РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ
Факультет физико-математических и естественных наук

А. В. Королькова, Д. С. Кулябов

Моделирование информационных процессов

Учебное пособие

Москва
Российский университет дружбы народов
2014

УДК 519.6:004(076.5)
ББК 22.18+32.973-018я73
К68

Утверждено
РИС Учёного совета
Российского университета
дружбы народов

Рецензенты:
старший научный сотрудник Института проблем информатики РАН
кандидат физико-математических наук *Р. В. Разумчик*;
начальник сектора телекоммуникаций УИТО РУДН
кандидат физико-математических наук, доцент *К. П. Ловецкий*

К68 Королькова, А. В.
Моделирование информационных процессов : учебное пособие / А. В. Королькова, Д. С. Кулябов. — М. : РУДН, 2014. — 192 с. : ил.

ISBN 978-5-209-05772-7

Пособие представляет собой лабораторный практикум по дисциплине «Моделирование информационных процессов» и предназначено для студентов направлений «Математика и компьютерные науки», «Фундаментальная информатика и информационные технологии», «Бизнес-информатика».

УДК 519.6:004(076.5)
ББК 22.18+32.973-018я73

ISBN 978-5-209-05772-7

Содержание

Лабораторный практикум по дисциплине

«Моделирование информационных процессов»

Часть I
Имитационное моделирование в NS-2

I.1. Теоретические сведения

I.1.1. Общее описание NS-2

В данном разделе использованы материалы работ [2–4].

Network Simulator (NS-2) — один из программных симуляторов моделирования процессов в компьютерных сетях. NS-2 позволяет описать топологию сети, конфигурацию источников и приёмников трафика, параметры соединений (полосу пропускания, задержку, вероятность потерь пакетов и т.д.) и множество других параметров моделируемой системы. Данные о динамике трафика, состоянии соединений и объектов сети, а также информация о работе протоколов фиксируются в генерируемом trace-файле.

NS-2 является объектно-ориентированным программным обеспечением. Его ядро реализовано на языке C++. В качестве интерпретатора используется язык скриптов (сценариев) OTcl (Object oriented Tool Command Language). NS-2 полностью поддерживает иерархию классов C++ и подобную иерархию классов интерпретатора OTcl. Обе иерархии обладают идентичной структурой, т.е. существует однозначное соответствие между классом одной иерархии и таким же классом другой. Объединение для совместного функционирования C++ и OTcl производится при помощи TclCl (Classes Tcl). В случае, если необходимо реализовать какую-либо специфическую функцию, не реализованную в NS-2 на уровне ядра, для этого используется код на C++.

Процесс создания модели сети для NS-2 состоит из нескольких этапов:

1) создание нового объекта класса Simulator, в котором содержатся методы, необходимые для дальнейшего описания модели (например, методы `new` и `delete` используются для создания и уничтожения объектов соответственно);
2) описание топологии моделируемой сети с помощью трёх основных функциональных блоков: узлов (nodes), соединений (links) и агентов (agents);
3) задание различных действий, характеризующих работу сети.

Для создания узла используется метод `node`. При этом каждому узлу автоматически присваивается уникальный адрес. Для построения однонаправленных и двунаправленных линий соединения узлов используют методы `simplex-link` и `duplex-link` соответственно.

Важным объектом NS-2 являются агенты, которые могут рассматриваться как процессы и/или как транспортные единицы, работающие на узлах моделируемой сети. Агенты могут выступать в качестве источников трафика или приёмников, а также как динамические

маршрутизирующие и протокольные модули. Агенты создаются с помощью методов общего класса `Agent` и являются объектами его подкласса, т.е. `Agent/type`, где `type` определяет тип конкретного объекта. Например, TCP-агент может быть создан с помощью команды:

```
set tcp [ new Agent/TCP ]
```

Для закрепления агента за конкретным узлом используется метод `attach-agent`. Каждому агенту присваивается уникальный адрес порта для заданного узла (аналогично портам tcp и udp). Чтобы за конкретным агентом закрепить источник, используют методы `attach-source` и `attach-traffic`. Например, можно прикрепить ftp или telnet источники к TCP-агенту. Есть агенты, которые генерируют свои собственные данные, например, CBR-агент (Constant Bit-Rate) — источник трафика с постоянной интенсивностью.

Действия разных агентов могут быть назначены планировщиком событий (Event Scheduler) в определённые моменты времени (также в определённые моменты времени могут быть задействованы или отключены те или иные источники данных, запись статистики, разрыв, либо восстановление соединений, реконфигурация топологии и т.д.). Для этого может использоваться метод `at`. Моделирование начинается при помощи метода `run`.

В качестве дополнения к NS-2 часто используют средство визуализации nam (network animator) для графического отображения свойств моделируемой системы и проходящего через неё трафика и пакет Xgraph для графического представления результатов моделирования.

Запуск сценария NS-2 осуществляется в командной строке с помощью команды:

```
ns [tclscript]
```

Здесь [tclscript] — имя файла скрипта Tcl, который определяет сценарий моделирования (т.е. топологию и различные события).

Nam можно запустить с помощью команды

```
nam [nam-file]
```

Здесь [nam-file] — имя nam trace-файла, сгенерированного с помощью ns.

I.1.2. Список некоторых команд NS-2

Замечание. При копировании текстов листингов не забывайте убирать знак переноса строки «\» и перенабирать кавычки.

Объекты типа NODE

- `$node id` — возвращает идентификатор узла;
- `$node neighbors` — возвращает список соседних узлов;
- `$node attach agent` — прикрепляет агент типа `agent` к узлу;

- `$node detach agent` — отменяет прикрепление агента типа `agent` к узлу;
- `$node agent port` — возвращает ссылку на агента, прикреплённого к порту `port` на данном узле, или пустую строку, если порт не используется.
- `$node reset port` — отменяет прикрепление всех агентов на данном узле и реинициализирует все переменные, связанные с агентами данного узла.
- `$node join-group agent group` — добавляет объект, определяемый объектной ссылкой `agent` для многопользовательской группы с адресом `group`. Это также приводит к выделению соответствующего многопользовательского трафика протоколом групповой работы для обеспечения работы агента `agent`.
- `$node allocaddr` — возвращает адреса в многопользовательской группе в возрастающем порядке для каждого соединения, начиная с 0x8000 и заканчивая 0xFFFF.

