(2000,200)) ;
     *: split(1,ret);
    *: Terminate(1);
  {~mte} end;end;
  {~mb} procedure Simulation;  begin
start(Getterm);
  {~me}end;
  {~rb} procedure Report;begin
  {~re} end;
  {~cab} procedure CloseAllObj;begin
  {~cae} end;
```

Выходная статистика модели подтверждает корректность расчетов среднего времени обслуживания и занятости устройства.

```
  {/gen0} {::gen0_}   1:gen0.generate(Exponential(100));
   2: que0.Queue;
   3: fac0.Seize;
   4: que0.Depart;
   5: Advance(48,20);
   6: fac0.Release;
   7: Terminate;
{/g} {::g_}   8:g.generate(0,0,0,1);
{::ret}     9: fac0.Setgoto;
      10: advance(normal(2000,200)) ;
      11: fac0.Setgoto(-1);
      12: advance(normal(2000,200)) ;
      13: split(1,ret);
     14: Terminate(1);
StartTime         0.00000 EndTime      82254.63232
          BLOCK Report
    Location   Entries   Current
         1       807         0
         2       807        17
         3       790         0
         4       790         0
         5       790         0
         6       790         0
         7       790         0
         8         1         0
         9        20         0
        10        20         0
        11        20         0
        12        20         0
        13        20         1
        14        20         0
          Report CURRENT LIST
       TRAN     Assem        Pr   Current      Next       TimeStamp       TimeEnter Interrupt
       811       811         0         2         3     80454.17663     80454.17663         0
       812       812         0         2         3     80632.81834     80632.81834         0
       813       813         0         2         3     80701.64896     80701.64896         0
       814       814         0         2         3     80835.80409     80835.80409         0
       815       815         0         2         3     81046.88068     81046.88068         0
       816       816         0         2         3     81048.61655     81048.61655         0
       817       817         0         2         3     81189.93120     81189.93120         0
       818       818         0         2         3     81422.75946     81422.75946         0
       819       819         0         2         3     81448.38931     81448.38931         0
       820       820         0         2         3     81478.58179     81478.58179         0
       821       821         0         2         3     81487.44253     81487.44253         0
       822       822         0         2         3     81601.36262     81601.36262         0
       823       823         0         2         3     81636.03647     81636.03647         0
```

```
       824       824         0         2         3     81680.06593     81680.06593         0
       825       825         0         2         3     81742.86540     81742.86540         0
       826       826         0         2         3     81895.40697     81895.40697         0
       827       827         0         2         3     81966.73047     81966.73047         0
       829         2         0        13         9         0.00000     82254.63232         0
         Report FUTURE LIST
       TRAN     Assem        Pr   Current      Next       TimeStamp         EndTime Interrupt
       828       828         0         0         1     82357.83318     82357.83318         0
  Facility   Entries   Current         Utility        AverTime    NextGo       TRAN      TRANP
NumObj
      fac0       790         0         0.45984        47.87852        -1          0      0
4
     Queue   Entries   Current       Max      Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
      que0       807        17        41        36        14.41499      1469.26865
1537.87263         5
```

Аналогичная задача для трехканального устройства при среднем интервале поступления заявок равном 33, решается следующим образом.

```
  {~vb} Var
{/gen0} gen0:Tgenerate;
{/que0} que0:Tqueue;
{/g} g:Tgenerate;
{/St} St:Tstorage;
  {~ve}
  {~pfe}
  {~ib} procedure Initial;begin
 setstart(20);
{/gen0}  init(gen0,' gen0',gen0_,Exponential(33.33));
{/que0}  init(que0,' que0');
{/g}  init(g,'g',g_,0,0,0,1);
{/St}  init(St,' St',3);
  {~ie}end;
  {~mtb}procedure ModelTxt;begin case ActiveBlock of
{/gen0} ::gen0_   *:gen0.generate(Exponential(33.33));
   *: que0.Queue;
   *: St.Enter;
   *: que0.Depart;
   *: Advance(48,20);
   *: St.Leave;
   *: Terminate;
{/g} ::g_   *:g.generate(0,0,0,1);
::ret      *: St.Setgoto;
     *: advance(normal(2000,200)) ;
     *: St.Setgoto(-1);
     *: advance(normal(2000,200)) ;
     *: split(1,ret);
    *: Terminate(1);
  {~mte} end;end;
  {~mb} procedure Simulation;  begin
start(Getterm);
  {~me}end;
```

{~rb} procedure Report;begin
{~re} end;
{~cab} procedure CloseAllObj;begin
{~cae} end;

Если обратиться к итоговой статистике, то можно убедиться, что модель работает правильно, то есть в модели корректно учитывается период недоступности. В этой статистике не приводится список текущих событий, как малоинформативный в данном случае.

```
 {/gen0} {::gen0_}   1:gen0.generate(Exponential(33.33));
   2: que0.Queue;
   3: St.Enter;
   4: que0.Depart;
   5: Advance(48,20);
   6: St.Leave;
   7: Terminate;
{/g} {::g_}   8:g.generate(0,0,0,1);
{::ret}      9: St.Setgoto;
      10: advance(normal(2000,200)) ;
      11: St.Setgoto(-1);
      12: advance(normal(2000,200)) ;
      13: split(1,ret);
     14: Terminate(1);

StartTime         0.00000 EndTime      80231.19099
          BLOCK Report
    Location   Entries   Current
         1      2495         0
         2      2495        66
         3      2429         0
         4      2429         0
         5      2429         0
         6      2429         0
         7      2429         0
         8         1         0
         9        20         0
        10        20         0
        11        20         0
        12        20         0
        13        20         1
        14        20         0
          Report FUTURE LIST
      TRAN     Assem        Pr   Current      Next       TimeStamp         EndTime Interrupt
      2516      2516         0         0         1     80274.99228     80274.99228         0
     Queue   Entries   Current       Max      Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
      que0      2495        66       127        48        55.27359      1777.42111
1812.28674         4
   Storage   Entries   Current       Max       Min        AverStor        AverTime
Utility
        St      2429         0         3         0         1.45113        47.93172
0.48371
    NextGo       TRAN             AC1    ACapac  Capacity    NumObj
        -1      2448     78347.13843     3.000         3         6
```

Для многоканальных устройств такого способа управления доступностью и недоступностью вполне достаточно. Однако, для одноканальных устройств, понятие недоступности более многогранное.

Поэтому вернемся к блокам

*:ObjName.Extract(NumBl,NumRP); Блок удаляет обслуживаемую заявку из устройства. NumBl - номер или метка блока, куда удаляется заявка. RP(NumRP) - параметра запоминает оставшееся время задержки заявки. Устройство остается незанятым, и, если были прерванные заявки, то они его не получают. Поэтому после

выполнения этого блока возможно занятие устройства по Seize или Preempt. Возможно также выполнение Return0, для передачи его прерванной ранее заявке. Если последний параметр опущен, то оставшееся время задержки заявки не запоминается.

*:ObjName.ExtractInterrupt(NumBl,NumRP); Блок удаляет прерванные заявки из устройства. NumBl - номер или метка блока, куда удаляются заявки. RP(NumRP) - определяет оставшееся время для продолжения обслуживания заявки. Если последний параметр опущен, то оставшееся время задержки заявки не запоминается.

*:ObjName.Preempt0; Блок прерывает обслуживание заявки на устройстве. Устройство остается незанятым. Используется для моделирования недоступности.

*:ObjName.Return0; Блок возвращает для обслуживания в устройстве, прерванную заявку. Это происходит после того, как устройство было оставлено незанятым. Используется для моделирования возврата из недоступности.

Рассмотрим моделирование различных способов задания недоступности одноканального устройства с помощью этих блоков.

Вариант А

Пусть заявка проходит через блоки

*:ObjName.Preempt0;

*:ObjName.SetGoto(-1);

Тогда устройство станет недоступным, в том числе и для той заявки, которая только что обслуживалась.

Когда мы решим, что устройство снова следует сделать доступным, то следует выполнить блоки

*:ObjName.Return0;

*:ObjName.SetGoto;

Вариант В

Пусть заявка проходит через блоки

*:ObjName.Extract(NumBl,NumRP);

*:ObjName.SetGoto(-1);

Тогда устройство станет недоступным, а заявка, которая только что обслуживалась, будет удалена из устройства.

Когда мы решим, что устройство снова следует сделать доступным, то следует выполнить блок

*:ObjName.SetGoto;

Вариант С

Пусть заявка проходит через блоки

*:ObjName.Extract(NumBl,NumRP);

*:ObjName.ExtractInterrupt(NumBl,NumRP);

*:ObjName.SetGoto(-1);

Тогда устройство станет недоступным, а заявка, которая только что обслуживалась, будет удалена из устройства. Из него будут также удалены заявки, которые были ранее прерваны на этом устройстве.

Когда мы решим, что устройство снова следует сделать доступным, то следует выполнить блок

*:ObjName.SetGoto;

Отметим, что организацию доступности – недоступности должна выполнять не та заявка, которая сама претендует на устройство.

Вероятно, это далеко не все способы организации недоступности для устройства, которые обеспечивают эти блоки. Эти блоки обеспечивают и более гибкое управление устройством, не связанное с прерыванием. Например, если заявка пройдет через такую серию блоков

Вариант D

*:ObjName.Extract(NumBl,NumRP);

*:ObjName.ExtractInterrupt(NumBl,NumRP);

*:ObjName.Preempt;

То она получит в свое распоряжение устройство, избавив его от обслуживавшейся заявки, и всех ранее прерванных. Естественно, после обслуживания заявка должна выполнить блок

*:ObjName.Return;

Очевидно, что возможен также

Вариант E

*:ObjName.Extract(NumBl,NumRP);

*:ObjName.ExtractInterrupt(NumBl,NumRP);

*:ObjName.Seizet;

Тогда заявка получит в свое распоряжение устройство, избавив его от обслуживавшейся заявки, и всех ранее прерванных. Естественно, после обслуживания заявка должна выполнить блок

*:ObjName.Release;

Он очень похож на вариант D, но устройство при этом будет занято обычным образом.

Из этих вариантов, только вариант A позволяет заявке продолжить обслуживание после периода недоступности. Все остальные варианты, тем или иным способом удаляют заявки из устройства. Поэтому, рассмотрим пример моделирования доступности – недоступности в этом варианте.

Пусть заявки поступают в устройство со средним интервалом 100, распределенным по экспоненциальному закону.

Пусть обслуживание заявки занимает в среднем 80+-70 единиц времени.

Каждая следующая заявка прерывает обслуживание предыдущей заявки.

Пусть также со средним интервалом 2000+-1500 наступает период недоступности, длящийся в среднем 200 единиц времени, распределенный по экспоненциальному закону.

Промоделировать 100 периодов недоступности. Протабулировать время обслуживания заявок в устройстве.

Пример такой модели мог бы выглядеть следующим образом.

```
 {~vb} Var
{/G} G:Tgenerate;
{/Q} Q:Tqueue;
{/Fa} Fa:Tfacility;
{/G0} G0:Tgenerate;
{/Tb} Tb:Ttable;
 {~ve}
 {~pfe}
```

```
 {~ib} procedure Initial;begin
setStart(100);
{/G}  init(G,'G',G_,Exponential(100));
{/Q}  init(Q,'Q');
{/Fa}  init(Fa,' Fa');
{/G0}  .init(G0,' G0',G0_,2000,1500);
{/Tb}  init(Tb,' Tb',0,100,100);
 {~ie}end;
 {~mtb}procedure ModelTxt;begin case ActiveBlock of
{/G} ::G_   *:G.generate(Exponential(100));
      *: q.queue ;
      *: fa.preempt ;
      *: q.depart ;
      *: mark;
      *: advance(80,70)  ;
      *: tb.tabulate(TranAge) ;
      *: fa.return ;
      *: Terminate ;
{/G0} ::G0_   *:G0.generate(2000,1500);
         *:fa.Preempt0;
         *:fa.SetGoto(-1);
         *:advance(Exponential(200))  ;
         *:fa.Return0;
         *:fa.SetGoto;
       *: Terminate(1) ;
 {~mte} end;end;
 {~mb} procedure Simulation;  begin
  start(GetTerm);
  show(tb,1);
 {~me}end;
 {~rb} procedure Report;begin
 {~re} end;
 {~cab} procedure CloseAllObj;begin
 {~cae} end;
```

Результирующая статистика модели имеет вид.

```
{/G} {::G_}   1:G.generate(Exponential(100));
         2: q.queue ;
         3: fa.preempt ;
         4: q.depart ;
         5: mark;
         6: advance(80,70)  ;
         7: tb.tabulate(TranAge) ;
         8: fa.return ;
         9: Terminate ;
{/G0} {::G0_}   10:G0.generate(2000,1500);
11:fa.Preempt0;
12:fa.SetGoto(-1);
13:advance(Exponential(200))  ;
14:fa.Return0;
15:fa.SetGoto;
           16: Terminate(1) ;
```

StartTime 0.00000 EndTime 200210.59821

BLOCK Report

Location	Entries	Current
1	1985	0
2	1985	3
3	1982	0
4	1982	0
5	1982	0
6	1982	18
7	1964	0
8	1964	0
9	1964	0
10	100	0
11	100	0
12	100	0
13	100	0
14	100	0
15	100	0
16	100	0

Report CURRENT LIST

TRAN	Assem	Pr	Current	Next	TimeStamp	TimeEnter	Interrupt
2083	2083	0	2	3	200072.78334	200072.78334	0
2085	2085	0	2	3	200125.36502	200125.36502	0
2086	2086	0	2	3	200160.17109	200160.17109	0

Report FUTURE LIST

TRAN	Assem	Pr	Current	Next	TimeStamp	EndTime	Interrupt
2087	2087	0	0	1	200264.71605	200264.71605	0
2082	2082	0	6	7	199871.69771	200303.45018	0
2084	2084	0	0	10	202764.23150	202764.23150	0
2073	2073	0	6	7	199275.86943	-117.63638	1
2068	2068	0	6	7	199007.65718	-115.29520	1
1850	1850	0	6	7	180288.85770	-101.22134	1
2067	2067	0	6	7	198985.69154	-95.79407	1
2070	2070	0	6	7	199274.67483	-88.07089	1
2077	2077	0	6	7	199516.38191	-63.04802	1
2078	2078	0	6	7	199586.88100	-55.97808	1
1852	1852	0	6	7	180288.85770	-53.17635	1
2069	2069	0	6	7	199009.52601	-40.49107	1
2079	2079	0	6	7	199647.26049	-34.51908	1
2076	2076	0	6	7	199454.93202	-33.89592	1
2072	2072	0	6	7	199274.67483	-33.16111	1
1847	1847	0	6	7	178622.36687	-19.85073	1
2075	2075	0	6	7	199400.62144	-17.19751	1
2074	2074	0	6	7	199296.36428	-13.20421	1
1848	1848	0	6	7	178626.25056	-6.49619	1
1846	1846	0	6	7	178521.79534	-5.99402	1

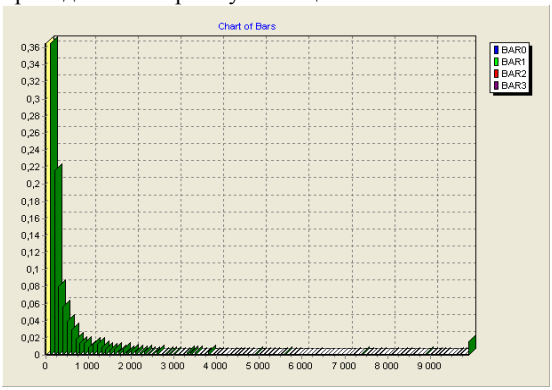
Report Interrupt LIST

TRAN	NumFacil	TRANP
1846	5	1846
1847	5	1846
1848	5	1847
1850	5	1848
1852	5	1850
2067	5	1852
2068	5	2067
2069	5	2068
2070	5	2069
2072	5	2070
2073	5	2072
2074	5	2073
2075	5	2074
2076	5	2075
2077	5	2076
2078	5	2077
2079	5	2078

Queue	Entries	Current	Max	Zero	AverQueue	AverTime	AverTime(-Ze)	NumObj
Q	1985	3	21	1761	0.26423	26.65097	236.17043	4

```
  Facility    Entries    Current         Utility        AverTime     NextGo       TRAN      TRANP
NumObj
        Fa       1982          1         0.79684        80.49276          0       2082    2079
5
```

В этой статистике не приведены результаты табулирования, вместо них мы приводим гистограмму таблицы.



30 Блоки работы с группами.

Группы позволяют иметь множества целых чисел, вещественных чисел, строк, а также множества заявок. Используя группы заявок, можно получать доступ к ее членам вне зависимости от того, в каких списках находится та или иная заявка. Для групп, можно выполнять проверку на принадлежность заявок или чисел к группе. В группу можно добавлять числа или заявки. Из нее можно удалять числа или заявки. Можно также менять значения P – параметров заявок группы или их приоритет, а также обнулять время жизни заявок.

К блокам, обеспечивающим работу с группами вещественных чисел, относятся следующие блоки. Здесь ObjName- это имя группы.

*:ObjName.Join (ValueList); этот блок определяет вещественные числа, добавляемые к группе.

*:ObjName.Remove (Value,NumBl); этот блок определяет вещественное число, удаляемое из группы. Если нужного числа в группе нет, то заявка уходит по метке NumBl. Если метка опущена или равна 0, то заявка всегда переходит в следующий блок.

*:ObjName.RemoveAll ; этот блок удаляет все числа.

*:ObjName.Examine(Value,NumBl); этот блок проверяет наличие в группе заданного числа. Если его в группе нет, то заявка уходит по указанной метке.

*:ObjName.Copy(Values); Копирует данные из группы в динамический массив.

Следующие функции позволяют получить статистические данные по группе.

ObjName.A Определяет среднюю величину числовой группы.

ObjName.C Определяет число вхождений в группу.

ObjName.L Определяет текущую длину группы.

ObjName.M Определяет текущую максимальную длину группы.

Следующие процедуры управляют использованием группы.

ObjName.JoinProc (ValueList); определяет вещественные числа, добавляемые к группе.

ObjName.CopyProc(Values); Копирует данные из группы в динамический массив.

ObjName.RemoveAllProc; - удаляет все числа.

ObjName.ReportFile(FileName); Выводит отчет по группе. FileName – имя файла.

ObjName.GroupReportFile(FileName); Создает полный отчет по членам группы. FileName – имя файла.

Для работы с группами целых чисел, используются точно такие же блоки, процедуры и функции. Однако добавляемые, удаляемые, и проверяемые числа должны быть целыми, а тип группы TIGroup.

Для работы с группами строк, используются точно такие же блоки, процедуры и функции. Однако добавляемые, удаляемые, и проверяемые элементами должны быть строки, а тип группы TSGroup.

Для работы с группами заявок, используются большая номенклатура блоков, процедур и функций.

Блоки.

*:ObjName.Join (ATran) ; этот блок добавляет заданную заявку в группу, по умолчанию, добавляется активная заявка.

*:ObjName.Remove (Num,Del,NumIP); этот блок удаляет до Num заявок из группы. Если 0, то потенциально могут быть удалены все заявки. Del – отдельно описанная логическая функция без параметров, которая определяет заявки, подлежащие удалению. В IP(NumIP) запишется число реально удаленных заявок. Могут быть опущены, в том числе и все параметры. Тогда подразумевается удаление одной любой заявки. Причем тогда количество реально удаленных заявок никуда не записывается.

*:ObjName.Examine(NumBl,ATran); Блок проверяет наличие заданной заявки в группе. По умолчанию, проверяется наличие активной заявки. Если нужной заявки в группе нет, то активная заявка переходит на указанный блок.

*:ObjName.Scan (YesF,ValueF,NumRP,NumBl); Блок отыскивает заявки, удовлетворяющие отдельно описанной логической функции Yesf. Для первой попавшейся такой заявки, вычисляется значение вещественной функции ValueF. Оно записывается в RP(NumRP) - параметр активной заявки. Если нужной заявки не найдется, то заявка уходит по метке NumBl, если этот параметр есть.

*:ObjName.Scan (YesF,ValueF,NumIP,NumBl); Блок аналогичен предыдущему, но вычисляет значение целой функции. Запись происходит в IP - параметр.

*:ObjName.Scan (YesF,ValueF,NumSP,NumBl); Блок аналогичен предыдущему, но вычисляет значение строковой функции. Запись происходит в SP - параметр.

*:ObjName.Scan (YesF,ValueF,NumBP,NumBl); Блок аналогичен предыдущему, но вычисляет значение логической функции. Запись происходит в BP - параметр.

*:ObjName.Alter (YesF,ValueF,NumRP,Num,NumIPCount); Блок - вычисляет логическую функцию Yesf для определения подходящих заявок из группы. Для первых попавшихся подходящих заявок вычисляется значение вещественной функции ValueF. Оно записывается в RP(NumRP) для подходящей заявки из группы. При NumRP=-1 – выполняется изменение времени рождения на текущий момент времени. Всего модифицируется не более Num заявок из группы, или все. Реальное число модифицированных заявок записывается в IP(NumIPCount) активной заявки. Два

последних параметра могут быть опущены. Тогда модифицируются все заявки (значение Num=0).

*:ObjName.Alter (YesF,ValueF,NumIP,Num,NumIPCount); Блок аналогичен предыдущему, но вычисляется значение целой функции. Оно записывается в IP(NumIP) для подходящей заявки из группы. При NumIP=-1 Value рассматривается как новый приоритет заявки из группы.

*:ObjName.Alter (YesF,ValueF,NumSP,Num,NumIPCount); Блок аналогичен предыдущему, но вычисляется значение строковой функции. Оно записывается в SP(NumSP) для подходящей заявки из группы.

*:ObjName.Alter (YesF,ValueF,NumBP,Num,NumIPCount); Блок аналогичен предыдущему, но вычисляется значение логической функции. Оно записывается в BP(NumBP) для подходящей заявки из группы.

*:ObjName.Select(Log ,ANums); Блок записывает в динамический массив номера заявок из группы, для которых условие - функция Log - истинна.

Функции.

ObjName.C Определяет число вхождений в группу.

ObjName.L Определяет текущую длину группы.

ObjName.A Определяет среднее число заявок в группе.

ObjName.V Определяет временной интеграл.

ObjName.T Определяет среднее время, проведенное заявкой в группе.

ObjName.M Определяет максимальное число заявок в группе.

ObjName.I Выдает минимальное число заявок в группе.

Эти функции обычно используются внутри отдельно описанных функций YesF, Value, Ssum и Log.

ObjName.Assem Определяет семейство заявки из группы.

ObjName.TranAge Определяет время жизни заявки из группы.

ObjName.RP(Num) Определяет значение RP(Num) для заявки из группы.

ObjName.Prior Определяет приоритет для заявки из группы.

ObjName.IP(Num) Определяет значение IP(Num) для заявки из группы.

ObjName.BP(Num)Определяет значение BP(Num) для заявки из группы.

ObjName.SP(Num) Определяет значение SP(Num) для заявки из группы.

ObjName.Tran Определяет номер заявки для заявки из группы.

ObjName.ScanNum Выдает порядковый номер заявки в списке в процессе просмотра с помощью процедур группы Scan и Alter. В начале просмотра значение функции равно 0.

Эти функции хотя и возможно указывать в YesF и Value, но рекомендуется вычислять заранее и записывать в RP – параметр активной заявки.

ObjName.Sum(Ssum,Log) Определяет значение суммы значений функции Ssum для заявок из группы. Учитываются только заявки, удовлетворяющие условию - функции Log. Последний параметр может быть опущен. Тогда учитываются все заявки.

ObjName.Avg(Ssum,Log) Аналогична предыдущей функции, но вычисляет среднее значение.

ObjName.Min(Ssum,Log) Аналогична предыдущей функции, но вычисляет минимальное значение.

ObjName.Max(Ssum,Log) Аналогична предыдущей функции, но вычисляет максимальное значение.

Процедуры управления моделью.

Delete(Num, del) - удаляет заданное количество заявок, удовлетворяющих условию - функции Del - результат ее это количество реально удаленных заявок из группы.

ObjName.SelectProc(Log ,ATran); записывает в динамический массив номера заявок из группы, для которых условие - функция Log - истинна.

Delete; удаляет все заявки из группы.

ObjName.ReportFile(FileName); Выводит отчет по группе. FileName – имя файла.

TTGroup.(ObjName); Создает группу заявок.

ObjName.GroupReportFile(FileName); Создает полный отчет по заявкам группы. FileName – имя файла.

Пример задачи, использующей указанные блоки.

Пусть в каждом из трех магазинов, имеется набор товаров, с кодами от 10 до 121. Первоначальная номенклатура товаров каждого из магазинов создается путем наложения 150 случайно сгенерированных кодов товаров.

Далее, с интервалами 3600+-1800 в каждый магазин добавляется товар со случайным кодом, если его там не было. И с таким же интервалом, заканчивается товар со случайным кодом, если он там был.

Покупатели приходят в магазины за покупками со средним интервалом 3, распределенным по экспоненциальному закону. Код покупки генерируется равновероятно. Нас интересуют только те покупатели, для которых справедлив следующий порядок обхода магазинов: вначале проверяется, есть ли нужный товар в первом магазине, и если есть, то совершается покупка. Иначе посещается второй магазин, и так далее. Хотя, вероятно, есть и другие покупатели. Поиск товара в каждом из магазинов занимает 300+-200 единиц времени. Промоделировать работу магазинов в течение 100000 единиц времени. Определить число покупок в каждом из магазинов.

Текст такой модели может выглядеть следующим образом:

```
 {~vb} Var
{/ng1} ng1:Tgenerate;
{/ng2} ng2:Tgenerate;
{/ng3} ng3:Tgenerate;
{/ng4} ng4:Tgenerate;
{/ng5} ng5:Tgenerate;
{/ng6} ng6:Tgenerate;
{/ng7} ng7:Tgenerate;
{/ng8} ng8:Tgenerate;
{/ng9} ng9:Tgenerate;
 mag2:Integer;
 mag3:Integer;
 no_bye:Integer;
 mag1:Integer;
{/gr1} gr1:TIgroup;
{/gr2} gr2:TIgroup;
{/gr3} gr3:TIgroup;
```

```
  {~ve}
  {~pfe}
  {~ib} procedure Initial;begin
    setstart (1 );
{/ng1}      init(ng1,' ng1',ng1_, 0,0,0,150);
{/ng2}      init(ng2,' ng2',ng2_,3600,1800);
{/ng3}      init(ng3,' ng3',ng3_,3600,1800);
{/ng4}      init(ng4,' ng4',ng4_,3600,1800);
{/ng5}      init(ng5,' ng5',ng5_,3600,1800);
{/ng6}      init(ng6,' ng6',ng6_,3600,1800);
{/ng7}      init(ng7,' ng7',ng7_,3600,1800);
{/ng8}      init(ng8,' ng8',ng8_,exponential(3));
{/ng9}      init(ng9,' ng9',ng9_,100000);
  mag2:=0;
  mag3:=0;
  no_bye:=0;
  mag1:=0;
{/gr1}  init(gr1,' gr1');
{/gr2}  init(gr2,' gr2');
{/gr3}  init(gr3,' gr3');
  {~ie}end;
  {~mtb}procedure ModelTxt;begin case ActiveBlock of
{/ng1} ::ng1_   *:ng1.generate( 0,0,0,150);
*:gr1.join([duniform(10,121)]);
*:gr2.join([duniform(10,121)]);
*:gr3.join([duniform(10,121)]);
*:terminate;
{/ng2} ::ng2_   *:ng2.generate(3600,1800);
*:gr1.join ([duniform(10,121)]);
*:terminate;
{/ng3} ::ng3_   *:ng3.generate(3600,1800);
*:gr2.join([duniform(10,121)]);
*:terminate;
{/ng4} ::ng4_   *:ng4.generate(3600,1800);
*:gr3.join([duniform(10,121)]);
*:terminate;
{/ng5} ::ng5_   *:ng5.generate(3600,1800);
*:gr1.remove(duniform(10,121));
*:terminate;
{/ng6} ::ng6_   *:ng6.generate(3600,1800);
*:gr2.remove(duniform(10,121));
*:terminate;
{/ng7} ::ng7_   *:ng7.generate(3600,1800);
*:gr3.remove(duniform(10,121));
*:terminate;
{/ng8} ::ng8_   *:ng8.generate(exponential(3));
```

```
*:IPlet(1,[duniform(10,121)]);
*:advance( 300,200);
*:gr1.examine(Ip(1),next);
*:Let(mag1,mag1+1);
*:terminate;
::next  *:advance( 300,200);
*:gr2.examine(Ip(1) ,next1);
*:Let(mag2,mag2+1);
*:terminate;
::next1 *:advance( 300,200);
*:gr3.examine(Ip(1),next2);
*:Let(mag3,mag3+1);
*:terminate;
::next2 *:Let(no_bye,no_bye+1);
*:terminate;

{/ng9} ::ng9_   *:ng9.generate(100000);
*:terminate(1);
  {~mte} end;end;
  {~mb} procedure Simulation;  begin
 {/sh}     start(GetTerm);
  {~me}end;
  {~rb} procedure Report;begin
  {~re} end;
  {~cab} procedure CloseAllObj;begin
  {~cae} end;
```

Выходная статистика модели имеет вид:

```
{/ng1} {::ng1_}   1:ng1.generate( 0,0,0,150);
2:gr1.join([duniform(10,121)]);
3:gr2.join([duniform(10,121)]);
4:gr3.join([duniform(10,121)]);
5:terminate;
{/ng2} {::ng2_}   6:ng2.generate(3600,1800);
7:gr1.join ([duniform(10,121)]);
8:terminate;
{/ng3} {::ng3_}   9:ng3.generate(3600,1800);
10:gr2.join([duniform(10,121)]);
11:terminate;
{/ng4} {::ng4_}   12:ng4.generate(3600,1800);
13:gr3.join([duniform(10,121)]);
14:terminate;
{/ng5} {::ng5_}   15:ng5.generate(3600,1800);
16:gr1.remove(duniform(10,121));
17:terminate;
{/ng6} {::ng6_}   18:ng6.generate(3600,1800);
19:gr2.remove(duniform(10,121));
20:terminate;
{/ng7} {::ng7_}   21:ng7.generate(3600,1800);
22:gr3.remove(duniform(10,121));
23:terminate;
{/ng8} {::ng8_}   24:ng8.generate(exponential(3));
25:IPlet(1,[duniform(10,121)]);
26:advance( 300,200);
27:gr1.examine(Ip(1),next);
28:Let(mag1,mag1+1);
```

```
29:terminate;
{::next}  30:advance( 300,200);
31:gr2.examine(Ip(1) ,next1);
32:Let(mag2,mag2+1);
33:terminate;
{::next1} 34:advance( 300,200);
35:gr3.examine(Ip(1),next2);
36:Let(mag3,mag3+1);
37:terminate;
{::next2} 38:Let(no_bye,no_bye+1);
39:terminate;
{/ng9} {::ng9_}   40:ng9.generate(100000);
41:terminate(1);

StartTime         0.00000 EndTime     100000.00000
          BLOCK Report
    Location   Entries   Current
          1       150         0
          2       150         0
          3       150         0
          4       150         0
          5       150         0
          6        29         0
          7        29         0
          8        29         0
          9        30         0
         10        30         0
         11        30         0
         12        29         0
         13        29         0
         14        29         0
         15        28         0
         16        28         0
         17        28         0
         18        27         0
         19        27         0
         20        27         0
         21        27         0
         22        27         0
         23        27         0
         24     32987         0
         25     32987         0
         26     32987        93
         27     32894         0
         28     22619         0
         29     22619         0
         30     10275        36
         31     10239         0
         32      7409         0
         33      7409         0
         34      2830        11
         35      2819         0
         36      2326         0
         37      2326         0
         38       493         0
         39       493         0
         40         1         0
         41         1         0
           Report FUTURE LIST
       TRAN     Assem        Pr   Current      Next       TimeStamp         EndTime Interrupt
      33187     33187         0        26        27     99571.55574    100000.98875         0
      33315     33315         0         0        24    100001.36866    100001.36866         0
      33004     33004         0        30        31     99066.82429    100001.95708         0
      33259     33259         0        26        27     99809.50841    100004.07499         0
      33213     33213         0        26        27     99666.09070    100004.63813         0
      33258     33258         0        26        27     99806.09910    100006.05616         0
      33284     33284         0        26        27     99889.14194    100012.23940         0
      33194     33194         0        30        31     99593.12728    100013.95465         0
      33141     33141         0        30        31     99445.70482    100022.67705         0
      33251     33251         0        26        27     99788.36337    100028.47410         0
```

```
     33162     33162         0        30        31     99511.86418    100029.24017         0

          Report  IP LIST (  Integer  )
       TRAN       Num   Integer
     32982         1       103
     32994         1       103
     33004         1        50
     33011         1        66
     33025         1        66
     33026         1        95
     33037         1        66
     33060         1        66

mag2  =        7409
mag3  =        2326
no_bye  =         493
mag1  =       22619
  GroupInt   Entries   Current       Max       AverGroup    NumObj
       gr1       179        73        85        77.35774        12
gr1           Report IGROUP LIST ( INTEGER ) num=73
              10
              11
              12

  GroupInt   Entries   Current       Max       AverGroup    NumObj
       gr2       180        67        79        71.98670        13
gr2           Report IGROUP LIST ( INTEGER ) num=67
              10
              12

  GroupInt   Entries   Current       Max       AverGroup    NumObj
       gr3       179        75        88        81.13900        14
gr3           Report IGROUP LIST ( INTEGER ) num=75
              10
              11
              12
              13
              15
              16
              17
```

Часть списков приводится не полностью.

Сугубо экспериментальная модель, просто подтверждающая работоспособность блоков, работающих с группами заявок.

Пусть заявки поступают на вход системы с интервалом 100+-90 и задерживаются на 300+-200 единиц времени. Заявки получают случайное значение RP(4), равномерно распределенное в интервале от 1 до 10. И случайный приоритет, равномерно распределенный в интервале от 3 до 7.

С вероятностью 0.4 заявки попадают в группу заявок, где они пребывают 1500+-200 единиц времени, а затем покидают систему. С вероятностью 0.6 заявки задерживаются на 400+-200 единиц времени, а затем покидают систему.

Заявки, манипулирующие с группой, появляются с интервалом 2000+-900 единиц времени, а затем ищут в группе заявки, у которых параметр RP(4)> 7 и в параметр RP(2) заносят значение RP(4)*PR/2, взятое из параметров заявок группы.

При успехе поиска заявка задерживается на 10+-5 единиц времени, а при неуспехе – сразу удаляется из системы. Выполнить моделирования до ухода 100 заявок манипулирующих с группой.

Текст этой модели имеет вид.

{~vb} Var

```
{/g1} g1:Tgenerate;
{/g2} g2:Tgenerate;
{/grou} grou:TTgroup;
  {~ve}
{/logf}   function logf : boolean;   begin result:=grou.rp(4)> 7 ;       end;
{/rezf}   function rezf : double;   begin result:=grou.rp(4)*grou.pr/2  ;   end;
  {~pfe}
  {~ib} procedure Initial;begin
setstart(100) ;
{/g1}      init(g1,' g1',g1_,100,90);
{/g2}      init(g2,' g2',g2_,2000,900);
{/grou}  init(grou,' grou');
  {~ie}end;
  {~mtb}procedure ModelTxt;begin case ActiveBlock of
{/g1} ::g1_   *:g1.generate(100,90);
    *:advance(300,200);
 *:rPlet(4,[duniform(1,10)]);
 *:priority(duniform(3,7));
  *:test([random<0.4,true],[!*,lb3]);
   *:grou.Join;
   *:advance(1500,200);
   *:terminate;
::lb3 *:advance(400,200);
*:terminate;
{/g2} ::g2_   *:g2.generate(2000,900);
*:grou.scan(logf,rezf,2,md1);
 *:split(1,!*+1);
*:advance(10,5);
::md1        *:tostring(0,'TG1= '+INTTOSTR(Term));
*:terminate(1);
  {~mte} end;end;
  {~mb} procedure Simulation;  begin
 {/sh}     start(GetTerm);
  {~me}end;
  {~rb} procedure Report;begin
  {~re} end;
  {~cab} procedure CloseAllObj;begin
  {~cae} end;
```

Итоговая статистика подтверждает, что модель работает в соответствии с описанием.

```
{/g1} {::g1_}    1:g1.generate(100,90);
     2:advance(300,200);
 3:rPlet(4,[duniform(1,10)]);
 4:priority(duniform(3,7));
  5:test([random<0.4,true],[6,lb3]);
   6:grou.Join;
   7:advance(1500,200);
   8:terminate;
{::lb3} 9:advance(400,200);
```

```
10:terminate;
{/g2} {::g2_}   11:g2.generate(2000,900);
12:grou.scan(logf,rezf,2,md1);
 13:split(1,14+1);
14:advance(10,5);
{::md1}         15:tostring(0,'TG1= '+INTTOSTR(Term));
16:terminate(1);

StartTime         0.00000 EndTime     109326.72162
          BLOCK Report
    Location   Entries   Current
           1      1125         0
           2      1125         4
           3      1121         0
           4      1121         0
           5      1121         0
           6       455         0
           7       455         6
           8       449         0
           9       666         3
          10       663         0
          11        54         0
          12        54         0
          13        47         0
          14        47         1
          15       100         0
          16       100         0
            Report FUTURE LIST
        TRAN     Assem        Pr   Current      Next       TimeStamp         EndTime Interrupt
       1224      1224         0         2         3    109088.20425    109332.23273         0
       1205      1205         0        14        15    109326.72162    109334.01640         0
       1221      1221         0         2         3    108957.87903    109352.07176         0
       1226      1226         0         0         1    109369.14947    109369.14947         0
       1222      1222         0         2         3    109006.51836    109447.33965         0
       1219      1219         3         9        10    108890.47757    109492.42912         0
       1207      1207         5         7         8    107365.36630    109515.78266         0
       1220      1220         4         9        10    108911.05960    109531.82653         0
       1209      1209         6         7         8    107579.98193    109567.15147         0
       1223      1223         6         9        10    109055.99802    109634.68713         0
       1225      1225         0         2         3    109243.85093    109643.95470         0
       1211      1211         3         7         8    107879.16087    109761.15196         0
       1212      1212         5         7         8    107999.35386    109783.08226         0
       1215      1215         7         7         8    108360.13775    109912.75657         0
       1218      1218         4         7         8    108766.16073    110555.54790         0
       1227      1227         0         0        11    111571.53990    111571.53990         0
            RP LIST ( Double )
        TRAN       Num          Double
       1205         2        22.50000
       1207         4         9.00000
       1209         4        10.00000
       1211         4         3.00000
       1212         4         2.00000
       1215         4         2.00000
       1218         4         9.00000
       1219         4         5.00000
       1220         4         1.00000
       1223         4         8.00000
 GroupTran   Entries   Current    NumObj
      grou       455         6         5
grou          Report TGROUP LIST ( TRANSACTIONS ) num=6
           1207
           1209
           1211
           1212
           1215
           1218
```

31 Редко используемые блоки.

*: MoveNext; блок без операции, иногда этот блок бывает полезным, чтобы точнее разобраться в статистике по результатам моделироввания.

*: SetTerm(Num); этот блок устанавливает новое значение TG1 - счетчика.

*: Displace(ATran,NumBl,NumRP,NumSP); этот блок отправляет заявку с номером ATran- в блок с меткой Numbl. NumRP - определяет номер RP - параметра, где сохраняется оставшееся время задержки. NumSP- номер SP -параметра, где записывается обозначение списка, где заявка обнаружена.

Здесь "N" - это отсутствие заявки. "P" – заявка обнаружена в списке прерываний.

"C"- заявка обнаружена в списке текущих событий. "F" – заявка обнаружена в списке будущих событий. "U" - заявка обнаружена в списке пользователя. "S" – заявка обнаружена в списке синхронизации. По умолчанию, соответствующая информация не записывается в P – параметры.

*:FindTran(ATran,NumSP,NxtIP,CurIP); этот блок разыскивает заявку с данным номером. Параметр NumSp – аналогичен такому же параметру блока Displace. Номер следующего блока и текущего блока, записываются в параметры IP(NxtIP), IP(CurIP). По умолчанию, эти номера не записываются.

*: RPDelete([NumList],ATran); Блок удаляет параметр RP с номерами из списка для данной заявки. По умолчанию - для активной заявки.

*: IPDelete([NumList],ATran); Блок удаляет параметр IP с номерами из списка для данной заявки. По умолчанию - для активной заявки.

*: SPDelete([NumList],ATran); Блок удаляет параметр SP с номерами из списка для данной заявки. По умолчанию - для активной заявки.

*: BPDelete([NumList],ATran); Блок удаляет параметр BP с номерами из списка для данной заявки. По умолчанию - для активной заявки.

*: RPDelete(ATran); Блок удаляет все RP -параметры данной заявки. По умолчанию - для активной заявки.

*: IPDelete(ATran); Блок удаляет все IP -параметры данной заявки. По умолчанию - для активной заявки.

*: SPDelete(ATran); Блок удаляет все SP -параметры данной заявки. По умолчанию - для активной заявки.

*: BPDelete(ATran); Блок удаляет все BP -параметры данной заявки. По умолчанию - для активной заявки.

*: IncTran; Блок увеличивает на 1 номер последней запланированной или выпущенной заявки. Он предназначен для создания фиктивных наборов P –параметров, для которых нет соответствующей заявки. Таким образом, можно в любой момент создать новый набор P –параметров.

32 Элементарные сведения о языке Pascal

Данный раздел не является руководством по языку Pascal, он скорее содержит некоторые сведения об этом языке, которые с большей вероятностью могут понадобиться разработчикам моделей.

Этот язык состоит из следующих операторов и разделов:

- ✓ Procedure - определяет заголовок процедуры.
- ✓ Function - определяет заголовок функции.

- ✓ Var - определяет переменные в модели в целом, процедуре или функции.
- ✓ Const - определяет константы в модели в целом.
- ✓ Все операторы, кроме заголовков процедур и функций могут иметь метку, отделяющуюся от оператора пробелами.
- ✓ Ключевые слова BEGIN и END- определяют начало и конец составного оператора в процедуре, то есть позволяют создавать составные операторы.
- ✓ Оператор присваивания.
- ✓ Вызов процедур или функций.
- ✓ Оператор If Then или
- ✓ If Then Else -используется для ветвлений в программе.
- ✓ Оператор Case -используется для ветвлений в программе.
- ✓ Оператор While Do- позволяет организовывать циклы с проверкой в начале .
- ✓ Оператор For Do- позволяет организовывать циклы с известным числом повторений.
- ✓ Оператор Repeat Until - позволяет организовывать циклы с проверкой в конце.
- ✓ Оператор GOTO – позволяет выполнить безусловный переход.

Оператор присваивания имеет вид:

A:=B;

Здесь A – глобальная или локальная переменная в модели, или такой же элемент массива. B - выражение.

Например

Home:=(RP(time)+RP(num))*3.6+home;

Примечание. В процедуре нельзя изменить значения P – параметров путем присваивания, но для этого есть процедуры типа RPLet и так далее.

Текст заголовка процедуры имеет вид

Pprocedure имя(список аргументов с их типом через точку с запятой); Var описание временных переменных функции; составной оператор;

Составной оператор представляет собой последовательность операторов, разделенных точкой с запятой, и обрамленных ключевыми словами BEGIN END;

Процедуры и функции могут использоваться рекурсивно, то есть вызывать сами себя, как прямо, так и косвенно.

Заголовок функции имеет вид

Function имя(список аргументов с их типом через точку с запятой):тип функции; Var описание временных переменных функции; Составной оператор;

Как правило, используются переменные и функции следующих типов.

Integer, Double, String, Boolean, что означает целый, вещественный двойной точности строковый и логический тип данных. При описании переменных в заголовке процедуры или функции, каждая переменная описывается следующим образом.

Имя:Тип;

Если перед именем переменной стоит Var, то изменение соответствующей переменной в процедуре или функции изменяет значение переменной, указанной в вызове функции.

При разработке моделей, не рекомендуется использовать процедуры и функции с параметрами, а рекомендуется напрямую использовать параметры модели, которые все равно «видны» в процедуре или функции.

Переменная Result используется для присваивания значения результату функции.

Раздел Var описывает в процедуре временные величины, которые существуют только во время выполнения процедуры. Если Var относится к модели в целом, то есть, описан вне процедур и функций, то его переменные существуют все время работы программы. Раздел имеет вид:

Var - список описаний переменных через точку с запятой.

Описание массивов выглядит несколько иначе, имя: Array[список пар границ индексов через запятую] of Тип элемента массива. Каждая пара границ индексов определяет минимальное и максимальное значение соответствующего индекса. Они разделяются двумя точками, по образцу 2..15, однако, минимальное значение традиционно бывает только 0 или 1.

Примеры

Var Value1:Integer; tmp:Double;

Var MATRIX DataArray: array [5,7,15] of String;

Fac:array [1..20] of Tfacility;

Раздел Const описывает в модели имена, сопоставленные константам. Он состоит из пар

Имя=значение;

Тогда в модели можно использовать описанное имя вместо соответствующей константы. Чаще всего используются константы для номеров X и P – параметров, для повышения наглядности модели.

Оператор If имеет вид:

If логическое выражение Then оператор;

Если выражение равно True (истина), то встроенный оператор выполняется, иначе он не выполняется.

Форма оператора.

If логическое выражение Then оператор1
Else оператор2;

В этом случае выполнение оператора происходит несколько по-другому. Если выражение равно True, то выполняется встроенный оператор1, иначе оператор2.

Оператор Case имеет вид:

```
case Par of
Const1:Oper1;
Const2: Oper2;
.
.
else ElseOper;
end;
```

Здесь Par -это управляющее выражение оператора. Оно должно быть целым или символьным. Const1, Const2 – целые или символьные константы. Oper1, Oper2, ElseOper – простые или составные операторы. Если значение Par совпадает с одной из констант, то выполняется соответствующий оператор, а затем управление передается на оператор,

расположенный после Case. Если значение Par не совпадает ни с одной из констант, то выполняется оператор ElseOper, а затем управление передается на оператор, расположенный после Case.

Часть else ElseOper – может отсутствовать. Количество пар Const:Oper – не ограничено.

Оператор GoTo имеет вид:

GoTo A;

Здесь A – это метка в процедуре.

Следующим будет выполняться оператор с этой меткой. Этот оператор категорически не рекомендуется.

Оператор While предназначен для организации циклов и имеет вид:

While логическое выражение DO оператор;

Оператор здесь, как правило, является составным. Логическое выражение вычисляется и если оно имеет значение True, то выполняется оператор, вставленный в цикл. Иначе вставленный оператор не выполняется, цикл заканчивается, и выполнение процедуры идет дальше. Как в процедурах, так и в выражениях модели, можно вызывать как встроенные, так и написанные вами процедуры и функции.

Вызов процедуры и функции выглядит одинаково, однако вызовы процедур нельзя помещать в выражения. Они образуют самостоятельный оператор. Вызов имеет следующий вид.

Имя(список параметров через запятую)

Операторы For Do и Repeat Until – не являются необходимыми. Однако их тоже часто используют для построения циклов.

Оператор For Do имеет следующий вид.

For I:=начало to конец Do оператор;

Здесь I – это переменная цикла. Внутренний оператор цикла выполняется при всех целых значениях переменной цикла, от начала – до конца. Если оператор имеет вид

For I:=конец DownTo начало оператор;

То внутренний оператор цикла выполняется при всех целых значениях переменной цикла, от конца – до начала.

Оператор Repeat Until имеет следующий вид.

Repeat

список операторов через точку с запятой

Until - логическое выражение.

Операторы, указанные между Repeat и Until выполняются многократно, пока логическое выражение не окажется истинным. Тогда цикл завершается, и выполнение процедуры идет дальше.

Очень важно учитывать, что операторы могут вкладываться друг в друга.

В качестве примера рассмотрим процедуру вычисления факториалов, а также модель, для вычисления первых 15 значений факториала.

```
  {~vb} Var
{/G} G:Tgenerate;
{\Sys} Sys:Tparam;
  {~ve}
{/factorial}   function factorial(i:integer) : double;
```

```
begin
  Result:=1;
  while i>1 do
  begin
    result:=result*i;
    i:=i-1;
  end;
end;
  {~pfe}
  {~ib} procedure Initial;begin
setstart(15);
{/G}  init(G,'G',G_,1);
{\Sys} Init(Sys,'Sys');
  {~ie}end;
  {~mtb}procedure ModelTxt;begin case ActiveBlock of
{/G} ::G_   *:G.generate(1);
            *: IPlet(1,[iP(1, sys.N)+1] , sys.N) ;
            *: RPlet(iP(1, sys.N),[Factorial(iP(1, sys.N))] , sys.N) ;
            *: Terminate(1)  ;
  {~mte} end;end;
  {~mb} procedure Simulation;  begin
   start(GetTg1);
  {~me}end;
  {~rb} procedure Report;begin
  {~re} end;
  {~cab} procedure CloseAllObj;begin
  {~cae} end;
```

Здесь в ячейки с номерами 1..15 заносятся значения соответствующих факториалов. Важно, что формальные параметры процедуры, и переменные, описанные в Var, локализованы в ней. То есть такая процедура может изменять только значения глобальных переменных или X и P - параметров, а также возвращать результат вычислений через Result.

Выходная статистика этой модели имеет вид:

```
{/G} {::G_}   1:G.generate(1);
              2: SYS.IPlet(1,[iP(1)+1]) ;
              3: SYS.RPlet(iP(1),[Factorial(iP(1))]) ;
              4: Terminate(1)  ;

StartTime         0.00000 EndTime         15.00000
          BLOCK Report
    Location   Entries   Current
         1        15         0
         2        15         0
         3        15         0
         4        15         0
          Report FUTURE LIST
       TRAN     Assem        Pr   Current      Next       TimeStamp         EndTime Interrupt
        16        16         0         0         1        16.00000        16.00000         0
SYS          Report RP LIST ( DOUBLE ) Count= 15
       Num          double
         1         1.00000
         2         2.00000
         3         6.00000
```

```
          4        24.00000
          5       120.00000
          6       720.00000
          7      5040.00000
          8     40320.00000
          9    362880.00000
         10   3628800.00000
         11  39916800.00000
         12 479001600.00000
         13 6227020800.00000
         14 87178291200.00000
         15 1307674368000.00000
SYS          Report IX LIST ( Integer ) Count= 1
       Num         Integer
         1             15
```

Количество и содержание процедур определяется потребностями разработчика модели.

В принципе, потребность в разработке личных процедур и функций возникает только при профессиональном использовании системы.

33 Выборка требуемых объектов в Object GPSS.

В Object GPSS не предусмотрены блоки выборки требуемых объектов, однако их несложно создать самому. В принципе, в этой среде гораздо удобнее создавать не блоки выборки, а функции. Как это может выглядеть на практике, демонстрирует следующая задача.

Пусть в систему поступают заявки с интервалом 100+-80 минут. Для обслуживания заявок следует выбрать первое попавшееся устройство из 10 имеющихся, у которого активная очередь - минимальна. Примечание: занятие пустого устройства - предпочтительнее. Обслуживание в любом устройстве
занимает в среднем 800 минут и распределено по экспоненциальному закону.

Промоделировать обслуживание 1000 заявок.

Текст модели, решающей эту задачу с помощью пользовательской функции имеет вид.

```
Const
num=10;
  {~vb} Var
{/gen0} gen0: Tgenerate;
{/fac0} fac0:array[1..num] of Tfacility;
{/que0} que0:array[1..num] of Tqueue;
  {~ve}
procedure  facShow;
var i:integer;
begin
  for i:=1 to num do
  ValuePut(i,fac0[i].A);
end;
procedure  QueShow;
var i:integer;
begin
  for i:=1 to num do
```

```
 ValuePut(i,que0[i].a);
end;
{/fun0}   function fmin : integer;
var i,j:integer;
begin
  j:=1;
  for i:=1 to num do
  if (que0[i].L<que0[j].L) or (que0[i].L=que0[j].L)
  and (fac0[i].L=0)and (fac0[j].L=1)   then
  j:=i;
  result:=j;
end;
  {~pfe}
  {~ib} procedure Initial;  var  i:integer; begin
    setstart(1000);
  {/gen0}  init(gen0,' gen0',gen0_,100,80);
{/fac0}  for i:=1 to num do init(fac0[i] , 'fac0'+inttostr(i));
{/que0}  for i:=1 to num do init(que0[i] , ' que0'+inttostr(i));
customProc(['facShow','QueShow'],[facShow,QueShow]);
  {~ie}end;
  {~mtb}procedure ModelTxt;begin case ActiveBlock of
{/gen0} ::gen0_   *:gen0.generate(100,80);
   *: IPlet(1,[fmin]);
   *: que0[IP(1)].Queue;
   *: fac0[IP(1)].Seize;
   *: Advance(exponential(800));
   *: que0[IP(1)].Depart;
   *: fac0[sIP(1)].Release;
   *: Terminate(1);
  {~mte} end;end;
  {~mb} procedure Simulation;  begin
    start(GetTerm);
  {~me}end;
  {~rb} procedure Report;  var  i:integer;  begin
  {~re} end;
  {~cab} procedure CloseAllObj;begin
  {~cae} end;
```

В пользовательской функции fmin используются две переменные i и j. Переменная i – предназначена для организации цикла просмотра длин всех очередей, а j – запоминает номер очереди с минимальной длиной.

Именно ее итоговое значение и является результатом функции. Поэтому весь выбор требуемого устройства сводится к выполнению блока

*: IPlet(1,[fmin]); тогда Ip(1) будет хранить номер нужного устройства, и вся задача становится намного проще.

Текст процедуры имеет следующий вид.

```
{/fun0}   function fmin : integer;
```

```
var i,j:integer;
begin
 j:=1;
 for i:=1 to num do
 if (que0[i].L<que0[j].L) or (que0[i].L=que0[j].L)
 and (fac0[i].L=0)and (fac0[j].L=1)   then
 j:=i;
 result:=j;
end;
```

Итоговая статистика по задаче выглядит следующим образом.

```
{/gen0} {::gen0_}   1:gen0.generate(100,80);
   2: IPlet(1,[fmin]);
   3: que0[IP(1)].Queue;
   4: fac0[IP(1)].Seize;
   5: Advance(exponential(800));
   6: que0[IP(1)].Depart;
   7: fac0[IP(1)].Release;
   8: Terminate(1);

StartTime         0.00000 EndTime     102932.17650
         BLOCK Report
    Location   Entries   Current
         1      1007         0
         2      1007         0
         3      1007         0
         4      1007         0
         5      1007         7
         6      1000         0
         7      1000         0
         8      1000         0
          Report FUTURE LIST
       TRAN     Assem        Pr   Current      Next       TimeStamp         EndTime Interrupt
      1008      1008         0         0         1    102956.77948    102956.77948         0
      1004      1004         0         5         6    102479.39678    103112.76111         0
      1007      1007         0         5         6    102873.51971    103133.35370         0
      1005      1005         0         5         6    102593.41883    103140.49610         0
       999       999         0         5         6    101903.49676    103452.04870         0
      1000      1000         0         5         6    102025.90206    103531.78466         0
       994       994         0         5         6    101516.22624    104212.32554         0
      1001      1001         0         5         6    102152.36291    106224.63821         0
          Report  IP LIST (  Integer  )
       TRAN       Num   Integer
       994          1         8
       999          1         7
      1000          1         9
      1001          1         1
      1004          1         2
      1005          1         3
      1007          1         5
  Facility   Entries   Current         Utility        AverTime    NextGo       TRAN      TRANP
NumObj
     fac01       117         1         0.95482       840.01785         0      1001      0
4
  Facility   Entries   Current         Utility        AverTime    NextGo       TRAN      TRANP
NumObj
     fac02       117         1         0.94476       831.16601         0      1004      0
5
  Facility   Entries   Current         Utility        AverTime    NextGo       TRAN      TRANP
NumObj
     fac03       121         1         0.91769       780.66065         0      1005      0
6
  Facility   Entries   Current         Utility        AverTime    NextGo       TRAN      TRANP
NumObj
     fac04       124         0         0.88282       732.82767         0         0      0
7
```

```
  Facility   Entries   Current         Utility        AverTime    NextGo       TRAN      TRANP
NumObj
     fac05       103         1         0.86363       863.06081         0      1007      0
8
  Facility   Entries   Current         Utility        AverTime    NextGo       TRAN      TRANP
NumObj
     fac06       103         0         0.81779       817.25335         0         0      0
9
  Facility   Entries   Current         Utility        AverTime    NextGo       TRAN      TRANP
NumObj
     fac07        96         1         0.72759       780.12680         0       999      0
10
  Facility   Entries   Current         Utility        AverTime    NextGo       TRAN      TRANP
NumObj
     fac08        96         1         0.70810       759.23572         0       994      0
11
  Facility   Entries   Current         Utility        AverTime    NextGo       TRAN      TRANP
NumObj
     fac09        70         1         0.62342       916.71040         0      1000      0
12
  Facility   Entries   Current         Utility        AverTime    NextGo       TRAN      TRANP
NumObj
    fac010        60         0         0.55877       958.58762         0         0      0
13
     Queue   Entries   Current       Max      Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
     que01       117         1         3         0         1.34517      1183.42875
1183.42875        14
     Queue   Entries   Current       Max      Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
     que02       117         1         2         0         1.21866      1072.13451
1072.13451        15
     Queue   Entries   Current       Max      Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
     que03       121         1         2         0         1.05872       900.63161
900.63161        16
     Queue   Entries   Current       Max      Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
     que04       124         0         2         0         0.98288       815.88999
815.88999        17
     Queue   Entries   Current       Max      Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
     que05       103         1         2         0         0.94119       940.57125
940.57125        18
     Queue   Entries   Current       Max      Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
     que06       103         0         2         0         0.85958       859.01426
859.01426        19
     Queue   Entries   Current       Max      Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
     que07        96         1         2         0         0.74199       795.57259
795.57259        20
     Queue   Entries   Current       Max      Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
     que08        96         1         2         0         0.71866       770.54998
770.54998        21
     Queue   Entries   Current       Max      Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
     que09        70         1         2         0         0.63015       926.60783
926.60783        22
     Queue   Entries   Current       Max      Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
    que010        60         0         2         0         0.56092       962.27408
962.27408        23
```

Пусть необходимо решить такую же задачу, но длина допустимой очереди ограничена 2. И, если подходящего устройства не найдется, то заявка должна удаляться из системы.

Тогда текст модели остается почти тем же.

```
Const
num=10;
  {~vb} Var
{/gen0} gen0: Tgenerate;
{/fac0} fac0:array[1..num] of Tfacility;
{/que0} que0:array[1..num] of Tqueue;
  {~ve}
procedure  facShow;
var i:integer;
begin
   for i:=1 to num do  ValuePut(i,fac0[i].A);
end;
procedure  QueShow;
var i:integer;
begin
 for i:=1 to num do  ValuePut(i,que0[i].a);
end;
{/fun0}   function fmin : integer;
var i,j:integer;  begin
j:=1;
for i:=1 to num do if (que0[i].L<que0[j].L) then j:=i;
  if que0[j].L>1 then j:=0;
 result:=j;
  end;
  {~pfe}
  {~ib} procedure Initial;  var  i:integer; begin
    setstart(1000);
  {/gen0}  init(gen0,' gen0',gen0_,100,80);
{/fac0}  for i:=1 to num do init(fac0[i] , 'fac0'+inttostr(i));
{/que0}  for i:=1 to num do init(que0[i] , ' que0'+inttostr(i));
customProc(['facShow','QueShow'],[facShow,QueShow]);
  {~ie}end;
  {~mtb}procedure ModelTxt;begin case ActiveBlock of
{/gen0} ::gen0_   *:gen0.generate(100,80);
   *: IPlet(1,[fmin]);
      *: test([Ip(1)<>0,true ],[!*,out]) ;
   *: que0[IP(1)].Queue;
   *: fac0[IP(1)].Seize;
   *: Advance(exponential(800));
   *: que0[IP(1)].Depart;
   *: fac0[IP(1)].Release;
   *: Terminate(1);
::out  *: Terminate ;
  {~mte} end;end;
  {~mb} procedure Simulation;  begin
```

```
  start(GetTerm);
{~me}end;
{~rb} procedure Report;  begin
{~re} end;
{~cab} procedure CloseAllObj;begin
{~cae} end;
```

В нем только изменена пользовательская функция выбора. В ней после нахождения очереди с минимальной длиной выполняется проверка. Не является ли она более 1. И, если это так, то результат функции окажется равным 0.

Поэтому в новой модели добавлен блок

```
*: test([Ip(1)<>0,true ],[!*,out]) ;
```

Он пропускает заявки с ненулевым значением Ip(1), и отправляет на метку Out все другие заявки.

Даже на этих примерах, становится ясно, что модели с пользовательскими функциями выбора более логичны и понятны, чем модели, решающие эту задачу «в лоб». Подобным же образом можно выбирать любые другие требуемые объекты.

34 Работа с файлами

В принципе, для работы с текстовыми файлами следует использовать процедуры самого языка Pascal. Для этого следует описать текстовый файл как переменную в самой модели. Эта переменная должна иметь тип TextFile.

Далее следует связать эту переменную с файлом на диске и открыть файл для чтения или записи данных.

В первом случае нужно вызвать процедуры

AssignFile(FileVar,FileName);

Reset(FileVar);

А во втором случае нужно вызвать процедуры

AssignFile(FileVar,FileName);

ReWrite(FileVar);

Здесь FileVar – это файловая переменная, а FileName – это строка, содержащая имя файла на диске.

Далее для чтения данных из файла следует использовать процедуры

Read(FileVar, список переменных через запятую);

Readln(FileVar, список переменных через запятую);

Процедура Read читает данные, не обращая внимания на концы строк, а процедура Readln обязательно дочитывает начатую строку до конца, даже если в ней есть данные, которым не сопоставлены переменные.

Тип переменных может быть вещественным, целым, или строковым.

Переменные в указанном порядке будут читаться из файла, если в файле еще есть непрочитанные данные. Вещественные и целые данные должны разделяться пробелами или концами строки, а строковые - должны занимать строку целиком. В процедуре Read не следует указывать строковые переменные.

А в процедуре Readln можно указывать данные любого из трех указанных типов.

Обращаю ваше внимание, что чтение данных возможно только в переменные соответствующих типов, и невозможно чтение в P или X – параметры.

Для проверки закончился ли файл в процессе чтения, можно использовать функцию EOF(FileVar) – которая имеет значение True (Истина), если данные в файле уже закончились.

Для записи данных в файл следует использовать процедуры

Write(FileVar, список выражений с указание ширины полей вывода через запятую);

Writeln(FileVar, список выражений с указание ширины полей вывода через запятую);

Процедура Write записывает данные в одну строку подряд, а процедура Writeln завершает запись концом строки.

Тип выражений может быть вещественным, целым, или строковым.

Выражения будут вычислены и их значения в указанном порядке будут записаны в файл. После каждого выражения может стоять двоеточие и минимальная ширина поля вывода для значения выражения. При выводе, данные могут «слипаться», если ширина поля вывода будет недостаточной для вывода значения. Поэтому рекомендуется включать в список выражений строки из одного или нескольких пробелов в апострофах, или выбирать ширину поля вывода достаточно большой.

После того, как работа с файлом завершена, файл необходимо закрыть с помощью процедуры CloseFile(FileVar).

Если же вы хотите выполнять те же действия в ходе моделирования, то вы можете воспользоваться следующими блоками.

*: OpenGetBlock(Filevar,FileName); Блок открывает файл Filevar для чтения. Имя файла - FileName.

*: OpenPutBlock(Filevar,FileName); Блок открывает файл Filevar для записи. Имя файла - FileName.

*: CloseBlock(Filevar); Блок закрывает файл Filevar.

*: WriteBlock(Filevar,Value,Leng); Блок записывает вещественное, целое, или строковое Value в файл Filevar. Leng - длина поля вывода, по умолчанию - 0.

*: WritelnBlock(Filevar,Value,Leng); Блок записывает вещественное, целое, или строковое Value с концом строки в файл Filevar. Leng - длина поля вывода, по умолчанию - 0.

*: WritelnBlock(Filevar); Блок записывает конец строки в файл Filevar.

*: ReadBlock(Filevar,Value); Блок читает вещественное или целое число Value из файла Filevar.

*: ReadlnBlock(Filevar,Value); Блок читает вещественное или целое число Value с концом строки из файла Filevar.

*: ReadBlock(Filevar,Value); Блок читает строку Value с концом строки из файла Filevar.

*: ReadlnBlock(Filevar); Блок читает конец строки из файла Filevar.

Категорически не рекомендуется открывать и закрывать файлы прямо в ходе моделирования. Гораздо лучше, если вы откроете файлы в процедуре Initial, а закроете в CloseAllObj.

В принципе, для вывода в текстовые файлы информации типа отчетов «россыпью», можно использовать следующие процедуры, которые, впрочем, вряд ли найдут широкое применение.

ObjName.ReportFile(FileName); Создает отчет по X - параметрам объекта. Здесь FileName – имя файла для вывода.

ObjName.GroupReportFile(FileName); Создает полный отчет по заявкам группы. FileName – имя файла.

ReportBlocks(FileName); Выводит отчет о блоках в указанный файл.

ReportCurrentList(FileName); Выводит отчет о списке текущих событий в указанный файл.

ReportFutureList(FileName); Выводит отчет о списке будущих событий в указанный файл.

ReportSynchronizeList(FileName); Выводит отчет о списке синхронизации в указанный файл.

ReportInterruptList(FileName); Выводит отчет о списке прерываний в указанный файл.

ReportUserList(FileName); Выводит отчет о списках пользователя в указанный файл.

ReportRPList(FileName); Выводит отчет о RP -параметрах в указанный файл.

ReportIPList(FileName); Выводит отчет о IP -параметрах в указанный файл.

ReportSPList(FileName); Выводит отчет о SP -параметрах в указанный файл.

ReportBPList(FileName); Выводит отчет о BP -параметрах в указанный файл.

В качестве примеров работы с вводом и выводом данных, рассмотрим следующие задачи.

Пусть заявки первого типа поступают на обслуживание в систему с интервалом int1±int2 секунд. Они проходят вначале через num канальное устройство Nac, где обслуживаются в течение, time1±time2 сек. А затем они проходят через одноканальное устройство F, где обслуживаются в течение, time3±time4 сек, и покидают модель.

Пусть также заявки второго типа поступают на обслуживание в систему с интервалом int3±int4 секунд. Они проходят через num канальное устройство Nac, где обслуживаются в течение, time5±time6 сек, занимая по 2 канала. Затем они покидают модель. Промоделировать обслуживание count заявок. Текст такой модели, мог бы иметь, например, следующий вид:

```
 {~vb} Var
{/F} F:Tfacility;
{/Nac} Nac:Tstorage;
{/QF} QF:Tqueue;
{/QNac} QNac:Tqueue;
{/G0} G0:Tgenerate;
{/G1} G1:Tgenerate;
 int1:Double;
 int2:Double;
 int3:Double;
 int4:Double;
```

```
 time1:Double;
 time2:Double;
 time3:Double;
 time4:Double;
 time5:Double;
 time6:Double;
 num:Integer;
 count:Integer;
input:textfile;
  {~ve}
  {~pfe}
  {~ib} procedure Initial;begin
 Assignfile(input,'test1.txt');
 reset(input);
readln( input,int1,int2, int3, int4);
readln( input,time1, time2, time3, time4, time5, time6 );
readln( input, num, count);
closefile(input);
 SetStart(count);
{/F}  init(F,'F');
{/Nac}  init(Nac,' Nac',num);
{/QF}  init(QF,' QF');
{/QNac}  init( QNac,' QNac');
{/G0}  init(G0,' G0',G0_,int1,int2);
{/G1}  init(G1,' G1',G1_,int3,int4);
  {~ie}end;
  {~mtb}procedure ModelTxt;begin case ActiveBlock of
{/G0} ::G0_   *:G0.generate(int1,int2);
     *: QNac.Queue ;
     *: Nac.Enter ;
     *: QNac.Depart ;
     *: Advance(time1,time2) ;
     *: Nac.Leave ;
     *: QF.Queue ;
     *: F.Seize ;
     *: QF.Depart ;
     *: Advance(time3,time4) ;
      *: F.Release ;
     *: Terminate(1) ;
{/G1} ::G1_   *:G1.generate(int3,int4);
     *: QNac.Queue ;
     *: Nac.Enter(2) ;
     *: QNac.Depart ;
    *: Advance(time5,time6) ;
     *: Nac.Leave(2) ;
     *: Terminate ;
```

```
{~mte} end;end;
{~mb} procedure Simulation;  begin
  start(Gettg1);
{~me}end;
{~rb} procedure Report;begin
reportval(int1,'int1 = ');
reportval(int2,'int2 = ');
reportval(int3,'int3 = ');
reportval(int4,'int4 = ');
reportval(time1,'time1 = ');
reportval(time2,'time2 = ');
reportval(time3,'time3 = ');
reportval(time4,'time4 = ');
reportval(time5,'time5 = ‘ );
reportval(time6,'time6 = ');
reportval(num,'num = ');
reportval(count,'count = ');
{~re} end;
{~cab} procedure CloseAllObj;begin
{~cae} end;
```

По этому тексту следует дать ряд комментариев. Переменные не следует вставлять с помощью пункта Insert Object, а нужно просто описать в разделе Var. Это обусловлено тем, что чтение и запись этих переменных при инициализации и закрытии модели могут создать проблемы взаимодействия с файлом ‘Test1.txt’. По этой же причине, в процедуру report следует самостоятельно вставить строки вида reportval(int1,'int1 = '); по каждой переменной.

Они обеспечат вывод значений этих переменных в отчет. Обратите также внимание, что открытие файла происходит в процедуре initial а закрытие – в процедуре closeallobj модели.

Содержимое файла ‘Test1.txt’, могло быть следующим.

100 50 1000 100 int1 int2 int3 int4
400 100 90 20 350 200 time1 time2 time3 time4 time5 time6
5 1000 num count

Здесь данные расположены в соответствии с операторами чтения. А тексты вида int1 int2 int3 int4 на самом деле читаться не будут, так как чтение выполняется операторами readln. Сами такие тексты просто поясняют смысл чисел, введенных в строке. В принципе, подобный подход рекомендуется во всех случаях, когда данные вводятся из файла. Это поможет вам избежать ошибок при вводе данных.

Результирующая статистика по модели имеет вид.