DELAYLINK объекты

Объекты данного типа определяют количество времени, требующегося пакетам для прохождения соединения. Время определяется соотношением

```
size/bw + delay,
```

где `size` — размер пакета, `bw` — ширина полосы соединения, `delay` — задержка распространения соединения.

Конфигурационные параметры:

- `bandwidth_` — ширина полосы передачи соединения в бит/сек;
- `delay_` — задержка распространения в сек.

Методы динамики сети

- `$ns rtmodel model model-params node1 [node2]` — данная команда восстанавливает / разрывает соединение между узлами `node1` и `node2` в соответствии с моделью `model`;
  `model-params` содержит все параметры, необходимые для определения модели, и должны быть указаны в виде списка, т.е. параметры должны быть заключены в фигурные скобки;
  `model` может иметь одно из следующих значений: `Deterministic`, `Exponential`, `Manual` или `Trace`.
  Команда возвращает ссылку на объект модели в соответсвии с определением `model`.
  В модели `Deterministic` `model-params` имеют вид:
  `[start-time] up-interval down-interval [finish-time]`

Начиная с момента `start-time`, соединение восстанавливается при `up-interval` и разрывается при `down-interval` до `finish-time`. Значения по умолчанию для `start-time` — 0,5 с, `up-interval` — 2,0 с и `down-interval` — 1,0 с. Значение по умолчанию `finish-time` — время окончания моделирования.
При использовании модели типа `Exponential` параметры записываются в виде: `up-interval down-interval`. Времена восстановления и разрыва соединения выбираются из экспоненциального распределения со средними `up-interval` и `down-interval` соответственно. Значения по умолчанию для средних `up-interval` и `down-interval` — 10,0 с и 1,0 с.
Если используется модель типа `Manual`, в качестве `model-params` применяется `at op`, где `at` определяет время, когда операция `op` должна произойти. Значение `op` может быть `up` или `down`. Альтернативой данной модели может быть метод `rtmodel-at`.
Последний тип `Trace` используется в случае, когда динамика узлов / соединений читается из trace-файла. Аргумент `model-params` представляет собой ссылку на trace-файл, в котором находится информация о динамике.

- `$ns rtmodel-delete model-handle` — уничтожает модель, определённую `model-handle`;
- `$ns rtmodel-at at op node1 [node2]` — используется для указания времени восстановления / разрыва соединения между узлами `node1` и `node2`. Если определён только один узел `node1`, то команда будет применена ко всем соединениям, связанным с данным узлом; `at` определяет время выполнения операции `op`, которая может быть `up` или `down` по отношению к определённому соединению.

Объекты типа QUEUE

Объекты типа очередь — основной класс объектов управления пакетами при их движении по моделируемой топологии.

Конфигурационные параметры:

- `limit_` — размер очереди в пакетах;
- `blocked_` — по умолчанию установлено в false и имеет значение true, если очередь блокирована (не способна посылать пакеты соседнему узлу по соединению);
- `unblock_on_resume_` — по умолчанию установлено в true, показывает, что очередь автоматически разблокируется после передачи последнего полученного пакета.

Подклассы объектов типа Queue

Объекты типа `Drop-Tail` используют простейший алгоритм обработки типа FIFO. Для данного подкласса не определены никакие ме-

тоды, конфигурационные параметры или переменные состояния.

Объекты типа `FQ` используют алгоритм обработки типа *Fair Queuing (алгоритм организации равноправных очередей)*. Конфигурационные параметры:

– `secsPerByte_`

Объекты типа `SFQ` используют алгоритм обработки типа *Stochastic Fair queuing (алгоритм стохастического справедливого обслуживания)*. Конфигурационные параметры:

– `maxqueue_`
– `buckets_`

Объекты типа `DRR` используют *Deficit Round Robin Scheduling* (алгоритм циклического обслуживания с возможностью обработки сверх нормы некоторого количества пакетов (байт) очереди с последующим уменьшением на эту величину (deficit) при реализации следующего цикла обслуживания). Конфигурационные параметры:

– `buckets_` — показывает общее число областей памяти, используемых для смешивания каждого потока;
– `blimit_` — показывает размер используемого буфера в байтах;
– `quantum_` — показывает (в байтах), как много каждый поток может послать за отведённое ему время;
– `mask_` — когда установлено в 1, означает, что отдельный поток состоит из пакетов, имеющих одинаковые идентификаторы узла (и возможно различные идентификаторы порта), в противном случае поток состоит из пакетов с одинаковыми идентификаторами порта и узла.

Объекты типа `RED` используют алгоритм *Random Early-Detection (алгоритм случайного раннего обнаружения)*. Объект может быть сконфигурирован как для отбрасывания, так и для пометки для отбрасывания пакетов. Конфигурационные параметры:

– `bytes_` — установка в true означает byte-mode RED, где размер прибывающих пакетов влияет на вероятность отбрасывания («отметку») пакетов;
– `queue-in-bytes_` — установка в true показывает, что среднее измерение очереди производится в байтах, а не в пакетах. Установка этой опции также приводит к автоматическому масштабированию `thresh_` и `maxthresh_` с помощью `mean_pktsize_`.
– `thresh_` – минимальная граница для среднего размера очереди в пакетах;
– `maxthresh_` — максимальная граница для среднего размера очереди в пакетах;
– `mean_pktsize_` — оценка среднего размера пакета в байтах. Используется для обновления вычисленного среднего размера очереди после периода ожидания;
– `q_weight_` — вес очереди, используется для вычисления среднего размера очереди;

- `wait_` — при установке в true устанавливается интервал между отброшенными пакетами;
- `linterm_` — как средний размер очереди варьируется между `thresh_` и `maxthresh_`, так и вероятность отбрасывания пакета варьируется между 0 и 1/`linterm_`;
- `setbit_` — при установке в true пакеты не отбрасываются, а помечаются на удаление (в пакетах устанавливается бит, характеризующий перегрузку);
- `drop-tail_` – при установке в true вместо механизма randomdrop используется механизм drop-tail в случае переполнения очереди или когда средний размер очереди превосходит `maxthresh_`.