```
{/G0} {::G0_}   1:G0.generate(int1,int2);
      2: QNac.Queue ;
      3: Nac.Enter ;
      4: QNac.Depart ;
      5: Advance(time1,time2) ;
      6: Nac.Leave ;
      7: QF.Queue ;
      8: F.Seize ;
      9: QF.Depart ;
      10: Advance(time3,time4) ;
```

```
       11: F.Release ;
      12: Terminate(1) ;
{/G1} {::G1_}   13:G1.generate(int3,int4);
      14: QNac.Queue ;
      15: Nac.Enter(2) ;
      16: QNac.Depart ;
     17: Advance(time5,time6) ;
      18: Nac.Leave(2) ;
      19: Terminate ;

StartTime         0.00000 EndTime     100285.63870
          BLOCK Report
    Location   Entries   Current
         1      1006         0
         2      1006         3
         3      1003         0
         4      1003         0
         5      1003         3
         6      1000         0
         7      1000         0
         8      1000         0
         9      1000         0
        10      1000         0
        11      1000         0
        12      1000         0
        13       100         0
        14       100         0
        15       100         0
        16       100         0
        17       100         1
        18        99         0
        19        99         0
           Report CURRENT LIST
       TRAN     Assem        Pr   Current      Next       TimeStamp       TimeEnter Interrupt
      1105      1105         0         2         3    100110.08633    100110.08633         0
      1106      1106         0         2         3    100171.45118    100171.45118         0
      1107      1107         0         2         3    100277.95732    100277.95732         0
           Report FUTURE LIST
       TRAN     Assem        Pr   Current      Next       TimeStamp         EndTime Interrupt
      1101      1101         0         5         6     99773.24471    100326.59094         0
      1093      1093         0        17        18     99762.47650    100356.53992         0
      1108      1108         0         0         1    100370.84901    100370.84901         0
      1103      1103         0         5         6     99857.45307    100399.31281         0
      1104      1104         0         5         6     99998.65111    100588.43452         0
      1102      1102         0         0        13    100821.46731    100821.46731         0
  Facility   Entries   Current         Utility        AverTime    NextGo       TRAN      TRANP
NumObj
         F      1000         0         0.89906        90.16301         0          0      0
3
   Storage   Entries   Current       Max       Min        AverStor        AverTime
Utility
       Nac      1203         5         5         2         4.65968       388.44448
0.93194
    NextGo       TRAN             AC1    ACapac  Capacity    NumObj
         0      1104    100196.99771     5.000         5         4
     Queue   Entries   Current       Max      Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
        QF      1000         0         5       180         0.89434        89.68897
109.37680         5
     Queue   Entries   Current       Max      Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
      QNac      1106         3         7       410         1.06218        96.31193
153.04740         6
int1  =      100.0000000
int2  =       50.0000000
int3  =     1000.0000000
int4  =      100.0000000
time1  =      400.0000000
time2  =      100.0000000
time3  =       90.0000000
```

```
time4  =       20.0000000
time5  =      350.0000000
time6  =      200.0000000
num  =           5
count  =        1000
```

В тех случаях, когда данные из модели выводятся в файл, проблем с файлами заметно меньше. Рассмотрим следующую задачу

Пусть интервал поступления заявок в одноканальное устройство равен в среднем 110 и распределен по экспоненциальному закону.

Пусть время обслуживания для 10% заявок равно 100, для 20% заявок оно равно 80, для 40% заявок оно равно 120, а для остальных 30% - время обслуживания равно 60. Эту задачу мы уже рассматривали ранее, но сейчас выведем в файл ‘Test2.txt’ информацию по обслуживанию 50 заявок в виде. Номер заявки, время обслуживания.

Тогда модель работы такого устройства может иметь вид.

```
  {~vb} Var
{/q2} q2:Tqueue;
{/fu} fu:Tfacility;
{/tbl} tbl:Ttable;
{/ng1} ng1:Tgenerate;
out:textfile;
  {~ve}
  {~pfe}
  {~ib} procedure Initial;begin
 SetStart(50);
{/q2}  init(q2,’q2’);
{/fu}  init(fu,’ fu’);
{/tbl}  init(tbl,’ tbl’,0,10,20);
{/ng1}     init(ng1,’ ng1’,ng1_,Exponential(110));
  {~ie}end;
  {~mtb}procedure ModelTxt;begin case ActiveBlock of
{/ng1} ::ng1_   *:ng1.generate(300,50);
    *:q2.queue;
    *:fu.seize;
    *:q2.depart;
    *: mark ;
    *:advance(DFun(random,[0.1,0.3,0.7,1],[100,80,120,60]));
    *:fu.release;
    *:tbl.tabulate(TranAge);
    *: writeblock(out,'Tran= ') ;
    *: writeblock(out,Tran) ;
    *: writeblock(out,'m1= ') ;
    *: writelnblock(out,TranAge) ;
    *:terminate(1);
  {~mte} end;end;
  {~mb} procedure Simulation;  begin
  Assignfile(out,'test2.txt');
  Rewrite(out);
```

```
 {/sh}    start(Getterm);
 show(tbl);
show(tbl,2,1,5);
closefile(out);
  {~me}end;
  {~rb} procedure Report;begin
  {~re} end;
  {~cab} procedure CloseAllObj;begin
  {~cae} end;
```

Результирующая статистика в данном случае интереса не представляет, а содержимое выходного файла имеет вид. Отметим только, что в данной модели содержимое этого файла следует смотреть только после закрытия модели. Если вы хотите посмотреть ее раньше, то процедуру закрытия файла следует переместить в процедуру **Simulation** и расположить после процедуры **start.**

```
Tran= 1 m1=  6.0E+0001
Tran= 2 m1=  1.2E+0002
Tran= 3 m1=  1.2E+0002
Tran= 4 m1=  6.0E+0001
Tran= 5 m1=  1.2E+0002
Tran= 6 m1=  6.0E+0001
Tran= 7 m1=  1.2E+0002
Tran= 8 m1=  1.2E+0002
Tran= 9 m1=  6.0E+0001
Tran= 10 m1=  1.2E+0002
Tran= 11 m1=  6.0E+0001
Tran= 12 m1=  1.2E+0002
Tran= 13 m1=  6.0E+0001
Tran= 14 m1=  6.0E+0001
Tran= 15 m1=  8.0E+0001
Tran= 16 m1=  1.2E+0002
Tran= 17 m1=  6.0E+0001
Tran= 18 m1=  1.2E+0002
Tran= 19 m1=  6.0E+0001
Tran= 20 m1=  6.0E+0001
Tran= 21 m1=  1.2E+0002
Tran= 22 m1=  1.2E+0002
Tran= 23 m1=  1.2E+0002
Tran= 24 m1=  8.0E+0001
Tran= 25 m1=  1.2E+0002
Tran= 26 m1=  6.0E+0001
Tran= 27 m1=  8.0E+0001
Tran= 28 m1=  6.0E+0001
Tran= 29 m1=  1.0E+0002
Tran= 30 m1=  8.0E+0001
Tran= 31 m1=  1.0E+0002
Tran= 32 m1=  1.2E+0002
```

```
Tran= 33 m1=  1.2E+0002
Tran= 34 m1=  1.2E+0002
Tran= 35 m1=  1.2E+0002
Tran= 36 m1=  1.2E+0002
Tran= 37 m1=  1.2E+0002
Tran= 38 m1=  8.0E+0001
Tran= 39 m1=  8.0E+0001
Tran= 40 m1=  8.0E+0001
Tran= 41 m1=  6.0E+0001
Tran= 42 m1=  6.0E+0001
Tran= 43 m1=  6.0E+0001
Tran= 44 m1=  8.0E+0001
Tran= 45 m1=  6.0E+0001
Tran= 46 m1=  1.2E+0002
Tran= 47 m1=  6.0E+0001
Tran= 48 m1=  6.0E+0001
Tran= 49 m1=  1.0E+0002
Tran= 50 m1=  8.0E+0001
```

Если вы хотите выводить данные в отчет не с помощью Reporval, а с помощью операторов Write(FileVar, Value_List); Writeln(FileVar, Value_List); то вместо FileVar следует указывать имя файла вывода для отчета, а именно, FileRep.

35 Основы управления моделью и развитие Object GPSS

Для управления моделью используются процедуры переопределения объектов, вывода результатов в той или иной форме, рассмотренные ранее, а также следующие процедуры.

Start(Tg); Задает начальное значение параметру TG1 и запускает модель.

ResetModel; Выполняет сброс модели. Точнее она сбрасывает в ноль статистику, накопленную по устройствам, очередям, спискам пользователя, таблицам и блокам. Однако она не удаляет заявок из модели.

Подробнее действие процедуры ResetModel можно описать следующим образом:

значение условного относительного времени (C1) устанавливается в ноль;

значение условного абсолютного времени (AC1) остается неизменным;

датчик псевдослучайных чисел остается неизменным;

значения X- параметров и переменных не изменяются.

Счетчики блоков (Total(j)) полагаются равными (Current(j)). Временные интегралы устройств устанавливаются в ноль.

Временные интегралы содержимого многоканальных устройств устанавливаются в ноль. Счетчики числа входов в многоканальное устройство и максимальное содержимое многоканального устройства устанавливаются в соответствии с текущим числом занятых каналов в данный момент времени. Временные интегралы всех очередей сбрасываются в ноль.

Счетчики входов в очередь и максимальное содержимое очередей (QMj) полагаются равными текущей длине очереди. В таблицах стираются накопленные статистические данные.

Временные интегралы списков пользователя сбрасываются в ноль. Счетчик числа входов в список пользователя и максимальное содержимое устанавливаются равным текущей длине списка.

ClearModel(Clear); Очищает модель от всех данных при Clear =True . При Clear =False - в модели значения всех переменных и всех X - параметров остаются неизменными.

Следующие процедуры могут использоваться в процедуре Repot.

Reportval(Varia,Name); Выводит значение вещественного, целого, строкового, или логического типа в отчет, Name - заголовок.

Эти процедуры могут использоваться в процедуре Initial.

InitVar(Varia,Value) Устанавливает начальное значение Value для переменной Varia. Переменная и выражение должны быть одного типа. Тип может быть вещественным, целым, строковым, или логическим. Возможно, читает ее значение, из файла, сохранившего состояние модели к концу моделирования.

InitRand(Value); Устанавливает начальное целое значение для генератора случайных чисел. Эта процедура может применяться для гарантированного повторения одного и того же процесса моделирования. Для этого нужно вызвать ее в самом начале процедуры Initial, по крайней мере, до всех процедур инициализации генераторов заявок.

Для полноценного управления моделью может быть недостаточно возможностей имеющихся в модели процедур. Самое простое – это, возможно, вам понадобится несколько различных процедур в стиле Simulation. Или, допустим, вы захотите выводить гистограммы таблиц по одной. Тогда, возможно, вам понадобится иметь несколько пользовательских процедур. Чтобы иметь возможность вызывать такие процедуры, на главной странице модели имеется выпадающий список подписанный Custom Procedure. Из этого списка можно вызывать любую пользовательскую процедуру типа Simulation.
Эти процедуры должны быть описаны в модели, и не иметь параметров.

Далее, в процедуре Initial вам следует вызвать процедуру

CustomProc(st,apr); она создает перечень заголовков для вызова процедур пользователя и перечень имен самих процедур. Перечни записываются в квадратных скобках через запятую. После этого, на главной странице вы можете вызывать эти процедуры.

Процедура

SetPage(Page); - переключает приложение на нужную. Именованные константы для страниц.
spSimulation, ReportSetting, spReport, spLine, spValues, spBar, spAbout, spReserve ,spLicence

Если вам недостаточно тех блоков, процедур и функций, которые есть в модели, то нужные вам процедуры и функции просто описываются в начальной части модели, сразу после описания ее переменных.
Для построения блоков следует учитывать, что блок – это особая процедура, которая корректно работает со списками заявок. Поэтому, новые блоки должны строиться как особые процедуры, которые при любых значениях их параметров выполняют ровно один из ранее определенных блоков. Чаще всего новые блоки строятся на основе блока MoveNext, который, не делая ничего полезного, продвигает заявку в следующий блок. После того, как вы построили новый блок, вы можете пользоваться им наравне с блоками системы.

Приведем примеры применения процедуры CustomProc(st,apr), которая позволяет иметь несколько планов работы с моделью, а также примеры создания новых блоков.

В парикмахерскую приходят клиенты для стрижки с интервалом 16+-6 минут. Обслуживание клиента занимает 15+-3 минуты.

Промоделировать 1000 дней работы парикмахерской по 8 часов в день, учитывая, что по окончании рабочего дня новые клиенты на обслуживание не принимаются, а обслуживаются только те, которые уже есть в парикмахерской. В данной модели имеется 3 пользовательских процедуры, которые позволяют создать и просмотреть гистограммы таблиц, а также график, созданный в динамических массивах типа ADouble.

```
  {~vb} Var
{/ge} ge:Tgenerate;
{/ge1} ge1:Tgenerate;
SHUTDOWN:Boolean;
{/JOEQ} JOEQ:Tqueue;
{/JOE} JOE:Tfacility;
{/tb} tb:Ttable;
{/tab0} tab0:Ttable;
{/B0} BX,BY:ADouble;
  {~ve}
procedure p1;
begin
   show(tb,0);
   setPage(spbar);
end;
procedure p2;
begin
   show(tab0,1);
   setPage(spbar);
end;
procedure p3;
begin
  CShow(bX,BY,1);
  bX := Nil;
  bY := Nil;
  setPage(spline);
end;
  {~pfe}
  {~ib} procedure Initial;begin
SETSTART(1000);
{/ge}  init(ge,' ge',ge_,16,6);
{/ge1}  init(ge1,' ge1',ge1_,480);
  SHUTDOWN:=false;
{/JOEQ}  init(JOEQ,' JOEQ');
{/JOE}  init(JOE,' JOE');
```

```
{/tb}  init(tb,' tb',0,1,100);
{/tab0}  init(tab0,' tab0',0,1,100);
   CustomProc(['Tb','Tab0','Line'],[p1,p2,p3]);
  {~ie}end;
  {~mtb}procedure ModelTxt;begin case ActiveBlock of
{/ge} ::ge_   *:ge.generate(16,6); // ARRIVALS EVERY 18 +- 6 MINUTES
   *: TEST([NOT SHUTDOWN,true],[!*,go]);  // NOT AFTER SHUTDOWN
   *: JOEQ.QUEUE; // GET IN LINE
   *: JOE.SEIZE;  // GRAB THE BARBER
   *: JOEQ.DEPART;  //  EXIT LINE
   *: ADVANCE (15,3);  // HAIRCUT TAKES 15 +- 3 MINUTES ;
   *: JOE.RELEASE;   // FREE THE BARBER ;
   *: tb.tabulate(TranAge);
   *: TERMINATE(0);  // EXIT THE SHOP ;
::go  *: TERMINATE(0);  // EXIT THE SHOP ;
{/ge1} ::ge1_   *:ge1.generate(480);  //  RUN FOR 800 HRS (IN MINUTES)
   *:LET(SHUTDOWN,TRUE);  //LOGIC S SHUTDOWN ENTER
SHUTDOWN MODE
   *:TEST([(JOEQ.L=0) AND (JOE.L=0)],[!*]);   // DRAIN QUEUE,WAIT
UNTIL JOE IS IDLE
   *:LET(SHUTDOWN,false);
//   *:topoint(1,aclock,tb.b);
   *:  RAadd(BX,[aclock])  ;
   *:  RAadd(BY,[tb.A])  ;
   *: tab0.tabulate(TranAge);
   *:tostring(0,inttostr(term));
   *:currentblocks;
   *:TERMINATE( 1); // SHUT DOWN
  {~mte} end;end;
  {~mb} procedure Simulation;  begin
   Linetitle(1,'JOEQ.a');
   start(GetTerm);
  {~me}end;
  {~rb} procedure Report;begin
  reportval(SHUTDOWN,'SHUTDOWN = ');
  {~re} end;
  {~cab} procedure CloseAllObj;begin
  {~cae} end;
```

В данном случае итоговая статистика не представляет интереса, просто по окончании моделирования можно просмотреть соответствующие результаты с помощью пользовательских функций.

Рассмотрим пример создания пользовательских функций.

Пусть в систему поступают заявки со средним интервалом 120, распределенным по экспоненциальному закону. Пусть с вероятностью 0.4 выбирается для обслуживания устройство Kass, а с вероятностью 0.6 - устройство Kass1. В устройстве Kass заявка обслуживается за 150+-100 единиц времени, а в устройстве Kass1 – за среднее время 200,

распределенное по экспоненциальному закону. Промоделировать обслуживание 1000 заявок. Модель такой системы с применением пользовательского блока выглядит, например, следующим образом.

```
  {~vb} Var
{/QKass} QKass:Tqueue;
{/Kass} Kass:Tfacility;
{/Gen} Gen:Tgenerate;
{/kass1} kass1:Tfacility;
K:array[1..2] of Tfacility;
  {~ve}
{/Proc0}   procedure  AddAdvance(i:integer)   ;
    begin
      if i=1 then advance(150,100) else
        advance(exponential(200));
     end;
  {~pfe}
  {~ib} procedure Initial;begin
  SetStart(1000);
{/QKass}  init(QKass,' QKass');
{/Kass}  init(Kass,' Kass');
{/Gen}  init(Gen,' Gen',Gen_,Exponential(120));
{/kass1}  init(kass1,' kass1');
k[1]:=kass;
k[2]:=kass1;
  {~ie}end;
  {~mtb}procedure ModelTxt;begin case ActiveBlock of
{/Gen} ::Gen_   *:Gen.generate(Exponential(120));
     *: Qkass.Queue ;
     *: IPlet(1,[IByBool([random<0.4,true],[1,2])]) ;
     *:K[Ip(1)].Seize ;
    *: Qkass.Depart ;
    *: AddAdvance(ip(1)) ;
    *: K[Ip(1)].release ;
    *: ToPoint(2,aclock,Qkass.A) ;
   *: ToPoint(1, aclock,Kass1.A) ;
   *: ToPoint(0, aclock,Kass.A) ;
    *: terminate(1) ;
  {~mte} end;end;
  {~mb} procedure Simulation;  begin
   start(Getterm);
  {~me}end;
  {~rb} procedure Report;begin
  {~re} end;
  {~cab} procedure CloseAllObj;begin
  {~cae} end;
```

Выходная статистика такой модели имеет следующий вид, и ее содержание подтверждает ее предполагаемые свойства.