Объекты типа `CBQ` (Class-Based Queue) используют методы обработки в зависимости от классов трафика:

- `$cbq insert $class` — добавляет класс трафика `class` в структуру распределённых соединений, соответствующую объекту соединения `cbq`;
- `$cbq bind $cbqclass $id1 [$id2]` — устанавливает соответствие между пакетами с идентификатором потока `$id1` (или диапазона `$id1 - $id2`) и классом трафика `$cbqclass`.
- `$cbq algorithm $alg` — определяет внутренний алгоритм CBQ; `$alg` может быть установлено в одно из значений: `ancestor-only`, `top-level` или `formal`.

Объекты типа `CBQ/WRR` используют взвешенное циклическое управление (Round-Robin Scheduling) потоками между классами одного приоритетного уровня. Конфигурационные параметры:

- `maxpkt_` — максимальный размер пакета в байтах. Значение этого параметра используется только CBQ/WRR объектами для вычисления максимального выделения полосы для взвешенного циклического управления (Round-Robin Scheduling).

Объекты типа QUEUEMONITOR

Объекты типа QUEUEMONITOR используются для мониторинга получаемых, отправляемых и отбрасываемых пакетов и байтов. Они также поддерживают вычисление статистики (среднего размера очереди и т.д.):

- `$queuemonitor` — сбрасывает все описываемые далее счетчики (прибывших, отправленных и отброшенных байт) в ноль, а также значения интеграторов и элементов задержки, если это определено;
- `$queuemonitor set-delay-samples delaySamp_` — устанавливает `Samples` объект `delaySamp_` для записи статистики о задержках очереди; `delaySamp_` — управляющая переменная для объекта `Samples`, т.е. объект `Samples` должен быть уже создан;

- `$queuemonitor get-bytes-integrator` — возвращает состояние объекта типа `Integrator`, который может быть использован для вычисления интегрального значения размера очереди в байтах;
- `$queuemonitor get-pkts-integrator` — возвращает состояние объекта типа `Integrator`, который может быть использован для вычисления интегрального значения размера очереди в пакетах;
- `$queuemonitor get-delay-samples` — возвращает состояние объекта типа `Samples delaySamp_` для записи статистики о задержках очереди.

Переменные состояния:

- `size_` — текущий размер очереди в байтах;
- `pkts_` — текущий размер очереди в пакетах;
- `parrivals_` — общее число полученных пакетов;
- `barrivals_` — общее число полученных байт;
- `pdepartures_` — общее число отправленных пакетов (не отброшенных);
- `bdepartures_` — общее число байт, содержащееся в отправленных пакетах (не отброшенных);
- `pdrops_` — общее число отброшенных пакетов;
- `bdrops_` — общее число отброшенных байт;
- `bytesInt_` — объект типа `Integrator`, который вычисляет интегральное значение очереди в байтах. Параметр `sum_` содержит сумму (интеграл) размера очереди в байтах;
- `pktsInt_` — объект типа Integrator, который вычисляет интегральное значение очереди в пакетах. Параметр `sum_` содержит сумму (интеграл) размера очереди в пакетах.

Объекты - Агенты

- `$agent port` — возвращает порт транспортного уровня для агента;
- `$agent dst-addr` — возвращает адрес узла назначения, с которым соединён данный агент;
- `$agent dst-port` — возвращает порт узла назначения, с которым соединён данный агент;
- `$agent attach-source type` — устанавливает источник данных типа `type` (см. соответствующие методы агентов для информации о конфигурационных параметрах). Возвращает ссылку на объект источник;
- `$agent attach-traffic traffic-object` — прикрепляет объект `traffic-object` к агенту, который представляет собой генератор трафика `Traffic/Expoo`, `Traffic/Pareto` или `Traffic/Trace`. `Traffic/Expoo` генерирует трафик, основанный на экспоненциальном On/Off распределении; `Traffic/Pareto` — на Парето On/Off распределении; `Traffic/Trace` — на основе trace файла.

– `$agent connect addr port` — соединяет агент `agent` с агентом, имеющим адрес `addr` и порт `port`. Это приводит к тому, что пакеты, отправляемые данным агентом, содержат информацию об адресе и порте, показывающую, что они должны быть направлены соответствующему агенту. Два данных агента должны быть совместимы (т.е. tcp-source/tcp-sink пара может соответствовать паре cbr/tcp-sink).

Конфигурационные параметры:

– `dst_` — адрес назначения, с которым соединён данный агент. Обычно состоит из 32 бит, из которых 24 бита определяют идентификатор узла назначения, а оставшиеся 8 бит — номер порта.

Null объекты

Null объекты — подкласс объектов агентов, являющихся приёмниками трафика. Они наследуют все функциональные особенности данных объектов. Null объекты не имеют никаких методов, конфигурационных параметров и переменных состояния.

LossMonitor объекты

LossMonitor объекты — подкласс объектов агентов, являющихся приёмниками трафика, которые также поддерживают сбор статистики о полученных данных, т.е. число полученных байт, число потерянных пакетов и т.д. Они наследуют все функциональные особенности объектов приёмников трафика:

– `$lossmonitor clear` — сбрасывает ожидаемый порядковый номер пакета (sequence number) в -1.

Параметры состояния:

– `nlost_` — число потерянных пакетов;
– `npkts_` — число полученных пакетов;
– `bytes_` — число полученных байт;
– `lastPktTime_` — время получения последнего пакета;
– `expected_` — ожидаемый порядковый номер (sequence number) следующего пакета.

TCP-объекты

Конфигурационные параметры:

– `window_` — верхняя граница заявленного окна TCP-соединения;
– `maxcwnd_` — верхняя граница окна переполнения TCPсоединения (для отмены ограничения устанавливается в 0);
– `windowInit_` — начальное значение окна переполнения для медленного старта;

- `windowOption_` — алгоритм, использующийся для управления окном переполнения;
- `windowThresh_` — постоянная экспоненциально усредняющего (сглаживающего) фильтра, используемого для вычисления `awnd` (применяется для исследования различных алгоритмов увеличения окна);
- `overhead_` — случайно распределённая переменная, используемая для задержки каждого выходного пакета. Метод состоит в добавлении случайных задержек в источнике для устранения фазовых эффектов (применяется только в версии tcp Tahoe, в tcp Reno не используется);
- `ecn_` — при установке в true указывает, что в дополнение к отбрасыванию пакетов при переполнении используется механизм явного уведомления о перегрузке (*Explicit Congestion Notification, ECN*), что позволяет использовать быстрый повтор передачи (Fast Retransmit) после `quench()` в соответствии с битом ECN;
- `packetSize_` — размер всех пакетов источника;
- `tcpTick_` — таймер TCP, используемый для расчёта *времени двойного оборота пакета (Round-Trip Times, RTT)* — время движения пакета до узла назначения плюс время движения подтверждения. По умолчанию установлен в нестандартную величину 100 ms;
- `bugFix_` — указатель (true/false) запрета механизма быстрой повторной передачи при потере пакетов в одном окне данных;
- `maxburst_` — максимальное число пакетов, которое может посылать источник в ответ на одно полученное подтверждение (ACK) (при отсутствии ограничения устанавливается в 0);
- `slow_start_restart_` — признак использования (1/0) механизма медленного старта;

Определяемые величины:

- `MWS` (Maximum Window Size) — константа, определяющая максимальный размер окна (в пакетах) (по умолчанию `MWS=1024` пакетам).