```
{/Gen} {::Gen_}   1:Gen.generate(Exponential(120));
     2: Qkass.Queue ;
     3: IPlet(1,[IByBool([random<0.4,true],[1,2])]) ;
     4:k[Ip(1)].Seize ;
    5: Qkass.Depart ;
    6: AddAdvance(ip(1)) ;
    7: k[Ip(1)].release ;
    8: ToPoint(2,aclock,Qkass.A) ;
   9: ToPoint(1, aclock,Kass1.A) ;
   10: ToPoint(0, aclock,Kass.A) ;
    11: terminate(1) ;

StartTime         0.00000 EndTime     120201.76613
          BLOCK Report
    Location   Entries   Current
         1      1007         0
         2      1007         0
         3      1007         6
         4      1001         0
         5      1001         0
         6      1001         1
         7      1000         0
         8      1000         0
         9      1000         0
        10      1000         0
        11      1000         0
          Report CURRENT LIST
       TRAN     Assem        Pr   Current      Next       TimeStamp       TimeEnter Interrupt
      1001      1001         0         3         4    119298.68313    119298.68313         0
      1002      1002         0         3         4    119329.33579    119329.33579         0
      1003      1003         0         3         4    119588.56306    119588.56306         0
      1004      1004         0         3         4    119898.65460    119898.65460         0
      1006      1006         0         3         4    120063.46402    120063.46402         0
      1007      1007         0         3         4    120119.99714    120119.99714         0
          Report FUTURE LIST
       TRAN     Assem        Pr   Current      Next       TimeStamp         EndTime Interrupt
      1008      1008         0         0         1    120386.04053    120386.04053         0
      1000      1000         0         6         7    119295.66130    120543.97934         0
          Report  IP LIST (  Integer  )
       TRAN       Num   Integer
      1000         1         6
      1001         1         6
      1002         1         6
      1003         1         6
      1004         1         6
      1006         1         4
      1007         1         4
     Queue   Entries   Current       Max      Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
     QKass      1007         6        31       205        14.09321      1682.25271
2112.25496         3
  Facility   Entries   Current         Utility        AverTime    NextGo       TRAN      TRANP
NumObj
      Kass       413         0         0.53440       155.53386         0          0      0
4
  Facility   Entries   Current         Utility        AverTime    NextGo       TRAN      TRANP
NumObj
     kass1       588         1         0.98281       200.90993         0       1000      0
6
```

Такая модель выглядит более наглядной за счет использования пользовательских функций. Если бы ее построить «в лоб», то она оказалась бы менее наглядной.

Приложение 1. Справочник по Object GPSS.

Внимание, категорически запрещается удаление или редактирование строк, содержащих комментарий следующего вида {~текст}. Любая модификация этих строк может разрушить работоспособность модели!! Эти строки изначально должны быть в модели. Если модель испорчена, то нужно выполнить пункт "New", и строить ее заново.

Построение модели начинается с пункта “New” или сразу после запуска. Вначале надо вставить в модель все нужные объекты, (пункт “Insert Object”) а затем в процедуре ModelTxt дописать текст модели. Далее модель следует сохранить в файле на диске, и с ней можно начинать работу. Если позднее понадобится вставить новые объекты или удалить имеющиеся, то это всегда можно сделать. Подробности этого процесса описаны ниже. Для работы с системой следует знать любую версию языка GPSS, основы программирования на любом языке высокого уровня, например, Basic, Pascal, C++ и так далее. Кроме того, нужно проанализировать несколько примеров из папки Models. При разработке первых простых моделей, не следует вручную изменять содержимое каких – либо процедур или разделов модели, кроме ModelTxt.

Любая модель состоит из следующих разделов.

1. Раздел описаний личных переменных и объектов. Он заканчивается строкой

{~ve}

2. Раздел описаний личных процедур и функций, которые используются разработчиком в модели. Он заканчивается строкой

{~pfe}

Начало

3. Раздел инициализации (процедура Initial). Здесь создаются нужные объекты, и задаются начальные значения переменным. Здесь же может быть выполнена любая другая подготовительная работа для начала моделирования. Он заканчивается строкой.

{~ie}end;

Начало

4. Раздел собственно модели (процедура ModelTxt). Здесь последовательно располагаются блоки модели. (Фактически это специальные процедуры Object GPSS). Они размещаются после строки

{~mtb}procedure ModelTxt;begin case ActiveBlock of

Блоки модели должны иметь вид:

::метка *:блок; или

*:блок;

В тексте модели метки указываются непосредственно, и могут входить в корректные выражения целого типа. После конвертирования, метки представляются как константы целого типа, а конструкции *: - как номера блоков. Вместо метки следующего блока можно указывать конструкцию !* . Она также может участвовать в выражениях.

Если блок модели является составным, то есть, составлен из нескольких операторов языка Pascal, то он обрамляется ключевыми словами Begin End, и должен работать так, чтобы непременно пройти ровно один блок Object GPSS, лучше всего блок Block.

При создании новых блоков их желательно оформлять их по образцу:

Begin

```
  If ErrBl('BlockName') then exit; // Не обязательно!!
  DoSomething;
  Nop;
End;
```

Здесь ErrBl('BlockName') - функция проверки на правильность положения блока.

В описании, перед именем каждого блока помещен текст вида *: . Кроме того, описания всех блоков расположено в начале соответствующих классов, а для модуля AddModelUnit - в начале модуля.

При развитии системы, рекомендуется придерживаться тех же правил для вновь созданных блоков.

Раздел заканчивается строкой

{~mte} end;end;

Начало

5. Раздел манипуляций с моделью (процедура Simulation). Здесь указаны конкретные команды манипуляции с моделью. Здесь можно настраивать модель, и готовить ее к новому моделированию. В частности, можно менять значения X - параметров, функций, многоканальных устройств, таблиц, и переменных, выводить графики и гистограммы и делать многое другое. Раздел завершается строкой

{~me}end;

6. Раздел формирования дополнительных элементов отчета, например, по выражениям или переменным. (процедура Report). Раздел завершается строкой

{~re} end;

7. Раздел удаления динамически созданных пользователем переменных (процедура CloseAllObj). Раздел завершается строкой

{~cae} end;

При определении объектов, следует использовать пункт меню "Insert Obj". Тогда имя объекта должно вводиться в соответствующем поле приложения, а тип объекта выбирается из списка. В принципе, можно согласиться с автоматически сгенерированным именем переменной или объекта. Все нужные вызовы процедур для работы с объектом автоматически размещаются в соответствующих разделах. Иногда эти вызовы имеют различные наборы аргументов, тогда следует их ввести в поле аргументов.

Начало

Для удаления всех вызовов и описания объекта, вставленного с помощью пункта "Insert Obj", удобно воспользоваться пунктом "Delete Obj". Тогда нужно поместить курсор в строку, содержащую сгенерированный для объекта текст, и щелкнуть по "Delete Obj". За счет комментария в начале строки, будут удалены все строки текста, сгенерированные для объекта пунктом " Insert Obj".

Начало

Пример полного текста модели приводится ниже. Фактически, при использовании пунктов "Insert Obj" и "Delete Obj" ваши затраты времени на построение модели сравнимы, со временем на ввод собственно текста модели в разделе 4.