Переменные состояния:

- `dupacks_` — число дублирующих подтверждений (ACK), полученных после прихода последнего недублирующего подтверждения;
- `seqno_` — наибольший номер последовательности сегмента данных (sequence number) источника TCP;
- `t_seqno_` — текущий номер последовательности сегмента пакета;
- `ack_` — наивысшее значение из полученных подтверждений;
- `cwnd_` — текущее значение окна переполнения;
- `awnd_` — текущее значение окна переполнения при использовании усреднения;
- `ssthresh_` — текущее значение порога медленного старта;
- `rtt_` — оценка значения `round-trip time`;
- `srtt_` — оценка сглаженного значения `round-trip time`;

– `rttvar_` — оценка среднего отклонения значений `round-trip time`;
– `backoff_` — экспоненциальная постоянная задержки для параметра `round-trip time`.

Объекты TCPSINK

TCPSink-объекты представляют собой подкласс объектов агентов, являющихся приёмниками TCP пакетов. Симулятор использует только однонаправленные TCP-соединения, в которых TCP-источник посылает пакеты данных, а приёмник — подтверждения (ACK пакеты). TCPSink-объекты наследуют все функциональные особенности их родительских объектов. Для этих объектов не определено никаких методов и переменных.

Конфигурационные параметры:
– `packetSize_` — размер в байтах всех используемых пакетов подтверждений;
– `maxSackBlocks_` — максимальное число блоков данных, которое может быть подтверждено в опции SACK. Этот параметр используется только подклассом объектов TCPSink/Sack1. Эта величина не может быть увеличена для TCPSink-объекта после того как объект создан (после создания TCPSink-объекта величина может быть уменьшена, но не увеличена).

CONSTANT BIT-RATE объекты

CBR объекты предназначены для генерации пакетов данных с постоянной битовой скоростью:
– `$cbr start` — команда источнику начать генерацию;
– `$cbr stop` — остановка источника.

Конфигурационные параметры:
– `interval_` — задержка между генерацией пакетов;
– `packetSize_` — размер в байтах всех пакетов источника;
– `random_` — параметр определяет, присутствует ли случайный шум в процессе генерации пакетов. Если значение `random_` ноль, то время между генерацией пакетов определяется параметром `interval_`, в противном случае временной промежуток выбирается случайным образом из интервала `[0.5·interval_, 1.5·interval_]`.

Объекты типа SOURCE

Объекты Source генерируют данные для пересылки TCP.

Объекты `SOURCE/FTP`:
– `$ftp start` — команда источнику Source/FTP сгенерировать `maxpkts_` пакетов;

– `$ftp produce n` — команда источнику незамедлительно сгенерировать n пакетов;
– `$ftp stop` — команда прикреплённому TCP агенту прекратить пересылку данных;
– `$ftp attach agent` — прикрепляет Source/FTP объект к агенту `agent`;
– `$ftp producemore count` — команда источнику Source/FTP сгенерировать дополнительно `count` пакетов.

Конфигурационные параметры:
– `maxpkts` — максимальное число пакетов, генерируемых источником.

`TELNET SOURCE` объекты используются для генерации отдельных пакетов с заданными интервалами. Если `interval_` не ноль, то времена между генерацией пакетов выбираются из экспоненциального распределения со средним `interval_`. Если `interval_` имеет значение ноль, то времена между пакетами выбираются с использованием распределения tcplib telnet:
– `$telnet start` — запуск источника;
– `$telnet stop` — остановка источника;
– `$telnet attach agent` — прикрепление объекта Source/Telnet к агенту `agent`.

Конфигурационные параметры:
– `interval_` — среднее время в секундах между пакетами, генерируемыми источником SOURCE/Telnet.

Объекты типа TRAFFIC

Объекты типа `Traffic` создают данные для пересылки по транспортному протоколу. Данные объекты могут быть прикреплены к агентам протокола UDP. Объекты типа `Traffic` создаются методами `Traffic/type`, где `type Expoo`, `Pareto` или `Trace`.

Объекты `Traffic/Expoo` генерируют On/Off трафик. В течение on-периодов пакеты генерируются с постоянной битовой скоростью. Во время off-периодов трафик не генерируется. Данные времена выбираются из экспоненциального распределения.

Конфигурационные параметры:
– `packet-size` — размер пакетов в байтах;
– `burst-time` — период генерации в секундах (on-период);
– `idle-time` — длительность off-периодов;
– `rate` — максимальная скорость в бит в секунду.

`TRAFFIC/PARETO` объекты аналогичны `Traffic/Expoo` объектам, за исключением того, что используется не экспоненциальное распределение Парето.

Конфигурационные параметры:
– `packet-size` — размер пакетов в байтах;

– `burst-time` — период генерации в секундах (on-период);
– `idle-time` — длительность off-периодов;
– `rate` — максимальная скорость в бит в секунду;
– `shape` — параметр формы Парето.

TRAFFIC/TRACE объекты

TRAFFIC/TRACE объекты используются для генерации трафика из trace файла:
– `$trace attach-tracefile tfile` — прикрепляет `Tracefile` объект `tfile` к `trace` объекту. `Tracefile` объект определяет trace-файл, из которого будут читаться данные трафика. К одному `Tracefile` объекту может быть прикреплено несколько `Traffic/Trace` объектов. Для каждого `Traffic/Trace` объекта выбирается случайное стартовое место в `Tracefile`.
Конфигурационные параметры для данного объекта не определены.

Tracefile объекты используются для определения trace файла, на основе которого будет генерироваться трафик
– `$tracefile` — создаёт объект `Tracefile`;
– `$tracefile filename trace-input` — устанавливает имя файла `filename`, из которого `trace` данные будут читаться в `trace-input`.

Методы TRACE и MONITORING

Объекты Trace используются для отслеживания действий в сети и записи этой информации в файл:
– `$ns create-trace type fileID node1 node2` — создаёт объект `Trace` типа `type` и прикрепляет к нему управляющую переменную `fileID` для мониторинга очереди между узлами `node1` и `node2`. Type может быть Enque, Deque или Drop. Enque отслеживает прибывающие в очередь пакеты. Deque — отправляемые пакеты, а Drop — отброшенные. FileID должна быть управляющей файловой переменной, возвращаемой командой `Tcl open`. Соответствующий файл должен быть открыт для записи. Возвращает ссылку на trace объект.
– `$ns drop-trace node1 node2 trace` — удаляет trace объект, прикреплённый к соединению между узлами `node1` и `node2` с управляющей переменной `trace`.
– `$ns trace-queue node1 node2 fileID` — добавляет `Enque`, `Deque` и `Drop` мониторинг к соединению между узлами `node1` и `node2`.
– `$ns trace-all fileID` — добавляет `Enque`, `Deque` и `Drop Tracing` на все соединения топологии, созданные после вызова данной команды. Также добавляет отслеживание сетевой динамики. `FileID` должна быть управляющей файловой переменной, возвращаемой

командой `Tcl open`. Соответствующий файл должен быть открыт для записи.
- `$ns monitor-queue node1 node2` — добавляет мониторинг длины очереди между узлами `node1` и `node2`. Возвращает объект типа `QueueMonitor`, который может быть использован для вычисления среднего размера очереди и т.д.
- `$ns flush-trace` — закрывает выходные каналы, прикреплённые ко всем trace объектам.
- `$link trace-dynamics ns fileID` — отслеживает динамику данного соединения и записывает данные в файл с управляющей переменной `fileID`; `ns` — переменная объекта типа `Simulator` или `MultiSim`, который был создан для обеспечения моделирования.

I.1.3. Файл трассировки

Трейс-файл представляет собой текст в формате ASCII, в котором зарегистрированы необходимые события моделирования (рис. I.1.1).

```
+ 0.02896 2 3 tcp 1040 ------- 2 4.0 3.1 2 3
- 0.02896 2 3 tcp 1040 ------- 2 4.0 3.1 2 3
+ 0.03 5 2 tcp 40 ------- 4 5.0 3.3 0 4
- 0.03 5 2 tcp 40 ------- 4 5.0 3.3 0 4
r 0.03096 2 3 tcp 1040 ------- 2 4.0 3.1 1 2
+ 0.03096 3 2 ack 40 ------- 2 3.1 4.0 1 5
- 0.03096 3 2 ack 40 ------- 2 3.1 4.0 1 5
```

Рис. I.1.1. Отрывок из trace-файла

Рассмотрим структуру трейс-файла (рис. I.1.2).

Событие	Время	От узла	К узлу	Тип пакета	Размер пакета	Флаги	Идент. потока	Адр. ист.	Адр. получ.	Поряд. номер	Идент. пакета

Рис. I.1.2. Структура trace-файла

- В поле «событие» (code) могут стоять символы:
  - `r`: receive — принятие пакета узлом;
  - `+`: enque — постановка в очередь;
  - `-`: deque — снятие с очереди;
  - `d`: drop — отбрасывание пакета из очереди;
  - `h`: hop — указывает на переход к следующему узлу.
- Поле «время» (time) показывает модельное время данного события в секундах.

- Поля: «от узла» (hsrc) и «к узлу» (hdst) показывают, в каком звене происходит данное событие.
- Поле «тип пакета» (packet type) указывает на то, к какому приложению или агенту относится данный пакет (tcp|telnet|cbr|ack и т.д.).
- Поле «размер пакета» (size packet) показывает размер пакета на сетевом уровне с учётом заголовка IP.
- Поле «флаги» (flags) содержит шесть флагов:
  - E — опытная индикация перегрузки;
  - N — индикация явного извещения о перегрузке с возможностью переноса;
  - C — эхо явного извещения о перегрузке;
  - A — сокращение окна перегрузки
  - P — приоритет (priority);
  - F — быстрый старт TCP (TCP Fast Start).
- Поле «идентификатор потока» (flowID) совпадает с полем fid (flow ID) в заголовке IPv6 (применяется для анализа сети, а также для использования разных цветов в визуализаторе nam)
- Поля «адрес источника» (src.sport) и «адрес получателя» (dst.sport) имеют формат `узел.порт` (например: 1.0).
- Поле «порядковый номер» (seq) показывает порядковый номер пакета на транспортном уровне.
- Поле «идентификатор пакета» (pktID) показывает уникальный идентификатор пакета.

I.1.4. NAM

В качестве средства анимации в NS-2 используется nam (Network Animator). Он графически воспроизводит имитационную модель (топологию сети, анимацию прохождения пакетов по сети, постановки их в очередь и т.д.) и наглядно показывает алгоритмы работы протоколов, дисциплин обслуживания очередей.

Запустить nam можно либо с помощью команды

```
nam <nam-file>
```

Здесь `<nam-file>` — имя трейс-файла nam, созданного ns2.

На рис. I.1.3 приведён интерфейс пользователя nam.

Интерфейс содержит зону анимации, несколько меню и кнопок.

В меню *Views* находятся следующие пункты:

- *New view* — создаёт новый вид той же анимации. Пользователь может увеличивать и прокручивать изображение в новом окне.
- *Show monitors* — показывает окно в нижней части экрана, где осуществляется мониторинг.