```
{~vb} Var
{/Fac} Fac:TFacility;
{/Que} Que:TQueue;
{/Tab} Tab:TTable;
```

```
{/Gen} Gen:TGenerate;
  {~ve}
  {~pfe}
  {~ib} procedure Initial;begin
 setstart(10000);
{/Fac} Init(Fac,'Fac');
{/Que} Init(Que,'Que');
{/Tab} Init(Tab,'Tab',0,100,100);
{/Gen} Init(Gen,'Gen',Gen_,Exponential(120));
  {~ie}end;
  {~mtb}procedure ModelTxt;begin case ActiveBlock of
{/Gen} ::Gen_ *:Gen.Generate(Exponential(120));
    *:Que.Queue;
    *:Fac.Seize;
    *:Que.Depart;
    *:Advance  (Exponential(100));
    *:Fac.Release;
    *:Tab.Tabulate (TranAge);
    *:Terminate (1);
  {~mte} end;end;
  {~mb} procedure Simulation;  begin
 start(GetTerm);
 Show(Tab,1);
  {~me}end;
  {~rb} procedure Report;begin
  {~re} end;
  {~cab} procedure CloseAllObj;begin
  {~cae} end;
```

При отладке модели наиболее опасной ошибкой является появление блоков вне процедуры ModelTxt или появления вызовов процедур, не являющихся блоками в процедуре ModelTxt. Большую часть этих ошибок обнаруживает система, а остальные – обычно вызывают системные прерывания.

Отметим, что конвертер корректно обрабатывает все виды комментариев, допустимых в языка Object Pascal.

Большая часть элементов оборудования в GPSS являются объектами.

Всякий объект из класса должен быть вначале описан, затем создан, затем его можно использовать, и, наконец, он должен быть уничтожен перед закрытием приложения.

Почти все процедуры и функции любого класса, кроме процедур Init, должны вызываться в форме ObjName.ProcedureName(Argument List) или ObjName.FunctionName(Argument List) . Здесь ObjName - это то имя, под которым описан объект.

Для работы с системой, следует знать одну из версий GPSS, например, GPSS - World, и ознакомиться с примерами, из папки Models.

Работа по созданию конкретной модели требует выполнения следующих шагов.

1. Запустить приложение Converter.exe

2. Открыть в нем имеющуюся модель, или создать новую. Если создана новая модель, то ее следует сохранить на диске.

3. Выполнить пункт "Run" При выполнении этого пункта текст модели автоматически сохраняется на диске.

Если в файле Model.pas имеются ошибки, то нужно их просмотреть, исправить, и вновь выполнить шаг 3. Обычно строки с ошибками подсвечиваются в модели, когда вы просматриваете сообщения об ошибках, однако часть ошибок, типа повторной декларации переменных могут и не подсвечиваться. Тогда их нужно искать собственно в тексте модели, то есть в процедуре ModelTxt, где они фигурируют как метки блоков. Если ошибок нет, то запускается приложение с конкретной моделью и с ним можно проводить опыты.

Конвертер может выполнять вставку файла. Файл может быть в текстовом формате, или в формате .RTF.

Вставка выполняется по переключателю. При включенном переключателе, перезапись файла при компиляции не происходит.

Вставки могут быть вложенными. Они могут располагаться и в ModelTxt, то есть собственно в модели. Такого сорта вставки собственно компилятором обрабатываются неверно, то есть не создаются номера блоков. Вставка имеет вид

{#I.. FileName } и описывает файл относительно модели (папки с моделью).

Здесь допускается и имя файла относительно диска, то есть можно указывать полный путь.

Соединение моделей позволяет выполнять конструкция вида:

{#U.. FileNameList }. Здесь FileNameList - список файлов моделей через запятую. В этом случае соединяются одноименные разделы файлов с моделями.

Если нужно, чтобы часть текста прцедуры ModelTxt не попала в отчет по моделированию, и по нему не была сформирована статистика, то нужно обрамить эту часть текста символами !- и !+ , которые выключают и включают отображение сведений о блоках и выдачу самого текста блоков. Символы не могут появляться вне прцедуры ModelTxt.

То есть они должны располагаться ниже строки с {~mtb} но выше строки с {~mte}.

Выражения и функции языка Object Pascal в Object GPSS

В Object GPSS можно использовать выражения в качестве операндов,

Вид выражений фактически тот же, что и в Object Pascal. В выражении используют функции модели, метки, константы, знаки операций, вызовы библиотечных функций, и круглые скобки. Выражения должны составляться по правилам элементарной алгебры. Выражения вычисляются слева направо с учетом приоритетов операций. Для вещественных данных используется 8- байтовое представление (тип Double), а для целых - 4 - байтовое (тип Integer). Строковые константы представляют собой текст, заключенный в апострофы, например, ‘это строковая константа’ ‘It is string.’.

Логические константы могут быть только истина (True) и ложь (False). Массивы переменных и объектов могут использоваться для косвенного обращения к их элементам.

Ниже приводятся операции и библиотечные функции, используемые в выражениях.

Арифметические операции.

* - операция арифметического умножения;

/ - операция арифметического деления;

Div - операция деления нацело, используется только для целых данных;

Mod - операция деления по модулю, используется только для целых данных. Ее результатом является остаток от деления нацело;

Shl - оператор сдвига целого числа влево. Он эквивалентен умножению числа на соответствующую степень двойки.

Shr - оператор сдвига целого числа вправо. Он эквивалентен делению числа на соответствующую степень двойки.

\+ - операция арифметического или строкового сложения;

\- - операция арифметического вычитания.

\- знак минус;

Отношения.

Могут соединять пару арифметических или строковых выражений.

Если это действительно выражения, то они должны быть в круглых скобках.

< - меньше;

<= - меньше или равно;

> - больше;

>= - больше или равно;

= - равно;

<> - не равно;

Логические выражения.

Могут соединять логические данные, и, в частности, отношения.

NOT - логическое отрицание: True, если операнд False;

False, если операнд True;

AND - операция логического умножения: True, если оба операнда True, False - в противном случае;

OR - операция логического сложения: True, если хотя один из операндов True, False - в противном случае;

XOR - операция сложения по модулю два: True, если операнды различны, false - в противном случае;

Элементарные функции.

Abs( ) - абсолютное значение операнда;

ArcTan( ) – арктангенс аргумента;

Cos( ) - косинус операнда в радианах;

Exp( ) - экспонента операнда;

Int( ) - целая часть числа, результат - вещественный;

Ln( ) - натуральный логарифм операнда;

Log10 ()- десятичный логарифм операнда;

Pi – предопределенная константа равная пи.

Power(A,B) – возведение в вещественную степень B положительного вещественного числа A. Всегда может быть заменено выражением Exp(Ln(A)*B) .

Random – случайное число, равномерно распределенное в диапазоне от 0 до 1.

Random(Num) – случайное целое число, равномерно распределенное в диапазоне от 0 до Num-1.

RandSeed – целое число, представляет собой текущее значение встроенного генератора случайных чисел.

Round() – вещественное число округляется до целого.

Sin( ) - синус операнда в радианах;

Sqr( ) - квадрат операнда;

Sqrt() – корень квадратный из операнда.

Trunc() усеченное целое, дробная часть отбрасывается.

Tan( ) - тангенс операнда в радианах;

Функции преобразования.

StrToInt() – преобразует строку – в целое число.

IntToStr() – преобразует целое число – в строку.

StrToFloat() – преобразует строку – в вещественное число.

FloatToStr() – преобразует вещественное число – в строку.

Для строк можно выполнять операцию + , которая означает соединение строк, а также операции сравнения.

Порядок действий соответствует следующим приоритетам, расположенным от высшего - к низшему:

Логическое не. (Not)

Умножение, деление, остаток, логическое и. (* / Div Mod Or Shl Shr)

Сложение и вычитание, логическое или. (+ - Or)

Отношения. (< <= > >= = <>)

Кроме выше приведенных функций, имеется большая группа других функций.

Функции и процедуры работы со строками

Эти функции могут использоваться для манипуляций со строками при вводе – выводе данных, а также для формирования значений X и P – параметров.

Кроме ранее рассмотренных, имеется следующий набор функций и процедур для работы со строками:

Функция

Pos(Substr, S)

Она ищет подстроку Substr в строке S. Когда найдена нужная подстрока, то позиция начала совпадения считается результатом, иначе результат равен 0.

Функция

Copy(S,Index,Count:)

Выделяет из строки S, начиная с позиции Index заданное число символов. Полученная строка считается результатом.

Процедура

Delete(S,Index,Count);

В строке S удаляется s Count символов, начиная с символа с указанным номером Index. Полученная строка заменяет S.

Процедура

Insert(Source,S, Index);

В строку S вставляется строка Source, начиная с позиции index. Полученная строка заменяет S.

Процедура

SetLength(DynArray,Length) - определяет количество элементов в созданном процедурой динамическом массиве.

Функция

Length(S) – определяет длину строки S.

Функция

StringReplace(S,OldPattern,NewPattern,Flags)

Возвращает новую строку, которая получена следующим образом. В строке S заменяется многократно подстрока OldPattern на подстроку NewPattern. Флаги, записанные в квадратных скобках, управляют заменой.

Если есть флаг rfIgnoreCase, то при замене не учитывается регистр символов, то есть не различаются большие и малые буквы.

Если есть флаг rfReplaceAll, то заменяются все подстроки, иначе только первая попавшаяся подстрока.

Работа с файлами.

Эти процедуры могут рассматриваться как процедуры управления моделью.

В принципе, для работы с текстовыми файлами используются процедуры самого языка Pasacal. Для этого следует описать текстовый файл как переменную в самой модели. Эта переменная должна иметь тип TextFile.

Далее следует связать эту переменную с файлом на диске и открыть файл для чтения или записи данных.

В первом случае нужно вызвать процедуры

AssignFile(FileVar,FileName);

Reset(FileVar);

А во втором случае нужно вызвать процедуры

AssignFile(FileVar,FileName);

ReWrite(FileVar);

Здесь FileVar – это файловая переменная, а FileName – это строка, содержащая имя файла на диске.

Далее для чтения данных из файла следует использовать процедуры

Read(FileVar, список переменных через запятую);

Readln(FileVar, список переменных через запятую);

Процедура Read читает данные, не обращая внимания на концы строк, а процедура Readln обязательно дочитывает начатую строку до конца, даже если в ней есть данные, которым не сопоставлены переменные.

Тип переменных может быть вещественным, целым, или строковым.

Переменные в указанном порядке будут читаться из файла, если в файле еще есть непрочитанные данные. Вещественные и целые данные должны разделяться пробелами или концами строки, а строковые - должны занимать строку целиком. В процедуре Read не следует указывать строковые переменные.

А в процедуре Readln можно указывать данные любого из трех указанных типов.

Обращаю ваше внимание, что чтение данных возможно только в переменные соответствующих типов, и невозможно чтение в P или X – параметры.

Для проверки закончился ли файл в процессе чтения, можно использовать функцию EOF(FileVar) – которая имеет значение True (Истина), если данные в файле уже закончились.

Для записи данных в файл следует использовать процедуры

Write(FileVar, список выражений с указание ширины полей вывода через запятую);

Writeln(FileVar, список выражений с указание ширины полей вывода через запятую);

Процедура Write записывает данные в одну строку подряд, а процедура Writeln завершает запись концом строки.

Тип выражений может быть вещественным, целым, или строковым.

Выражения будут вычислены и их значения в указанном порядке будут записаны в файл. После каждого выражения может стоять двоеточие и минимальная ширина поля вывода для значения выражения. При выводе, данные могут «слипаться», если ширина поля вывода будет недостаточной для вывода значения. Поэтому рекомендуется включать в список выражений строки из одного или нескольких пробелов в апострофах, или выбирать ширину поля вывода достаточно большой.

После того, как работа с файлом завершена, файл необходимо закрыть с помощью процедуры CloseFile(FileVar). Работа с файлами в процессе моделирования рассматривается в разделе общих процедур и блоков.

Блоки, процедуры и функции языка, "Object GPSS"

Для вызова процедур или функций этого или любого другого класса следует вначале указать имя объекта класса, затем поместить точку, а затем - имя процедуры или функции. Естественно, сам объект вначале должен быть вставлен в модель. Здесь ObjName - это имя объекта. Вместо упоминаемых в описании списков в квадратных скобках, можно использовать динамические массивы соответствующего типа.

Во всех объектах, кроме генераторов и таблиц, имеется блок

*:ObjName.Active (True/false) при True - объект считается работающим и для него продолжается подсчет времени работы, а при false - объект считается не работающим и для него прекращается подсчет времени работы.

Функция ObjName.AV - определяет суммарное время работы объекта.

Внимание: в этом и всех остальных классах процедуры нельзя использовать в качестве блоков, то есть внутри процедуры ModelTxt. Аналогично, запрещено использование блоков вне процедуры ModelTxt.

Общие блоки, процедуры и функции системы.

Блоки

*: Advance(Mean,Dev); Блок задерживает заявку на заданное время. Mean, Dev средний интервал задержки заявки и отклонение от среднего. По умолчанию, параметр Dev равен 0.

Для работы с P – параметрами вместо номера можно использовать переменные, типа Integer или TPName, которым присвоено значение функции Init(st), где St – строка

для идентификации параметра. Примечание: следует избегать использования параметров с номером ' -1'.

Для работы с X и P – параметрами вместо номера можно использовать переменные, типа Integer или TPName, которым присвоено значение функции Init(st), где St – строка для идентификации параметра, а Num- не обязательный номер для идентифицируемого параметра.

*: RPlet([Num List],[Value list],ATran); Блок присваивания списку RP-параметров значений из списка. Списки указываются в квадратных скобках. Присваивание ведется указанной заявке, по умолчанию - активной заявке.

*: IPlet([Num List],[Value list],ATran); Блок присваивания списку IP- параметров значений из списка. Списки указываются в квадратных скобках. Присваивание ведется указанной заявке, по умолчанию - активной заявке.

*: RPAdd([Num List],[Value list],ATran); Блок добавления списку RP-параметров значений из списка. Списки указываются в квадратных скобках. Присваивание ведется указанной заявке, по умолчанию - активной заявке.

*: IPAdd([Num List],[Value list],ATran); Блок добавления списку IP- параметров значений из списка. Списки указываются в квадратных скобках. Присваивание ведется указанной заявке, по умолчанию - активной заявке.

*: SPlet([Num List],[Value list],Value,ATran); Блок присваивания списку SP-параметров значений из списка. Списки указываются в квадратных скобках. Присваивание ведется указанной заявке, по умолчанию - активной заявке.

*: BPlet([Num List],[Value list],ATran); Блок присваивания списку BP-параметров значений из списка. Списки указываются в квадратных скобках. Присваивание ведется указанной заявке, по умолчанию - активной заявке.

*:PLet([Numlist],[Valuelist],ATran); Блок задает набор значений для параметров P списков. Списки указываются в квадратных скобках. Тип параметра определяется типом выражения.

Вместо первого списка можно указать нижнее значение для номеров, запоминаемых значений.

*:PSortNum(ANum, AVal,NumB,NumE,ATran); Блок копирует значения P - параметров и их номеров в динамические массивы AVal и ANum, упорядоченные по номерам параметров. Здесь не обязательные параметры NumB, NumE, ATran - определяют диапазон номеров и номер заявки. Тип параметров определяется массивом AVal.

*: Terminate(Num); Блок уничтожает активную заявку и уменьшает значение TG1 на значение Num. По умолчанию Num равно 0.

*: Priority (Num,ATran); Блок задает новый приоритет Num заявке ATran, по умолчанию - активной заявке.

*: Mark(ATran); Блок обнуляет время жизни заявки, по умолчанию - активной.

*: Let(Varia,Value); Блок присваивает вещественное, целое, строковое, логическое значение Value - переменной Varia. Если Value отсутствует, то оно равно 0, 0, ‘’, False.

*: Add(Varia,Value); Блок увеличивает на вещественное или целое значение Value - переменную Varia. Если Value отсутствует, то она равна 1.

*: Test(Log,NumBl); Блок перехода активной заявки на блок с меткой из Numbl, для первого выражения с истинным значением из Log. И Log, и NumBl - это списки. Они

записываются в квадратных скобках через запятую. Если истинных значений нет, то заявка не входит в блок.

*: Transfer(NumBl,NumIP); Блок перехода активной заявки на блок с меткой Numbl. В параметр IP с номером NumIP - записывается номер следующего блока для возможного возврата. По умолчанию, запись номера не ведется.

*: Transfer(Ver,NumBl1,NumBl2); Блок перехода активной заявки на блок с меткой Numbl1 или Numbl2 с вероятностью Ver. Вероятность соответствует второй метке.

*: Transfer ([Label list]). Обеспечивает переход на одну из меток списка, если один из блоков с меткой примет заявку. Иначе заявка не входит в блок. Ни один из блоков с меткой не должен быть таким же Transfer. Впрочем, в этом нет смысла.

*: Loop(NumIP,NumBl); Блок организации цикла, уменьшает значение IP(Numip) на 1.

Если оно не нулевое, то заявка уходит по метке Numbl, иначе она идет дальше.

*: Buffer; Блок перемещает активную заявку в конец ее приоритетного класса.

*: Split(Num) Блок создает Num заявок соответствующего семейства, и отправляет их по метке Numbl. NumIP - это номер IP - параметра для последовательной нумерации созданных заявок. Если параметр опущен, то заявки не нумеруются. При опущенном параметре Numbl - заявки переходят на следующий блок.

*: Assemble(Num); Блок объединения заданного числа заявок одного семейства в одну заявку - инициатор.

*: Gather(Num); Блок группировки заданного числа заявок одного семейства.

*: Match(NumBl); Блок синхронизации заявок в текущем блоке, и в блоке с данной меткой.

*:Match([listoflabel]); Блок синхронизации заявок в текущем блоке, и в остальных блоках с данными метками.

*: Adopt(NumA1,ATran); Блок устанавливает новый номер семейства активной заявке, или заявке с заданным номером.

*: Nop; Блок без операции.

*: SetTerm(Num); Блок устанавливает новое значение TG1 - счетчика.

*: Execute(Num); Исполняет блок с указанным номером. В косвенно указанном блоке нужно аккуратно пользоваться метками.

*: .Displace(ATran,NumBl,NumRP,NumSP); Блок отправляет заявку с номером ATran- в Блок с меткой Numbl. NumRP - определяет номер RP параметра, где сохраняется оставшееся время задержки. NumSP- номер SP параметра, где записывается обозначение списка, где заявка обнаружена.

Здесь "N" - это отсутствие заявки. "P" - заявка обнаружена в списке прерываний.

"C"- заявка обнаружена в списке текущих событий. "F" - заявка обнаружена в списке будущих событий. "U" - заявка обнаружена в списке пользователя. "S" - заявка обнаружена в списке синхронизации. По умолчанию, соответствующая информация не записывается в P - параметры.

*: FindTran(ATran,NumSPList,NxtIP,CurIP); Блок разыскивает заявку с данным номером. Параметр NumSpList - определяет обозначение списка, записывается в SP(NumSpList). Номер следующего блока и текущего блока, записываются в параметры IP(NxtIP), IP(CurIP). По умолчанию, эти номера не записываются.

*:RPDelete([NumList],ATran); Блок удаляет RP - параметры из списка для данной заявки. По умолчанию - для активной заявки. Список указывается в квадратных скобках.

*:IPDelete([NumList],ATran); Блок удаляет IP - параметры из списка для данной заявки. По умолчанию - для активной заявки. Список указывается в квадратных скобках.

*:SPDelete([NumList],ATran); Блок удаляет SP - параметры из списка для данной заявки. По умолчанию - для активной заявки. Список указывается в квадратных скобках.

*:BPDelete([NumList],ATran); Блок удаляет BP - параметры из списка для данной заявки. По умолчанию - для активной заявки. Список указывается в квадратных скобках. Вместо списка можно указать верхнее и нижнее значение для номеров, удаляемых значений.

*:RPDelete(ATran); Блок удаляет все RP -параметры данной заявки. По умолчанию - для активной заявки.

*:IPDelete(ATran); Блок удаляет все IP -параметры данной заявки. По умолчанию - для активной заявки.

*:SPDelete(ATran); Блок удаляет все SP -параметры данной заявки. По умолчанию - для активной заявки.

*:BPDelete(ATran); Блок удаляет все BP -параметры данной заявки. По умолчанию - для активной заявки.

*:PDelete(ATran); Блок удаляет все P -параметры данной заявки. По умолчанию - для активной заявки.

*: IncTran; Блок увеличивает на 1 номер последней запланированной или выпущенной заявки. Он предназначен для создания фиктивных наборов P -параметров, для которых нет соответствующей заявки. Таким образом, можно в любой момент создать новый набор P -параметров.

*: OpenGetBlock(Filevar,FileName); Блок открывает файл Filevar для чтения. Имя файла - FileName.

*: OpenPutBlock(Filevar,FileName); Блок открывает файл Filevar для записи. Имя файла - FileName.

*: CloseBlock(Filevar); Блок закрывает файл Filevar.

*: WriteBlock(Filevar,Value,Leng); Блок записывает вещественное, целое, или строковое Value в файл Filevar. Leng - длина поля вывода, по умолчанию - 0.

*: WritelnBlock(Filevar,Value,Leng); Блок записывает вещественное, целое, или строковое значение Value с концом строки в файл Filevar.Leng - длина поля вывода, по умолчанию - 0.

*: WritelnBlock(Filevar); Блок записывает конец строки в файл Filevar.

*: ReadBlock(Filevar,Value); Блок читает вещественное или целое значение Value из файла Filevar.

*: ReadlnBlock(Filevar,Value); Блок читает вещественное или целое значение Value с концом строки из файла Filevar.

*: ReadlnBlock(Filevar,Value); Блок читает строку Value с концом строки из файла Filevar.

*: ReadlnBlock(Filevar); Блок читает конец строки из файла Filevar.

*: ToFuture; Блок принудительно завершает просмотр списка текущих событий. Он может быть полезен, в первую очередь, для отладки моделей.

*:Select(List, Log ,Values); Блок записывает в динамический массив номера заявок из списка, для которых условие - функция Log - истинна. List может быть 'C', 'F', 'S', 'I' , в этих случаях просмотр ведется для списка текущих событий, списка будущих событий, списка синхронизации, или списка прерванных заявок. Внимание. В функции выбора из списка текущих событий (Log), функция Tran('C') - это номер заявки из списка текущих заявок, а не активной заявки. Если её номер нужен, то его надо запомнить где-либо заранее.

*:Pause(Tran,Flag); Блок, переводит заявку Tran в паузу по умолчанию, а при Flag=False - заявка вновь потенциально активна.

Функции.

TranAge(ATran) Выдает время жизни заявки с данным номером.

Tran(List) или Tran Без параметра, выдает номер активной заявки. Если параметр List равен ‘F’, ‘S’, ‘I’, то выдается номер активной заявки, которая просматривается в списке будущих событий, списке синхронизации, или в списке прерываний. Фактически, такая форма функции может быть полезна только для использования внутри функций выбора Log для функции Select. (смотри ниже).

Prior(ATran) Выдает приоритет заявки с данным номером, по умолчанию - активной заявки.

Assem(ATran) Выдает номер семейства заявки с данным номером, по умолчанию - активной заявки.

Clock Выдает текущее время моделирования.

Aclock Выдает абсолютное текущее время моделирования.

IntervalToTime(time): преобразует секунды – в строку дни: часы: минуты: секунды.

Term Выдает значение счетчика завершений.

Total(NumBl) Выдает число вхождений заявок в блок с данным номером.

Current(NumBl) Выдает число заявок в блоке с данным номером.

MatchBlock(NumBl) Для текущего блока, определяет наличие заявки того же семейства в блоке с данным номером.

RP(Num,ATran) Выдает вещественное значение RP(Num) по номеру заявки ATran, по умолчанию, используется активная заявка.

IP(Num,ATran) Выдает целое значение IP(Num) по номеру заявки ATran, по умолчанию, используется активная заявка.

SP(Num,ATran) Выдает строковое значение SP(Num) по номеру заявки ATran, по умолчанию, используется активная заявка.

BP(Num,ATran) Выдает логическое значение BP(Num) по номеру заявки ATran, по умолчанию, используется активная заявка.

RPExist(Num,AXn) Определяет, имеется ли RP(Num) для данной заявки ATran, по умолчанию, используется активная заявка.

IPExist(Num,AXn) Определяет, имеется ли IP(Num) для данной заявки ATran, по умолчанию, используется активная заявка.

SPExist(Num,AXn) Определяет, имеется ли SP(Num) для данной заявки ATran, по умолчанию, используется активная заявка.

BPExist(Num,AXn) Определяет, имеется ли BP(Num) для данной заявки ATran, по умолчанию, используется активная заявка.

RPCount(ATran) Определяет количество RP - параметров у заявки с данным номером. По умолчанию - у активной заявки.

IPCount(ATran) Определяет количество IP - параметров у заявки с данным номером. По умолчанию - у активной заявки.

SPCount(ATran) Определяет количество SP - параметров у заявки с данным номером. По умолчанию - у активной заявки.

BPCount(ATran) Определяет количество BP - параметров у заявки с данным номером. По умолчанию - у активной заявки.

InterruptTran(Xn) Определяет, находится ли в состоянии прерывания заявка с данным номером.

MaxTran Определяет, номер последней запланированной или выпущенной заявки.

MaxNumObj – максимальный номер объекта.

BoolToInt(Log) - преобразует логическое значение в целое. True ->1 False ->0

ActiveBlock Выдает номер блока, в который пытается войти заявка.

CurrentBlock(List) Определяет номер текущего блока для просматриваемой в Select заявки из соответствующего списка. Используется только в функции Log для функции Select. Здесь не допускается список прерываний.

NextBlock(List) Определяет номер следующего блока для просматриваемой в Select заявки из соответствующего списка. Используется только в функции Log для функции Select. Здесь не допускается список прерываний.

GetParam(Num) - эта функция определяет объект типа Tparam по его номеру с минусом.

GetFacility(Num) - эта функция определяет объект типа TFacility по его номеру с минусом.

GetQueue(Num) - эта функция определяет объект типа TQueue по его номеру с минусом.

GetStorage(Num) - эта функция определяет объект типа TStorage по его номеру с минусом.

GetTable(Num) - эта функция определяет объект типа TTable по его номеру с минусом.

GetUser(Num) - эта функция определяет объект типа TUser по его номеру с минусом.

GetRGroup(Num) - эта функция определяет объект типа TRGroup по его номеру с минусом.

GetIGroup(Num) - эта функция определяет объект типа TIGroup по его номеру с минусом.

GetSGroup(Num) - эта функция определяет объект типа TSGroup по его номеру с минусом.

GetTGroup(Num) - эта функция определяет объект типа TTGroup по его номеру с минусом.

Номер объекта с минусом позволяет узнать функция ObjName.N.

Процедуры управления моделью.

ParamInList(Yes); определяет способ вывода в отчет значений параметров заявок. По умолчанию, они выводятся как списки параметров для всех заявок, а при Yes=True, по каждой заявке в отдельности.

SetDistance(Dis); устанавливает предельное число блоков, для продвижения одной заявки за один раз. По умолчанию установлена величина 10000. Этот параметр защищает модель от зацикливания.

SelectProc(List, Log ,Values) Процедура записывает в динамический массив номера заявок из списка, для которых условие - функция Log - истинна. List может быть 'C', 'F', 'S', 'I' , в этих случаях просмотр ведется для списка текущих событий, списка будущих событий,
списка синхронизации, или списка прерванных заявок. Внимание. В функции выбора из списка текущих событий (Log), функция Tran('C') - это номер заявки из списка текущих заявок, а не активной заявки. Если её номер нужен, то его надо запомнить где-либо заранее.

RPLetProc ([Num List],[Value list],ATran); RP - параметрам из списка присваивает значения из списка. ATran- номер заявки, по умолчанию, используется активная заявка. Списки указываются в квадратных скобках.

IPLetProc([Num List],[Value list],ATran); IP - параметрам из списка присваивает значения из списка. ATran- номер заявки, по умолчанию, используется активная заявка. Списки указываются в квадратных скобках.

RPaddProc ([Num List],[Value list],ATran); RP - параметрам из списка добавляет значения из списка. ATran- номер заявки, по умолчанию, используется активная заявка. Списки указываются в квадратных скобках.

IPAddProc([Num List],[Value list],ATran); IP - параметрам из списка добавляет значения из списка. ATran- номер заявки, по умолчанию, используется активная заявка. Списки указываются в квадратных скобках.

SPLetProc([Num List],[Value list],ATran); SP - параметрам из списка присваивает значения из списка. ATran- номер заявки, по умолчанию, используется активная заявка. Списки указываются в квадратных скобках.

BPLetProc([Num List],[Value list],ATran); BP - параметрам из списка присваивает значения из списка. ATran- номер заявки, по умолчанию, используется активная заявка. Списки указываются в квадратных скобках.

PLetProc([Numlist],[Valuelist],ATran); задает набор значений для параметров P списков. Списки указываются в квадратных скобках. Тип параметра определяется типом выражения.

Вместо первого списка можно указать нижнее значение для номеров, запоминаемых значений.

PSortNumProc (ANum, AVal,NumB,NumE,ATran); Процедура копирует значения P - параметров и их номеров в динамические массивы AVal и ANum, упорядоченные по номерам параметров. Здесь не обязательные параметры NumB, NumE, ATran - определяют диапазон номеров и номер заявки.

PSortValProc(ANum, AVal, ATran); Процедура копирует значения P - параметров и их номеров в динамические массивы AVal и ANum, упорядоченные по значениям параметров. Здесь не обязательный параметр ATran - определяют номер заявки.

RPADeleteProc ([Num List],ATran); Удаляет RP- параметры из списка для заявки ATran.

IPADeleteProc ([Num List],ATran); Удаляет IP- параметры из списка для заявки ATran.

SPADeleteProc ([Num List],ATran); Удаляет SP- параметры из списка для заявки ATran.

BPADeleteProc ([Num List],ATran); Удаляет BP- параметры из списка для заявки ATran. Вместо списка можно указать верхнее и нижнее значение для номеров, удаляемых значений.

RPDeleteProc(ATran); удаляет все RP - параметры данной заявки.
IPDeleteProc(ATran); удаляет все IP - параметры данной заявки.

SPDeleteProc(ATran); удаляет все SP - параметры данной заявки.

BPDeleteProc(ATran); удаляет все BP - параметры данной заявки.

PDeleteProc(ATran); удаляет все P - параметры данной заявки.

IncTranProc; Увеличивает на 1 номер последней запланированной или выпущенной заявки.

Для работы с P - параметрами, которые принадлежат системе (ячейками), нужно определить объект класса TRaram. Тогда можно использовать блоки, процедуры и функции, работающие с P – параметрами заявки, но в качестве последнего их параметра ATran – следует писать ObjName.N, где ObjName – это имя объекта. Для удобства работы, значение ObjName.N можно, и даже лучше, записать в переменную типа Integer.

Класс – TGenerate.

Блок.

*: ObjName.Generate(Mean,Dev, PR,LimCount,CanOut); Блок, задает параметры потока заявок. Здесь Mean - средний интервал появления заявки. Dev - отклонение интервала от среднего. PR- приоритет заявки. LimCont - граничное число заявок для генератора, CanOut - определяет, может ли быть выпущена заявка из генератора. Новая заявка планируется только после выхода предыдущей заявки.. По умолчанию, Dev и PR равны 0, LimCount - сохраняет заданный предел числа заявок, а CanOut - True. Используется только в ModelTxt и только один раз с данным именем.

Всякий блок Generate должен быть непременно вставлен, как объект.
При необходимости, его можно перемещать по модели, но нельзя копировать!!

*:ObjName.Pause(Flag); Блок, переводит генератор в паузу по умолчанию, а при Flag=False - генератор вновь активен.

Функции.

ObjName.C Выдает значение числа выпущенных заявок.

ObjName.Max Выдает предельное число заявок.

ObjName.Tran или ObjName.Tran Выдает номер последней запланированной заявки.

Пользование этой процедурой нужно предоставить системе.

TGenerate.Init(ObjName,' ObjName',Num,Mean,Dev,PR,LimCount); Создает объект и определяет номер блока, где находится блок Generate.

Здесь Mean - средний интервал появления первой заявки относительно начала моделирования. Dev - отклонение интервала от среднего. PR- приоритет заявки. LimCont - граничное число заявок для генератора. По умолчанию, параметры ev и PR равны 0, LimCount - не ограничивает предельное число заявок. Используется только в Initial.

Класс – TTable.

Блок.

*: ObjName.Tabulate(Value,ANum); Блок выполняет табулирование величины, Value с кратностью ANum. По умолчанию параметр Anum равен 1.

Функции.

ObjName.C Определяет число вхождений в таблицу с учетом кратности.

ObjName.A Определяет среднее значение табулируемой величины с учетом кратности.

ObjName.D Определяет среднеквадратичное отклонение табулируемой величины от средней.

ObjName.Count Число интервалов у таблицы.

ObjName.X(Num) Аргумент X для интервала Num>=0 у таблицы.

ObjName.Y(Num) Число вхождений для интервала Num>=0 у таблицы.

Процедуры управления моделью.

GetXYI(AX,AY,AI); записывает таблицу в пару динамических массивов.

ObjName. TabulateProc(Value,ANum); Выполняет табулирование величины, Value с кратностью ANum.

Show(ObjName,ANum); Выводит таблицу в виде гистограммы. В ней указываются номер графика.

GShow(ObjName,ANum); Выводит таблицу в виде графика. В остальном подобен Show.

IShow(Tab, ANum); Выводит таблицу в виде интегрального графика. В остальном подобен Show.

ObjName.ReportFile(FileName); Выводит отчет по таблице. FileName - имя файла.

ObjName.Change(Abeg,Astep,Anum); процедура аналогична init, но используется для переопределения таблицы.

Пользование этой процедурой нужно предоставить системе.

Init(ObjName,' ObjName', Abeg, Astep, ANum); Создает таблицу и определяет ее -Начало, шаг и количество интервалов. Для этого объекта отчет не создается, если текст в апострофах - пуст.

Класс – TFacility.

Блоки.

*: ObjName.Seize; Блок занятия устройства.

*: ObjName.Release; Блок освобождения устройства. Даже если у устройства есть прерванная заявка, она устройство не получит.

*: ObjName.Extract(NumBl,NumRP);Блок удаляет обслуживаемую заявку из устройства. NumBl - номер или метка блока, куда удаляется заявка. RP(NumRP) - параметра запоминает оставшееся время задержки заявки. Устройство остается незанятым, и, если были прерванные заявки, то они его не получают. Поэтому после выполнения этого блока возможно занятие устройства по Seize или Preempt. Возможно также выполнение Return0, для передачи его прерванной ранее заявке. Если последний параметр опущен, то оставшееся время задержки заявки не запоминается.

*: ObjName.ExtractInterrupt(NumBl,NumRP); Блок удаляет прерванные заявки из устройства. NumBl - номер или метка блока, куда удаляются заявки. RP(NumRP) - определяет время для завершения обслуживания заявки. Если последний параметр опущен, то оставшееся время задержки заявки не запоминается.

*: ObjName.Preempt; Блок прерывает обслуживание заявки на устройстве. Активная заявка занимает устройство по прерыванию.

*: ObjName.Preempt0; Блок прерывает обслуживание заявки на устройстве. Устройство остается незанятым. Используется для моделирования недоступности.

*: ObjName.Return; Блок освобождает устройство, занятое по прерыванию . Если есть прерванная на устройстве заявка, то она получат устройство.

*: ObjName.Return0; Блок возвращает для обслуживания в устройстве, прерванную заявку. Это происходит после того, как устройство было оставлено незанятым. Используется для моделирования возврата из недоступности.