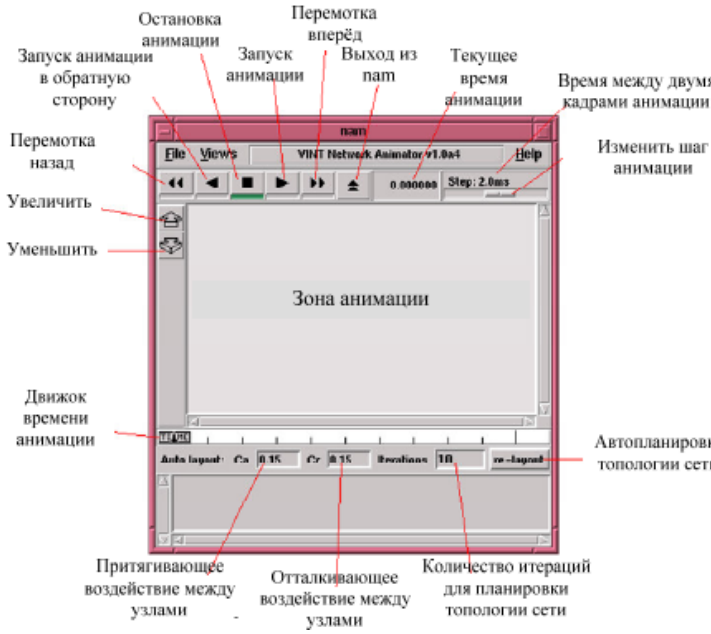


Рис. I.1.3. Интерфейс пользователя nam

– *Show autolayout* — показывает окно в нижней части экрана, которое содержит окна для ввода данных и кнопки для настроек автоматического расположения.
– *Show annotation* – показывает пункт в нижней части экрана с примечаниями по мере увеличения модельного времени.

В меню *Analysis* находятся следующие пункты:

– *Active Sessions* — открывает окно со списком активных на данный момент сессий;
– *Legend ...* — открывает окно с описанием условных обозначений.

Меню *Help* содержит справочную информацию об аниматоре.

Под панелью меню находятся кнопки *перемотки назад*, *запуска анимации в обратную сторону*, *остановки анимации*, *запуска анимации*, *перемотки вперёд* и *выхода* из nam, *индикатор текущего времени анимации* и *движок изменения скорости анимации* (текущая скорость изображена над ним). Также возможно увеличение и уменьшение изображения с помощью кнопок, расположенных в левой части экрана. Изменять текущее время анимации пользователь может при помощи *движка времени анимации*. Под движком времени анимации находится *панель автопланировки топологии сети* (изначально

может отсутствовать).

Существует три параметра для настройки процесса автоматической планировки:

- *Ca* — константа притягивающего воздействия между узлами, которая контролирует силу сжатия между узлами в зоне анимации.
- *Cr* — константа отталкивающего воздействия между узлами, которая контролирует силу отталкивания между узлами в зоне анимации.
- *Количество итераций* — определяет, сколько раз запускать процедуру автопланировки.

Для маленьких топологий с десятками узлов использование исходных параметров (с 20–30 итерациями) достаточно для изображения приемлемого вида сети. Но для больших топологий необходимо изменение этих параметров внутри скрипта ns-2 предназначенными для этого командами.

I.1.5. Основы работы в Xgraph

Xgraph разработан для построения графиков в среде X-Windows. Xgraph способен читать данные из файлов или непосредственно со стандартного ввода. Он может отображать до 64 независимых наборов данных, используя различные цвета и/или различные стили линий для каждого набора. При отображении графиков доступен вывод заголовков, меток осей, линий разметки или специальных отметок и условных обозначений.

Интерфейс, используемый для определения размера и расположения окна, зависит от используемого менеджера X-Windows. После того как окно было создано, все наборы данных отображаются графически с условными обозначениями в верхнем правом углу окна. Для увеличения части отображаемого графика нужно выделить её в окне xgraph, после чего она автоматически будет отображена в новом окне. Xgraph также имеет три кнопки управления в верхнем левом углу каждого окна: Close, Hardcopy и About.

Формат вызова Xgraph:

```
xgraph [ options ] [[-geometry |=]WxH+X+Y ] \
  [ -display host:display.screen ] [ file ... ]
```

Некоторые опции Xgraph:

- `-geometry WxH+X+Y или \=WxH+X+Y (Geometry)` — определение начального положения и размера окна xgraph;
- `-device [name]` — установка выходного устройства для xgraph (по умолчанию установлено 'X', другие доступные устройства — ps, hpgl, idraw и tgif);
- `-fitx` — масштабирование x-координат всех наборов данных к промежутку [0..1];

- `-fity` — масштабирование y-координат всех наборов данных к промежутку [0..1];
- `-scale [factor]` — выходной масштабный множитель для устройств postscript, hpgl и idraw. По умолчанию 1.0 и, например, при установке в 0.5 будет сгенерировано изображение с размером 50 % от исходного;
- `-fmtx [printf-format] -fmty [printf-format]` — устанавливает определенный формат отображения x или y осей;
- `-bb (BoundBox)` — отрисовка прямоугольников вокруг отображаемых данных. Полезно в случае использования меток вместо линий для отображения данных (см. опцию `-tk`);
- `-bd [color] (Border)` — определяет цвет границ для окна xgraph;
- `-bg [color] (Background)` — определяет цвет фона для окна xgraph;
- `-bw [size] (BorderSize)` — ширина границы окна xgraph;
- `-db (Debug)` — запуск xgraph в синхронном режиме и отображение значений всех величин, установленных по умолчанию;
- `-fg [color] (Foreground)` — установка цвета, которым отображаются все линии и текст в xgraph;
- `-gw (GridSize)` — ширина в пикселях линий разметки;
- `-gs (GridStyle)` — задание стиля отображаемых линий разметки;
- `-lf [fontname] (LabelFont)` – задание шрифта меток.
- `-lnx (LogX)` — отображение оси X в логарифмическом масштабе (разметка оси отображает степени десяти);
- `-lnx (LogY)` — отображение оси Y в логарифмическом масштабе (разметка оси отображает степени десяти);
- `-lw width (LineWidth)` — определяет ширину линий отображения данных в пикселях.
- `-lx [xl,xh] (XLowLimit, XHighLimit)` — опция ограничивает диапазон оси X определённым интервалом. Вместе с опцией `-ly` используется для масштабирования нужных участков больших графиков;
- `-ly [yl,yh] (YLowLimit, YHighLimit)` — опция ограничивает диапазон оси Y определённым интервалом.
- `-m (Markers)` — отметка каждой точки данных заданным маркером. В xgraph имеется восемь типов маркеров. Каждый тип имеет определенный вид линий и цвет.
- `-M (StyleMarkers)` — аналогично опции `-m`, но отдельный маркер присваивается каждому набору данных.
- `-nl (NoLines)` — опция отключает отображение линий. Вместе с опциями -m, -M, -p, или -P используется для отображения точечных графиков;
- `-ng (NoLegend)` — отключение отображения условных обозначений;
- `-p (PixelMarkers)` — маркирование каждой точки данных отдельным небольшим маркером (пиксельного размера). Обычно исполь-

зуется вместе с опцией `-nl` для построения точечных графиков;
- `-P (LargePixels)` — аналогично `-p`, но используются крупные маркеры;
- `-t [string] (TitleText)` — определение заголовка графика (строка-заголовок будет отображена в центре сверху графика);
- `-tk (Ticks)` — отображение данных с помощью маркеров, а не линий;
- `-tkax (Tick Axis)` — отображает оси при использовании маркеров;
- `-x [unitname] (XUnitText)` – определяет подпись к оси X;
- `-y [unitname] (YUnitText)` — определяет подпись к оси Y;
- `-zg [color] (ZeroColor)` — определяет цвет, используемый для отображения нулевой осевой отметки;
- `-zw [width] (ZeroWidth)` — ширина нулевой осевой отметки в пикселях.

I.1.6. Основы работы в Gnuplot

Gnuplot — программа для построения графиков функций и визуализации различных данных.

Работа в Gnuplot возможна в двух режимах:
- пакетном — готовится специальный файл, содержащий последовательность команд;
- интерактивном — обращение к программе осуществляется через командную строку в режиме реального времени.

Базовые команды Gnuplot:
- `help` — вывести справку;
- `load <имя файла>` — загрузить командный файл.

Терминалом в gnuplot является то устройство (или файл), в которое будет осуществляться вывод полученного результата. Таковым может быть монитор, принтер или же файл с расширением png, jpg, eps и др., а также latex-файл. Тип терминала задаётся командой:

```
set terminal <тип терминала>
```

Здесь `<тип терминала>` может принимать следующие значения:
- `windows` — вывод данных на дисплей в ОС Windows;
- `X11` — вывод данных на дисплей в ОС Linux;
- `png` — вывод данных в файл формата png (растровый формат);
- `jpeg` — вывод данных в файл формата jpeg (растровый формат);
- `postscript eps` — вывод данных в файл формата eps (векторный формат);
- `latex` — вывод данных в файл формата LaTex.

Пример вывода в файл:

```
#устанавливаем тип терминала
set terminal postscript eps
```

```
#устанавливаем имя выходного файла
set output "plot1.eps"
```

Для построения графика функции на плоскости используется команда `plot` (знак «\» обозначает переход на другую строку):

```
plot [<изменение аргумента>] [<изменение функции>] \
     <функция> <доп. параметры>
```

Например, график синусоиды при изменении x от -2π до 2π:

```
plot [-2*pi:2*pi] sin(x)
```

Область изменения значений аргумента/функции использует команды:

```
set xrange [<нач. значение>:<конечн. значение>]
set yrange [<нач. значение>:<конечн. значение>]
```

При выводе Gnuplot позволяет устанавливать различные визуальные параметры для графика. Для этого в команде `plot` после объявления функции следует ввести:

```
with <стиль графика> \
linetype <тип, целое число (комбинация стиль+цвет)>
```

Некоторые простые стили:
– `lines` (по умолч.) — линии;
– `points` — точки;
– `lines and points` — линии с точками;
– `dots` — очень маленькие точки;
– `impulses` — дискретные прямые;
– `steps` — ломаная под прямым углом линия.

Для изменения цвета и толщины линии графика в команде `plot` нужно указать:

```
linestyle <цвет, целое число> \
linewidth <толщина линии, pt>
```

Приведём пример построения графика функции $\sin(x)$:

```
#!/usr/bin/gnuplot -persist

# вывод в eps-файл
set terminal postscript eps enhanced color

# назначаем выходной файл
set output "plot1.eps"

# устанавливаем кодировку для кириллицы
set encoding utf8

# оси OX и OY, шрифт
```

```
set ylabel "ось y" font "Helvetica,18"
set xlabel "ось x" font "Helvetica,18"

# отступы
set bmargin 4  # отступ снизу
set lmargin 10 # отступ слева
set rmargin 10 # отступ справа
set tmargin 4  # отступ сверху

# метки по осям Ox,Oy
set xtics ("10"0,"20"1,"30"2)
set ytics ("-30"-2,"-20"-1,"0"0,"20"1,"30"2)

# изменение по Ox, Oy
set xrange [-2*pi:2*pi]
set yrange [-2:2]

# построение, цвет, толщина
plot sin(x) with lines lt 3 lw 2
```

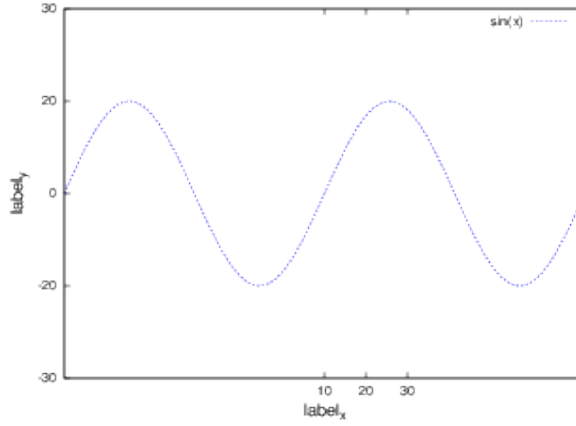


Рис. I.1.4. График функции $y = \sin(x)$

Для построения поверхностей в Gnuplot применяется команда `splot`. Обращение к ней происходит аналогично команде `plot`, за исключением некоторых особенностей.

Формировать данные для построения графиков можно не только

задавая промежутки изменения переменных, но и используя заранее подготовленный файл с данными.

I.1.7. AWK

AWK — утилита, предназначенная для простых (механических и вычислительных) манипуляций над данными:

- использует метод поиска по шаблону (pattern matching);
- оперирует двумя видами входных данных: файлом данных и командным;
- файл данных — упорядоченные данные, состоящие из строк, которые в свою очередь состоят из групп знаков (слов), разделённых пробелами;
- командный файл — инструкции (команды) поиска по шаблону;
- AWK — интерпретатор, исполняющий действия, записанные в командном файле, над файлом данных;
- все команды одновременно могут использовать все переменные программы AWK и только одну строку из файла данных — она автоматически загружается в специальные переменные;
- переменная `$0` содержит всю строку, `$1` — первое слово в строке, `$2` — второе слово и т.д. (максимальное количество — 100 слов).