*: ObjName.SetGoto(NumBl); Блок устанавливает номер блока, куда, возможно, будут отсылаться активные заявки. Положительное значение NumBl -определяет номер блока, отрицательное - запрещает вход в устройство. По умолчанию - восстанавливается возможность входа в устройство.

Функции.

ObjName.A Определяет коэффициент занятости устройства.

ObjName.V Определяет коэффициент временной интеграл устройства.

ObjName.T Определяет среднее время занятости устройства одной заявкой.

ObjName.L Определяет занятость устройства. 1 - устройство занято, 0 - свободно.

ObjName.C Определяет число заявок, вошедших в устройство.

ObjName.Tran или ObjName.FTran Определяет номер обслуживаемой заявки для устройства.

ObjName.InterruptTran или ObjName.PTran Определяет номер прерванной заявки для устройства.

ObjName.IsPree Определяет состояние прерывания устройства. True - если прервано, иначе False.

ObjName.Go Определяет значение параметра, установленного SetGoto.

Процедура управления моделью.

ObjName.ReportFile(FileName); Выводит отчет по устройству. FileName - имя файла.

Пользование этой процедурой нужно предоставить системе.

TFacility.Init(ObjName,' ObjName'); Создает устройство. Для этого объекта отчет не создается, если текст в апострофах - пуст.

Класс – TQueue.

Блоки.

*: ObjName.Queue (Units); Блок занятия очереди. В ней указывается число занимаемых единиц. По умолчанию 1.

*: ObjName.Depart (Units); Блок освобождения очереди. В ней указывается число освобождаемых единиц. По умолчанию 1.

Функции.

ObjName.A Определяет среднюю величину очереди.

ObjName.T Определяет среднее время в очереди для одной заявки.

ObjName.X Определяет среднее время в очереди для одной заявки. Оно определяется без учета заявок, в очереди не задержавшихся.

ObjName.C Определяет число вхождений в очередь, с учетом кратности.

ObjName.L Определяет текущую длину очереди с учетом кратности.

ObjName.M Определяет текущую максимальную длину очереди с учетом кратности.

ObjName.V Определяет временной интеграл очереди с учетом кратности.

ObjName.Z Определяет текущее число вхождений в очередь с учетом кратности. Оно определяется для заявок, в очереди не задержавшихся.

Процедура управления моделью.

ObjName.ReportFile(FileName);Создает отчет по очереди. FileName - имя файла.

Пользование этой процедурой нужно предоставить системе.

Tqueue.Init(ObjName,' ObjName'); Создает очередь. Для этого объекта отчет не создается, если текст в апострофах - пуст.

Класс – TStorage.

Блоки.

*: ObjName.Enter (Units); Блок определяет возможность входа в устройство. Units - требуемое число занимаемых единиц емкости. По умолчанию Units равно 1.

*: ObjName.Leave (Units); Блок обеспечивает выход из устройства. Units - количество освобождаемых единиц емкости. По умолчанию Units равно 1.

*: ObjName.SetStorage (Units); Блок устанавливает новую предельную емкость устройства.

*: ObjName.SetGoto (NumBl); Блок устанавливает номер блока, куда, возможно, будут отсылаться активные заявки. Положительное значение NumBl -определяет номер блока, отрицательное - запрещает вход в устройство. По умолчанию - восстанавливается возможность входа в устройство.

Функции.

ObjName.A Определяет среднее число занятых каналов.

ObjName.AM Определяет среднее предельное число занятых каналов.

ObjName.V Определяет временной интеграл.

ObjName.VM Определяет временной интеграл предельного числа занятых каналов.

ObjName.T Определяет среднее время, проведенное заявкой в устройстве.

ObjName.X Определяет среднее время, проведенное заявкой в устройстве, для заявок, в устройстве задержавшихся.

ObjName.C Определяет общее число занятий каналов.

ObjName.L Определяет текущее число занятых каналов.

ObjName.M Определяет максимальное число занятых каналов.

ObjName.Z Определяет общее число занятий каналов, при которых заявки в устройстве не задерживались.

ObjName.R Определяет текущее число свободных каналов.

ObjName.U Определяет среднюю занятость в расчете на 1 канал.

ObjName.I Выдает минимальную занятость устройства.

ObjName.Go Выдает значение, установленное по SetGoto.

Процедуры управления моделью.

ObjName.Change (Units); Процедура аналогична Init, но используется для переопределения многоканального устройства.

ObjName.ReportFile(FileName); Выводит отчет по устройству. FileName - имя файла.

Пользование этой процедурой нужно предоставить системе.

Tstorage.Init(ObjName,' ObjName',Lim); Создает устройство с заданным числом каналов. Для этого объекта отчет не создается, если текст в апострофах - пуст.

Класс – TUser.

Блоки.

*: ObjName.Link (TestGo,NumGo,Value,<Gr1);

*:ObjName.Link(Value,<Gr1); Если TestGo – истина, то активная заявка уходит на блок с меткой NumGo. Иначе блок помещает активную заявку в список. Выражение Value используется для упорядочения списка. Не обязательный целый параметр Gr1 также используется для упорядочения, причем он учитывается до Value. Это дает дополнительные возможности для формирования списка пользователя.

*: ObjName.UnLink(NumBl,Num,Log,IPCount); Блок удаляет заявки из списка. Здесь Numbl - метка, куда уйдут удаляемые заявки. Num- максимальное число удаляемых заявок. При Num=0 удаляются все заявки. При Num>0 заявки удаляются из начала списка, при Num<0 - с конца. Удалению подлежат только те заявки, для которых значение функции Log - истинно. Здесь Log - это отдельно описанная функция без параметров с результатом логического типа. Количество реально удаленных заявок, записывается в IP(IpCount). Если опущен параметр Log, то выбирать можно любые заявки. Если опущен параметр IPCount, то Количество реально удаленных заявок никуда не записывается.

*:ObjName.Select(Log ,Values) ; Блок записывает в динамический массив номера заявок из списка пользователя, для которых условие - функция Log - истинна.

*: ObjName.UnLink(NumBl,Num,Gr1,<Log,<IPCount); блок удаляет заявки из списка аналогично первой версии, но с Gr1=Gr (смотри Gr). Количество удаленных заявок сохраняется в IP(IPCount) активной заявки.

Функции.

ObjName.A Определяет среднее число заявок в списке.

ObjName.V Определяет временной интеграл списка.

ObjName.T Определяет среднее время, проведенное заявкой в списке.

ObjName.C Определяет общее число занятий списка.

ObjName.M Определяет максимальное число заявок в списке.

ObjName.L Выдает текущее число заявок в списке.

ObjName.I Выдает минимальное число заявок в списке.

ObjName.Assem Выдает номер семейства для заявки из списка.

ObjName.TranAge Выдает время жизни для заявки из списка.

ObjName.Prior Выдает значение приоритета для заявки из списка.

ObjName.Tran Выдает значение номера заявки для заявки из списка.

ObjName.ScanNum Выдает порядковый номер заявки в списке в процессе просмотра с помощью UnLink. В начале просмотра значение функции равно 0.

ObjName.Gr Определяет значение AddValue установленное для заявки из списка блоком Link.

ObjName.RP(Num) Выдает значение RP(Num) для заявки из списка.

ObjName.IP(Num) Выдает значение IP(Num) для заявки из списка.

ObjName.SP(Num) Выдает значение SP(Num) для заявки из списка.

ObjName.BP(Num) Выдает значение BP(Num) для заявки из списка.

ObjName.Val Выдает значение параметра упорядочения заявок для заявки из списка.

ObjName.Sum(Ssum,Log) Определяет значение суммы значений функции SSUM для заявок из списка.

Учитываются только заявки, для которых условие - функция LOG - истинно. Сами Ssum и Log - параметры - функции должны быть без параметров, описаны отдельно, и быть вещественного и логического типа соответственно. Если параметр Log опущен, то учитываются все заявки.

Функции типа Sum рекомендуется вычислять заранее и записывать в RP - параметры.

ObjName.Avg(Ssum,Log) функция аналогична предыдущей функции, но вычисляет среднее значение.

ObjName.Min(Ssum,Log) функция аналогична предыдущей функции, но вычисляет минимальное значение.

ObjName.Max(Ssum,Log) функция аналогична предыдущей функции, но вычисляет максимальное значение.

ObjName.Count(Log) функция аналогична предыдущей функции, но вычисляет количество заявок.

Процедура управления моделью.

ObjName.SelectProc(Log ,Values) Процедура записывает в динамический массив номера заявок из списка пользователя, для которых условие - функция Log - истинна.

ObjName.ReportFile(FileName); Выводит отчет по списку. FileName - имя файла.

Пользование этой процедурой нужно предоставить системе.

Tuser.Init(ObjName,' ObjName'); Создает список. Для этого объекта отчет не создается, если текст в апострофах - пуст.

Класс – TRGroup.

Блоки.

*: ObjName.Join (ValueList); Блок определяет вещественные числа, добавляемые к группе.

*: ObjName.Remove (Value,NumBl); Блок определяет вещественное число, удаляемое из группы. Если нужного числа в группе нет, то заявка уходит по метке NumBl. Если метка опущена или равна 0, то заявка всегда переходит в следующий блок.

*:RemoveAll; удаляет все числа из группы.

*: ObjName.Examine(Value,NumBl); Блок проверяет наличие в группе заданного числа. Если его в группе нет, то заявка уходит по указанной метке.

*:ObjName.Copy(Values); Копирует данные из группы в динамический массив.

Функции.

ObjName.A Определяет среднюю величину числовой группы.

ObjName.C Определяет число вхождений в группу.

ObjName.L Определяет текущую длину группы.

ObjName.V Определяет временной интеграл длину группы.

ObjName.M Определяет текущую максимальную длину группы.

ObjName.I Определяет текущую минимальную длину группы.

ObjName.T Определяет среднее время, проведенное элементом в группе.

ObjName.Exist(Value); Проверяет наличие в группе заданного значения

Процедуры управления моделью.

ObjName.JoinProc (ValueList); определяет вещественные числа, добавляемые к группе.

ObjName.CopyProc(Values); Копирует данные из группы в динамический массив.

ObjName.RemoveAllProc**;** - удаляет все числа.

ObjName.ReportFile(FileName); Выводит отчет по группе. FileName - имя файла.

ObjName.GroupReportFile(FileName); Создает полный отчет по членам группы. FileName - имя файла.

Пользование этой процедурой нужно предоставить системе.

TRGroup.Init(ObjName,' ObjName'); Создает числовую группу из вещественных чисел. Для этого объекта отчет не создается, если текст в апострофах - пуст.

Класс – TIGroup.

Блоки.

Это класс группы целых чисел. В нем все параметры, процедуры и функции аналогичны таким же в группе вещественных чисел.

*: ObjName.Join (Values);

*: ObjName.Remove (Value,NumBl);

*: ObjName.RemoveAll;

*: ObjName.Examine(Value,NumBl);

*: ObjName.Copy(Values);

Функции.

ObjName.A

ObjName.C

ObjName.L

ObjName.M

ObjName.V

ObjName.I

ObjName.T

ObjName.Exist(Value);

Процедуры управления моделью.

ObjName.JoinProc (Values);

ObjName.CopyProc(Values);

ObjName. RemoveAllProc;

ObjName.ReportFile(FileName);

ObjName.GroupReportFile(FileName);

Пользование этой процедурой нужно предоставить системе.

TIGroup.Init(ObjName,' ObjName'); Для этого объекта отчет не создается, если текст в апострофах - пуст.

Класс – TSGroup.

Блоки.

Это класс группы строк. В нем все параметры, процедуры и функции аналогичны таким же в группе вещественных чисел.

*: ObjName.Join (Values);
*: ObjName.Remove (Value,NumBl;
*: ObjName.RemoveAll;
*: ObjName.Examine(Value,NumBl);
*: ObjName.Copy(Values);

Функции.

ObjName.A
ObjName.C
ObjName.L
ObjName.M
ObjName.V
ObjName.I
ObjName.T
ObjName.Exist(Value);

Процедуры управления моделью.

ObjName.JoinProc (Values);
ObjName.CopyProc(Values);
ObjName.RemoveAllProc;
ObjName.ReportFile(FileName);
ObjName.GroupReportFile(FileName);

Пользование этой процедурой нужно предоставить системе.

TIGroup.Init(ObjName,' ObjName'); Для этого объекта отчет не создается, если текст в апострофах - пуст.

Класс – TTGroup.

Блоки.

*: ObjName.Join(ATran); Блок - добавляет заданную заяку в группу, по умолчанию, он добавляет активную заявку.

*: ObjName.Remove (Num,Del,NumIP); Блок удаляет до Num заявок из группы. Если 0, то потенциально могут быть удалены все заявки. Del - отдельно описанная логическая функция без параметров, которая определяет заявки, подлежащие удалению. В IP(NumIP) запишется число реально удаленных заявок. Могут быть опущены, в том числе и все параметры. Тогда подразумевается удаление одной любой заявки. Причем тогда количество реально удаленных заявок никуда не записывается.

*:ObjName.Remove(Axn1); Блок удаляет заявку с номером Axn1. По умолчанию -удаляется из группы активная заявка.

*:ObjName.RemoveAll - удаляет все заявки.

*: ObjName.Examine(NumBl,ATran); Блок проверяет наличие заданной заявки в группе. По умолчанию, проверяется наличие активной заявки. Если ее там нет, то активная заявка переходит на указанный блок.

*: ObjName.Scan (YesF,ValueF,NumRP,NumBl); Блок отыскивает заявки, удовлетворяющие отдельно описанной логической функции Yesf. Для первой попавшейся такой заявки, вычисляется значение вещественной функции ValueF. Оно записывается в RP(NumRP) параметр активной заявки. Если нужной заявки не найдется, то заявка уходит по метке Numbl, если этот параметр есть.

*: ObjName.Scan (YesF,ValueF,NumIP,NumBl); Блок аналогичен предыдущему, но вычисляет значение целой функции. Запись происходит в IP параметр.

*: ObjName.Scan (YesF,ValueF,NumSP,NumBl); Блок аналогичен предыдущему, но вычисляет значение строковой функции. Запись происходит в SP параметр.

*: ObjName.Scan (YesF,ValueF,NumBP,NumBl); Блок аналогичен предыдущему, но вычисляет значение логической функции. Запись происходит в BP параметр.

*: ObjName.Alter (YesF,ValueF,NumRP,Num,NumIPCount); Блок - вычисляется логическая функция Yesf для определения подходящих заявок. Для первых попавшихся подходящих заявок вычисляется значение вещественной функции ValueF. Оно записывается в RP(NumRP). При NumRP=-1 - изменение времени рождения на текущий момент времени. Всего модифицируется не более Num заявок из группы, или все. Реальное число модифицированных заявок записывается в IP(NumIPCount). Два последних параметра могут быть опущены. Тогда модифицируются все заявки (значение Num=0).

*: ObjName.Alter (YesF,ValueF,NumIP,Num,NumIPCount); Блок аналогичен предыдущему, но вычисляется значение целой функции. Оно записывается в IP(NumIP) При NumIP=-1 - изменение приоритета.

*: ObjName.Alter (YesF,ValueF,NumSP,Num,NumIPCount); Блок аналогичен предыдущему, но вычисляется значение строковой функции. Оно записывается в SP(NumSP).

*: ObjName.Alter (YesF,ValueF,NumBP,Num,NumIPCount); Блок аналогичен предыдущему, но вычисляется значение логической функции. Оно записывается в BP(NumBP).

*:ObjName.Select(Log ,ANum); Блок записывает в динамический массив номера заявок из группы, для которых условие - функция Log - истинна.

Функции.

ObjName.C Определяет число вхождений в группу.
ObjName.L Определяет текущую длину группы.
ObjName.A Определяет среднее число заявок в группе.
ObjName.V Определяет временной интеграл.
ObjName.T Определяет среднее время, проведенное заявкой в группе.
ObjName.M Определяет максимальное число заявок в группе.
ObjName.I Выдает минимальное число заявок в группе.
ObjName.Assem Определяет семейство заявки из группы.
ObjName.TranAge Определяет время жизни заявки из группы.
ObjName.Prior Определяет приоритет для заявки из группы.
ObjName.RP(Num) Определяет значение RP(Num) для заявки из группы.
ObjName.IP(Num) Определяет значение IP(Num) для заявки из группы.

ObjName.BP(Num) Определяет значение BP(Num) для заявки из группы.

ObjName.SP(Num) Определяет значение SP(Num) для заявки из группы.

ObjName.Tran Определяет номер заявки для заявки из группы.

ObjName.ScanNum Выдает порядковый номер заявки в списке в процессе просмотра с помощью процедур группы Scan и Alter. В начале просмотра значение функции равно 0.

ObjName.Sum(Ssum,Log) Определяет значение суммы значений функции Ssum для заявок из группы. Учитываются только заявки, удовлетворяющие условию - функции Log. Последний параметр может быть опущен. Тогда учитываются все заявки. Функции группы Sum, рекомендуется вычислять заранее и записывать в RP - параметр.

ObjName.Avg(Ssum,Log) функция аналогична предыдущей функции, но вычисляет среднее значение.

ObjName.Min(Ssum,Log) функция аналогична предыдущей функции, но вычисляет минимальное значение.

ObjName.Max(Ssum,Log) функция аналогична предыдущей функции, но вычисляет максимальное значение.

ObjName.Count(Log) функция аналогична предыдущей функции, но вычисляет количество заявок.

ObjName.Exist(Value); Проверяет наличие в группе заданного числа. По умолчанию - активной заявки.

Процедуры управления моделью.

ObjName.SelectProc(Log ,ATran); записывает в динамический массив номера заявок из группы, для которых условие - функция Log - истинна.

ObjName.RemoveProc(Num,Del,NumIP); Удаляет до Num заявок из группы. Если 0, то потенциально могут быть удалены все заявки. Del - отдельно описанная логическая функция без параметров, которая определяет заявки, подлежащие удалению. В IP(NumIP) запишется число реально удаленных заявок.

ObjName.RemoveProc(Axn1); Удаляет заявку с номером Axn1. По умолчанию - удаляется из группы активная заявка.

ObjName.RemoveAllProc - удаляет все заявки.

ObjName.ReportFile(FileName); Выводит отчет по группе. FileName - имя файла.

ObjName.GroupReportFile(FileName); Создает полный отчет по заявкам группы. FileName - имя файла.

Пользование этой процедурой нужно предоставить системе.

TTGroup.Init(ObjName,' ObjName'); Создает группу заявок. Для этого объекта отчет не создается, если текст в апострофах - пуст.

Общие функции и процедуры.

Uniform(VMin,VMax) Выдает вещественную случайную величину, распределенную по равномерному закону в интервале от VMin до VMax.

DUniform(VMin,VMax) Выдает целую случайную величину, распределенную по равномерному закону в интервале от VMin до Vmax включительно.

Exponential(Mean) Выдает случайную величину, распределенную по экспоненциальному закону со средним значением mean.

Normal(Mean,Dev) Выдает случайную величину, распределенную по нормальному закону. Ее среднее значение mean и среднеквадратичное отклонение dev.

Binomial(TrialCount,Probabilityt) Выдает случайную величину целого типа, распределенную по биномиальному закону. Ее степень – TrialCount, а Probabilityt – вероятность положительного исхода при испытании.

Geometric(Probabilityt) Выдает случайную величину целого типа, распределенную по геометрическому закону. Probabilityt – вероятность положительного исхода при испытании.

Negbinom(SuccessCount,Probabilityt) Выдает случайную величину целого типа, распределенную по негативному биномиальному закону. Ее счетчик положительных исходов - SuccessCount, а Probabilityt – вероятность положительного исхода при испытании.

Triangular(Min,Max,Mode) Выдает случайную величину, распределенную по треугольному закону. Min,Max,Mode – это минимальное, максимальное, и максимально-вероятное значение в треугольном распределении.

RByBool(Log,Values) Выдает значение вещественного выражения из списка Values, для первого истинного выражения из списка Log. Элементы каждого списка записываются в квадратных скобках через запятую.

IByBool(Log,Values) Выдает значение целого выражения из списка Values, для первого истинного выражения из списка Log. Элементы каждого списка записываются в квадратных скобках через запятую.

SByBool(Log, Values) Выдает значение строкового выражения из списка Values, для первого истинного выражения из списка Log. Элементы каждого списка записываются в квадратных скобках через запятую.

RByNum(Num,Values) Выдает значение вещественного выражения из списка Values, по номеру Num>=1.

IByNum(Num,Values) Выдает значение целого выражения из списка Values, по номеру Num>=1.

SByNum(Num,Values) Выдает значение строкового выражения из списка Values, по номеру Num>=1.

BByNum(Num,Values) Выдает значение логического выражения из списка Values, по номеру Num>=1.

CFun(X, Args,Values) Вычисляет числовую функцию как непрерывную. Функция задана списками координат.

DFun(X, Args,Values) Вычисляет функцию как дискретную. Функция задана списками координат.

ICFun(X, Args,Values); Вычисляет числовую функцию как непрерывную, но округленную до целого. Функция задана списками координат.

IDFun(X, Args,Values); Вычисляет числовую функцию как дискретную, но округленную до целого. Функция задана списками координат.

Any Всегда истинная логическая функция.

GetFile(FileName) Выдает содержимое файла в виде строки.

RMaxNum(Values) Выдает номер вещественного выражения из списка в квадратных скобках, у которого значение – максимально. Значение функции всегда >=0.

IMaxNum(Values) Выдает номер целого выражения из списка в квадратных скобках, у которого значение – максимально. Значение функции всегда >=0.

SMaxNum(Values) Выдает номер строкового выражения из списка в квадратных скобках, у которого значение – максимально. Значение функции всегда >=0.

BMaxNum(Values) Выдает номер логического выражения из списка в квадратных скобках, у которого значение – максимально. Значение функции всегда >=0.

RMinNum(Values) Выдает номер вещественного выражения из списка в квадратных скобках, у которого значение – минимально. Значение функции всегда >=0.

IMinNum(Values) Выдает номер целого выражения из списка в квадратных скобках, у которого значение – минимально. Значение функции всегда >=0.

SMinNum(Values) Выдает номер строкового выражения из списка в квадратных скобках, у которого значение – минимально. Значение функции всегда >=0.

BMinNum(Values) Выдает номер логического выражения из списка в квадратных скобках, у которого значение – минимально. Значение функции всегда >=0.

RMaxVal(Values) Выдает значение вещественного выражения из списка в квадратных скобках, у которого значение – максимально.

IMaxVal(Values) Выдает значение целого выражения из списка в квадратных скобках, у которого значение – максимально.

SMaxVal(Values) Выдает значение строкового выражения из списка в квадратных скобках, у которого значение – максимально.

BMaxVal(Values) Выдает значение логического выражения из списка в квадратных скобках, у которого значение – максимально.

RMinVal(Values) Выдает значение вещественного выражения из списка в квадратных скобках, у которого значение – минимально.

IMinVal(Values) Выдает значение целого выражения из списка в квадратных скобках, у которого значение – минимально.

SMinVal(Values) Выдает значение строкового выражения из списка в квадратных скобках, у которого значение – минимально.

BMinVal(Values) Выдает значение логического выражения из списка в квадратных скобках, у которого значение – минимально.

BoolToInt(Log) - представляет логическое значение - как целое.

Специальные процедуры управления моделью.

InitRand(Value); Устанавливает начальное целое значение для генератора случайных чисел.

ReportVal(Varia,Name); Выводит выражение вещественного, целого, строкового, или логического типа в отчет, Name - заголовок, по умолчанию он пустой.

Если вы хотите выводить данные в отчет не с помощью ReporVal, а с помощью операторов Write(FileVar, Value_List); Writeln(FileVar, Value_List); то вместо FileVar следует указывать имя файла вывода для отчета, а именно, FileRep.

ReportProc; Выводит отчет в целом.

ReportBlocks(FileName); Выводит отчет о блоках в указанный файл.

ReportCurrentList(FileName); Выводит отчет о списке текущих событий в указанный файл.

ReportFutureList(FileName); Выводит отчет о списке будущих событий в указанный файл.

ReportSynchronizeList(FileName); Выводит отчет о списке синхронизации в указанный файл.

ReportInterruptList(FileName); Выводит отчет о списке прерываний в указанный файл.

ReportUserList(FileName); Выводит отчет о списках пользователя в указанный файл.

ReportRPList(FileName); Выводит отчет о RP - параметрах в указанный файл.
ReportIPList(FileName); Выводит отчет о IP - параметрах в указанный файл.
ReportSPList(FileName); Выводит отчет о SP - параметрах в указанный файл.
ReportBPList(FileName); Выводит отчет о BP - параметрах в указанный файл.
Start(Tg); Задает начальное значение параметру TG1 и запускает модель.
AReset; Выполняет сброс модели.
ClearModel(Clear); Очищает модель от всех данных при Clear =True . При Clear =False - в модели сохраняются без изменений значения всех переменных и всех X - параметров. AClear – то же самое, что и ClearModel(True)

Динамические массивы.

Для облегчения работы с массивами предусмотрены следующие типы динамических массивов.

ADouble=array of Double;
AIntger=array of Integer;
AString=array of String;
ABoolean=array of Boolean;
Переменные для массивов должны быть описаны по образцу
Name: ADouble;
Для дозаписи в них списка значений можно использовать блоки.
*: RALet(Name, [список] ,Num);
*: IALet(Name, [список] ,Num);
*: SALet(Name, [список] ,Num);
*: BALet(Name, [список] ,Num);

Здесь, Num - позиция, начиная с которой динамический массив Name пополняется списком. По умолчанию, содержимое динамического массива замещается списком. Сами массивы можно указывать вместо списков, но тогда без квадратных скобок.

*:RAAdd(Name,[список]);
*:IAAdd(Name,[список]);
*:SAAdd(Name,[список]);
*:BAAdd(Name,[список]);
Дописывают список в конец динамического массива.
Для записи в массивы списка значений можно использовать процедуры.
RALetProc(Name,[список],Num);
IALetProc(Name,[список],Num);
SALetProc(Name,[список],Num);
BALetProc(Name,[список],Num);
Они аналогичны блокам группы ALet. А также
RAAddProcNext(Name,[список]);
IAAddProc(Name,[список]);

SAAddProc(Name,[список]);
BAAddProc(Name,[список]);
Они аналогичны блокам группы AAdd.
Номера элементов динамических массивов начинаются с 0.
Для вывода отчетов о массивах - можно использовать процедуры. Здесь вместо списка можно указывать динамические массивы.
RAReportVal([список], заголовок)
IAReportVal([список], заголовок)
SAReportVal([список], заголовок)
BAReportVal([список], заголовок)
Функция GetSortNum(AVal,Val) -в динамическом массиве AVal, отыскивается номер, в котором расположено первое попавшееся подходящее значение, большее или равное Val. Массив непременно должен быть упорядоченным и должен быть одного из следующих типов: ADouble, AInteger, AString, ABoolean.
Функция ASub(AVal, NumB,Count) - в динамическом массиве AVal, выделяется под массив начиная с NumB, длиной в Count элементов. Массив непременно должен быть одного из следующих типов: ADouble, AInteger, AString, ABoolean.
Для описания динамических массивов основных объектов используются следующие типы.
ATparam=Array of Tparam;
ATFacility=Array of TFacility;
ATQueue=Array of TQueue;
ATStorage=Array of TStorage;
ATTable=Array of TTable;
ATUser=Array of TUser;
TRGroup=Array of TRGroup;
ATIGroup=Array of TIGroup;
ATSGroup=Array of TSGroup;
ATTGroup=Array of TTGroup);
Набор процедур для этих массивов очень ограничен и состоит из следующих процедур для присваивания массивам набора значений, то есть объектов. Тогда к объектам можно будет обращаться как по именам, так и по их номерам в массиве. В процедурах – Name – это имя массива.
ParamSet(Name, [список]);
FacilitySet(Name, [список]);
QueueSet(Name, [список]);
StorageSet(Name, [список]);
TableSet(Name, [список]);
UserSet(Name, [список]);
RGroupSet(Name, [список]);
IGroupSet(Name, [список]);
SGroupSet(Name, [список]);
BGroupSet(Name, [список]);
TGroupSet(Name, [список]);
В процедуре CloseAllObj все динамические массивы должны быть освобождены, то есть для каждого массива следует выполнить оператор

Name:=Nil;

Для описания динамических массивов процедур и функций без параметров используются следующие типы.

ATProcedure=Array of TProcedure;
ATRFunction=Array of TRFunction;
ATIFunction=Array of TIFunction;
ATSFunction=Array of TSFunction;
ATBFunction=Array of TBFunction;

Набор процедур для этих массивов аналогичен набору процедур для основных объектов и состоит из следующих процедур для присваивания массивам набора значений, то есть процедур и функций. Тогда к процедурам или функциям можно будет обращаться как по именам, так и по их номерам в массиве. В процедурах – Name – это имя массива.

ProcedureSet(Name, [список]);
RFunctionSet(Name, [список]);
IFunctionSet(Name, [список]);
SFunctionSet(Name, [список]);
BFunctionSet(Name, [список]);

В процедуре CloseAllObj все динамические массивы должны быть освобождены, то есть для каждого массива следует выполнить оператор

Name:=Nil;

Процедуры и функции, реализованные в модуле AddModelUnit.

При работе с графическими изображениями можно передвигать их при нажатой правой кнопке. При выделении прямоугольника левой кнопкой, прямоугольник, выделенный вправо и вниз, увеличивает масштаб выделенной части, прямоугольник, выделенный любым другим способом – возвращает изображение в исходное состояние. Эти правила действуют на вкладках Lines и Bars.

Блоки.

*: ToPoint(Num,X,Y); Блок выполняет вывод на график следующей точки. Num>=0 - номер серии, X,Y - координаты.

*: ToString(Num,Value); Блок выполняет вывод на AddMemo строкового, вещественного целого или логического параметра . Num>=0 - номер строки, Value - параметр.

*: ToValue(Num,Value); Блок выполняет вывод значения Value для строки Num>=0 в виде величины соответствующего столбика.

*: LineClear (Num); Блок сброса графика с данным номером Num>=0, по умолчанию сбрасываются все графики.

*: BarClear (Num); Блок сброса гистограмм с данным номером Num>=0, по умолчанию сбрасываются все гистограммы.

*: CurrentBlocks; Блок выполняет вывод на AddMemo статистики по блокам.

*: Stop; блок, обеспечивающий почти немедленное прекращение моделирования. Точнее, моделирование прекращается, как только заявка, прошедшая через него, прекращает движение. Блок полезен при отладке и в некоторых других случаях. Содержимое счетчика завершений не становится равным нулю.

*:Message('Сообщение'); выводит сообщение в окне сообщений. Если текста нет, то сообщение закрывается.

Функции.

GetTerm Выдает значение для TG1 из формы.

RInput(Line,Def) Вызывает диалог для ввода вещественного значения. Line - строка для заголовка. Def- начальное значение, по умолчанию -0.

IInput(Line,Def) Вызывает диалог для ввода целого значения. Line - строка для заголовка. Def- начальное значение, по умолчанию -0.

SInput(Line,Def) Вызывает диалог для ввода строкового значения. Line - строка для заголовка. Def- начальное значение, по умолчанию – пустая строка.

BInput(Line,Def) Вызывает диалог для ввода логического значения. Line - строка для заголовка. Def- начальное значение, по умолчанию -False.

InputItem(Line,Def) Вызывает диалог для выбора строки из выпадающего списка, Line – заголовок диалога. Def- начальное значение номера строки, по умолчанию -0.

RGet(NumStr) Ввод вещественного значения из строки Numstr>=0 в AddMemo.

IGet(NumStr) Ввод целого значения из строки Numstr>=0 в AddMemo.

SGet(NumStr) Ввод строкового значения из строки Numstr>=0 в AddMemo.

BGet(NumStr) Ввод логического значения из строки Numstr>=0 в AddMemo.

RGB(Red,Green,Blue) Формирует цвет по интенсивностям трех основных цветов. Каждый цвет может иметь значение от 0 до 255.

FileNameGet. Вызывает диалог открытия файла и возвращает его имя как строку.

FileNamePut. Вызывает диалог закрытия файла и возвращает его имя как строку.

GetColor. Позволяет выбрать цвет из диалога.

Процедуры управления моделью.

CShow(X,Y, Num); Выводит график непрерывной функции. В ней указываются координаты точек, номер графика и масштабы величин по X и по Y. По умолчанию, номер графика – 0, а масштабы равны 1.

DShow(X,Y, Num); Выводит график дискретной функции. В ней указываются координаты точек, номер графика и масштабы величин по X и по Y. По умолчанию, номер графика – 0, а масштабы равны 1.

NewItems(lines); Заполняет строками выпадающий список для InputItem. Список строк записывается в квадратных скобках через запятую.

AddMemo(Line); Добавляет вещественное, целое, строковое или логическое значение к AddMemo.

AddMemoReset; Очистка AddMemo.

RepMemoReset; Очистка AddMemo.

AddLoad (FileName); Загружает файл в AddMemo.

AddSave (FileName); Сохраняет файл из AddMemo.

Line3D(Yes3D); Переключает, плоские - объемные графики.

Bar3D(Yes3D); Переключает, плоские - объемные гистограммы.

LineSave (FileName); Сохраняет графики в файле на диске.

BarSave (FileName); Сохраняет гистограммы в файле на диске.

ReportSave (FileName); Сохраняет отчет по модели в файле.

LineColor(Num,Col); Задает цвет графика с номером Num>=0 .

LineTitle(Num,Line); Задает имя графика.

ValueColor(Num,Col); Задает цвет столбика с номером Num>=0.

ValueTitle(Num,Line); Задает имя столбика с номером Num>=0.

BarColor(Num,Col); Задает цвет гистограммы с номером Num>=0.

BarTitle(Num,Line); Задает имя гистограммы с номером Num>=0.

GetLine (Num,AX,AY); линия с номером Num копируется в динамические массивы.

GetBar (Num,AX,AY); гистограмма с номером Num копируется в динамические массивы.

SetStart(Tg1); Устанавливает значения счетчика завершений на главной форме.

ToStringProc(Num,Value); На AddMemo выполняет вывод вещественного, целого, строкового или логического значения Value. Номер строки - Num>=0.

ToValueProc(Num,Value); Выполняет вывод значения Value для строки Num>=0 в виде величины соответствующего столбика.LineShow (HideShow,Num); Показать-скрыть (HideShow) график Num>=0. По умолчанию, все графики.

LineShow(HideShow,Num); Показать- скрыть (HideShow) график Num>=0. По умолчанию, все графики.

BarShow (HideShow,Num); Показать- скрыть (HideShow) гистограмму Num>=0. По умолчанию, все гистограммы.

RepMemoAdd (Line); Добавляет значение вещественного, целого строкового или логического выражения к RepMEMO.

Bar(X,Y, Num); создает из динамических массивов X,Y - гистограмму, Num>=0 - серия.

ToPointProc(Num,X,Y); Добавляет следующую точку X,Y к графику, Num>=0 - серия.

LineClearProc(Num); Сброс заданной серии для графиков, по умолчанию, сбрасываются все серии.

BarClearProc(Num); Сброс заданной серии для гистограмм, по умолчанию, сбрасываются все серии.

procedure ValueClearProc; Сброс серии значений.

CustomProc(st,apr); Создает перечень заголовков процедур пользователя и перечень самих процедур. Перечни записываются в квадратных скобках через запятую.

SetPage(Page); Переключает приложение на страницу.

ShowPage(Page,flag); Показывает или скрывает страницу.

PageColor(Page,Col); Задает цвет страницы.

Список констант для SetPage, PageColor и ShowPage.

spSimulation, ReportSetting, spReport, spLine, spValues, spBar, spAbout, spReserve ,spLicence

Для работы в пакетном режиме, имеются следующие процедуры.

FormShow(log) Показывает или скрывает форму.

Quit Закрывает приложение.

RunBat Выполняет моделирование в пакетном режиме. На амом деле при этом выполняются FormShow(False); Simulation; ReportProc; Quit ;

SetReportFlag(Flag, val); Соответствующий флаг отчета устанавливается в True или False, по умолчанию – в истину.

Константы для управления настройкой отчета из модели.

rfModel, rfInterruptList, rfUserList, rfPList, rfXList, rfCurrentList,
rfFutureList, rfSynchronizeList, rfGroupList, rfBlocks, rfQueue,rfStorage,

rfUser, rfFacility, ;rfTable, rfVal, rfGroup

MessageProc('Сообщение'); выводит сообщение в окне сообщений. Если текста нет, то сообщение закрывается.

Список предопределенных констант для задания цвета.

ClAqua – бирюзовый
clBlack - черный
clBlue - синий
clCream - кремовый
clDkGray Темно-серый
clFuchsia фуксиновый
clGray – темно серый
clGreen - зеленый
clLime – лимонно зеленый
clLtGray - светло-серый
clMaroon марон
clMedGray средне серый
clMoneyGreen цвет доллара
clNavy морской синий
clOlive оливковый
clPurple фиолетовый
clRed красный
clSilver серебряный
clSkyBlue голубой
clTeal темно бирюзовый
clWhite белый
clYellow желтый

Приложение 2. Стандартная выходная статистика.

На вкладке Report содержится стандартная статистику об объектах Object GPSS, используемых в данной модели (FACILITY, QUEUE, STORAGE и т.д.). Начинается отчет с конвертированного текста самой модели, который мы уже неоднократно наблюдали, например:

```
{/G} {::G_}   1:G.generate(exponential(100));
     2:IPlet(1,[duniform(3,17)]) ;
     3: split(Ip(1)-1)  ;
     4: advance(300,290) ;
     5: assemble(Ip(1)) ;
     6: tostring(IX(1),' ac1 '+Floattostr(ac1)+' M1 '+Floattostr(m1)
         +' P= '+inttostr(Ip(1))+' Assem '+inttostr(assem)) ;
     7: IXLet(1,IX(1)+1) ;
     8: terminate ;
{/G0} {::G0_}   9:G0.generate(2000);
     10: terminate(1) ;
```

Далее идет строка, сообщающая о начальном и конечном времени моделирования, которая выглядит, например, следующим образом.

```
StartTime         0.00000 EndTime       2000.00000
```

Затем идет информация о блоках, в которой первая колонка определяет номер блока, вторая – число вхождений в блок, а третья – текущее число заявок в блоке. Это выглядит, например, следующим образом.

При истолковании статистики по блокам следует учитывать, что заявка может находиться в блоке, или оказаться задержанной на его выходе, из-за невозможности войти в следующий блок. Колонка **Current** содержит суммарное число заявок, как внутри блока, так и на его выходе. Для различения этих ситуаций можно включать в текст модели блоки *:Nop. Этот блок не выполняет никаких действий, но и не препятствует входу в него заявки.

Также рекомендуется помещать блок *:Priority(-1) непосредственно перед *:Terminate, тогда статистика будет соответствовать ситуации в модели, когда уже завершена обработка событий текущего момента. Впрочем, для большинства задач такие подробности не имеют значения.

BLOCK Report

Location	Entries	Current
1	16	0
2	16	0
3	16	0

Далее в модели идет информация о списке текущих событий, списке будущих событий, списке синхронизации, списке прерываний, списке пользователя, списках P – параметров. А затем, в произвольном порядке, точнее в порядке описания, идет информация об объектах следующих типов Tparam, TFacility, TQueue, TStorage, TTable, TUser, TRGroup, TIGroup, TSGroup, TTGroup.

Если какой либо объект отсутствует, или список пуст, то соответствующий раздел отчета – отсутствует. Можно отменить вывод любого из перечисленных разделов отчета.

Список текущих событий выводится в следующей форме.

Report CURRENT LIST

Tran	Assem	Prior	Current	Next	TimeStamp	TimeEnter	Interrupt
918	918	0	2	3	98862.81090	98862.81090	0
919	919	0	2	3	98876.88954	98876.88954	0

Здесь `TRAN` - номер заявки, `Assem` – номер семейства, `Pr` – приоритет, `Current` – блок, где заявка находится в данный момент, `Next` – блок, куда заявка планирует перейти, `TimeStamp` – время рождения заявки, `TimeEnter` – время поступления заявки в список, `Interrupt` – состояние прерывания, 0- заявка не прервана, 1- прервана.

Список будущих событий выводится в следующей форме.

Report FUTURE LIST

Tran	Assem	Prior	Current	Next	TimeStamp	EndTime	Interrupt
917	917	0	6	7	99850.25705	100005.32145	0
932	932	0	0	1	100017.33636	100017.33636	0

В этом списке назначение колонок почти такое же, как и для списка текущих событий, а колонка `EndTime` – определяет момент начала движения для заявки.

Список синхронизации выводится в следующей форме.

Report SINCHRONIZE LIST

Tran	Assem	Prior	Current	Next	TimeStamp	TimeEnter	Num	Interrupt
3498	1	0	4	5	239841.48540	239841.48540	0	0
3499	1	0	4	5	239945.41928	239945.41928	0	0

В этом списке назначение колонок почти такое же, как и для списка текущих событий, а `Num` – количество заявок, собранных для семейства.

Список прерываний выводится в следующей форме.

Report Interrupt LIST

```
Tran  NumFacil      IntTran
991         5       991
992         5       991
```

В этом списке TRAN - номер заявки, NumFacil - номер объекта устройство, TRANP - номер прерванной заявки.

Список пользователя выводится в следующей форме.

```
  Report USER LIST
Tran     Assem        Prior   Current      Next       TimeStamp       Value     NumList
Interrupt
131       131         0         3         0     12270.19646       251.30371         5
0
```

В этом списке назначение колонок почти такое же, как и для списка текущих событий, а Value – параметр упорядочения, NumList- номер объекта список пользователя.

Списки P – параметров выводится в следующей форме.

```
          RP LIST ( Double )
  Tran       Num          Double
 1001         1     99305.13187
 1001         2     99622.20574
```

В этом списке вначале выводится заголовок, определяющий вид списка, а затем сам список в три колонки, первая – номер заявки, вторая – номер параметра, а третья – его значение.

Списки X – параметров как для объекта SYS, так и для любого другого выводится в следующей форме.

```
SYS          Report RP LIST ( DOUBLE ) Count= 5
       Num          double
         1  16691407.28571
         2  11050932.31342
```

В этом списке вначале выводится заголовок, определяющий вид списка и число элементов в нем, а затем сам список в две колонки, первая – номер параметра, а вторая – его значение.

Информация об устройствах имеет вид.

```
  Facility   Entries   Current         Utility        AverTime    NextGo       Tran      IntTran
NumObj
        fu       916         1         0.91878       100.30302         0       917      0
4
```

Здесь Facility - имя устройства, Entries - количество заявок, вошедших в устройство, Current - активная занятость устройства, Utility - коэффициент занятости, AverTime - среднее время обслуживания заявки, NextGo - номер блока для возможного ухода заявки, TRAN - номер обслуживаемой заявки, TRANP - номер прерванной заявки, NumObj - номер объекта.

Информация об очередях имеет вид.

```
    Queue   Entries   Current       Max      Zero       AverQueue        AverTime   AverTime(-
Ze)    NumObj
       q2       930        14        28        76         8.53572       917.81894
999.49838         3
```

Здесь Queue - имя очереди, Entries - количество заявок, вошедших в очередь с учетом кратности, Current - активная длина очереди, Max - максимальная длина очереди, Zero - количество заявок в очереди не задержавшихся, AverQueue - средняя длина очереди, AverTime - среднее время, проведенное заявкой в очереди, AverTime(-Ze) - среднее время, проведенное заявкой в очереди, без учета в очереди не задержавшихся, NumObj - номер объекта.

Информация о многоканальных устройствах имеет вид.

```
   Storage   Entries   Current       Max       Min        AverStor        AverTime
Utility
```

```
      Nac      1205         5         5         2         4.71657       388.12223
0.94331
    NextGo       TRAN             AC1    ACapac  Capacity    NumObj
         0      1105     99086.19113     5.000         5         4
```

Здесь `Storage` - имя устройства, `Entries` - количество заявок, вошедших в устройство с учетом кратности, `Current` - активная занятость, `Max` - максимальная занятость, `Min` - минимальная занятость, `AverStor` - среднее число занятых каналов, `AverTime` - среднее время, проведенное заявкой в устройстве, `Utility` - коэффициент занятости, `NextGo` - номер блока для возможного ухода заявки, `TRAN` - - номер последней поступившей заявки, `AC1` - время поступления последней заявки, `ACapac` - средняя емкость, `Capacity` - активная емкость, `NumObj` - номер объекта.

Информация о таблице имеет вид.

```
Table       Tt1  Entries      1000.00000
 Mean       144.16466 StdDev        57.45456    NumObj        11
           Range       Frequency
         50.00000         0.00000
         60.00000        58.00000
```

Здесь в заголовке последовательно указаны имя таблицы, количество вхождений в нее с учетом кратности, среднее значение по таблице, среднеквадратичное отклонение, номер объекта. А далее, в два столбца идет правая граница интервала, и количество вхождений в интервал.

Информация о списке пользователя имеет вид.

```
    User   Entries   Current       Max       Min        AverUser        AverTime    NumObj
Flag
       UU       108         4        19         0         6.68740       708.07642         5
1
```

Здесь `User` - имя списка, `Entries` - количество заявок, вошедших в список, `Current` - текущее число заявок в списке, `Max` - максимальное число заявок в списке, `Min` - минимальное число заявок в списке, `AverUser` - среднее число заявок в списке, `AverTime` - среднее время, проведенное заявкой в списке, `NumObj` - номер объекта, `Flag` - флаг занятости списка.

Информация о группе целых, вещественных, и строк имеет вид.

```
   GroupInt   Entries   Current       Max       AverGroup    NumObj
       gr1       151        77        78        77.44262        12
gr1          Report IGROUP LIST ( INTEGER ) num=77
             11
             13
```

Здесь первая колонка - имя группы, `Entries` - количество заявок, вошедших в группу, `Current` - текущее количество элементов группы, `Max` - максимальное количество элементов группы, `AverGroup` - среднее количество элементов группы, `NumObj` - номер объекта. А дальше следует колонка значений элементов группы.

Информация о группе заявок имеет вид.

```
GroupTran   Entries   Current    NumObj
      grou       758         8         5
grou          Report TGROUP LIST ( TRANSACTIONS ) num=8
           2080
           2082
           2085
```

Здесь первая колонка - имя группы, `Entries` - количество заявок, вошедших в группу, `Current` - текущее количество заявок в группе, `NumObj` - номер объекта. А дальше следует колонка номеров заявок из группы.

Перечень ссылок

1. Основы теории вычислительных систем, М. Высшая школа, 1978.
2. Бусленко Н.П. Моделирование сложных систем, М. Наука, 1978.
3. Советов Б.Я. Моделирование систем, М. Высшая школа, 1985.
4. Советов Б.Я. Моделирование систем (курсовое проектирование), М. Высшая школа,1986.
5. Советов Б.Я. Моделирование систем (лабораторный практикум), М. Высшая школа,1986.
6. Молчанов А.А. Моделирование и проектирование сложных систем, К. Высшая школа,1988.
7. Альянах И.Н. Моделирование вычислительных систем, Л. Машиностроение, 1988
8. Серия разработка САПР Имитационное моделирование В.М. Черненький, N9, М. Высшая школа,1990.
9. Дал У., Мюрхауг Б., Нюгорд К. Универсальный язык моделирования. - М.: Мир, 1969.- 316 с.
10. Емельянов А.А., Власова Е.А. Имитационное моделирование в экономических информационных системах. - М.: МЭСИ, 1996.- 108 с.
11. Клейнен Дж. Статистические методы в имитационном моделировании. -М.: Статистика, 1978.-221с.
12. Машинные имитационные эксперименты с моделями экономических систем./ Под ред. Нейлора Т.М. - М.: Мир, 1975.-501с.
13. Шеннон Р. Дж. Имитационное моделирование систем - искусство и наука.- М.:Мир,1978.-418с.
14. Шрайбер Т. Дж. Моделирование на GPSS.- М.: Машиностроение, 1980.-592с.