Пример кода AWK, вычисляющий среднее значение чисел, записанных в четвёртой колонке:

```
BEGIN {FS =" "}{nl++} {s=s+$4} END {print "average:" s/nl}
```

Лабораторная работа 1. Простые модели компьютерной сети

Цель работы

Приобретение навыков моделирования сетей передачи данных с помощью средства имитационного моделирования NS-2, а также анализ полученных результатов моделирования.

1.1. Шаблон сценария для NS-2

В своём рабочем каталоге создайте директорию `mip`, к которой будут выполняться лабораторные работы. Внутри `mip` создайте директорию `lab-ns`, а в ней файл `shablon.tcl`:

```
mkdir -p mip/lab-ns
cd mip/lab-ns
touch shablon.tcl
```

Откройте на редактирование файл `shablon.tcl`. Можно использовать любой текстовый редактор типа emacs.

Сначала создадим объект типа Simulator:

```
# создание объекта  Simulator
set ns [new Simulator]
```

Затем создадим переменную `nf` и укажем, что требуется открыть на запись nam-файл для регистрации выходных результатов моделирования:

```
# открытие на запись файла out.nam для визуализатора nam
set nf [open out.nam w]

# все результаты моделирования будут записаны в переменную nf
$ns namtrace-all $nf
```

Вторая строка даёт команду симулятору записывать все данные о динамике модели в файл `out.nam`.

Далее создадим переменную `f` и откроем на запись файл трассировки для регистрации всех событий модели:

```
# открытие на запись файла трассировки out.tr
# для регистрации всех событий
set f [open out.tr w]

# все регистрируемые события будут записаны в переменную f
$ns trace-all $f
```

После этого добавим процедуру `finish`, которая закрывает файлы трассировки и запускает nam:

```
# процедура finish закрывает файлы трассировки
# и запускает визуализатор nam
proc finish {} {
  global ns f nf      # описание глобальных переменных
  $ns flush-trace     # прекращение трассировки
  close $f            # закрытие файлов трассировки
  close $nf           # закрытие файлов трассировки nam

  # запуск nam в фоновом режиме
  exec nam out.nam &
  exit 0
}
```

Наконец, с помощью команды `at` указываем планировщику событий, что процедуру `finish` следует запустить через 5 с после начала моделирования, после чего запустить симулятор `ns`:

```
# at-событие для планировщика событий, которое запускает
# процедуру finish через 5 с после начала моделирования
$ns at 5.0 "finish"

# запуск модели
$ns run
```

Сохранив изменения в отредактированном файле `shablon.tcl` и закрыв его, можно запустить симулятор командой:

```
ns shablon.tcl
```

При этом на экране появится сообщение типа

```
nam: empty trace file out.nam
```

поскольку ещё не определены никакие объекты и действия.

Получившийся шаблон можно использовать в дальнейшем в большинстве разрабатываемых скриптов NS-2, добавляя в него до строки `$ns at 5.0 "finish"` описание объектов и действий моделируемой системы.

1.2. Простой пример описания топологии сети, состоящей из двух узлов и одного соединения

Постановка задачи. Требуется смоделировать сеть передачи данных, состоящую из двух узлов, соединённых дуплексной линией связи с полосой пропускания 2 Мб/с и задержкой 10 мс, очередью с обслуживанием типа DropTail. От одного узла к другому по протоколу UDP

осуществляется передача пакетов, размером 500 байт, с постоянной скоростью 200 пакетов в секунду.

Реализация модели. Скопируем содержимое созданного шаблона в новый файл:

```
cp shablon.tcl example1.tcl
```

и откроем `example1.tcl` на редактирование. Добавим в него до строки `$ns at 5.0 "finish"` описание топологии сети:

```
# создание 2-х узлов:
set n0 [$ns node]
set n1 [$ns node]

# соединение 2-х узлов дуплексным соединением
# с полосой пропускания 2 Мб/с и задержкой 10 мс,
# очередью с обслуживанием типа  DropTail
$ns duplex-link $n0 $n1 2Mb 10ms DropTail
```

Создадим агенты для генерации и приёма трафика:

```
# создание агента UDP и присоединение его к узлу n0
set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0

# создание источника трафика CBR (constant bit rate)
set cbr0 [new Application/Traffic/CBR]

# устанавливаем размер пакета в 500 байт
$cbr0 set packetSize_ 500

#задаем интервал между пакетами равным 0.005 секунды,
#т.е. 200 пакетов в секунду
$cbr0 set interval_ 0.005

# присоединение источника трафика CBR к агенту udp0
$cbr0 attach-agent $udp0
```

Создаётся агент UDP и присоединяется к узлу n0. В узле агент сам не может генерировать трафик, он лишь реализует протоколы и алгоритмы транспортного уровня. Поэтому к агенту присоединяется приложение. В данном случае — это источник с постоянной скоростью (Constant Bit Rate, CBR), который каждые 5 мс посылает пакет $R =$ 500 байт. Таким образом, скорость источника:

$$R = \frac{500 \cdot 8}{0,005} = 800000 \text{ бит/с}.$$

Далее создадим Null-агент, который работает как приёмник трафика, и прикрепим его к узлу n1:

```
# Создание агента-приёмника и присоединение его к узлу n1
set null0 [new Agent/Null]
$ns attach-agent $n1 $null0
```

Соединим агенты между собой:

```
# Соединение агентов между собой
$ns connect $udp0 $null0
```

Для запуска и остановки приложения CBR добавляются at-события в планировщик событий (перед командой `$ns at 5.0 "finish"`)

```
# запуск приложения через 0,5 с
$ns at 0.5 "$cbr0 start"

# остановка приложения через 4,5 с
$ns at 4.5 "$cbr0 stop"
```

Сохранив изменения в отредактированном файле и запустив симулятор:

```
ns example1.tcl
```

получим в качестве результата запуск аниматора nam в фоновом режиме (рис. 1.1).

При нажатии на кнопку play в окне nam через 0.5 секунды из узла 0 данные начнут поступать к узлу 1. Это процесс можно замедлить, выбирая шаг отображения в nam. Можно осуществлять наблюдение за отдельным пакетом, щёлкнув по нему в окне nam, а щёлкнув по соединению, можно получить о нем некоторую информацию.

1.3. Пример с усложнённой топологией сети

Постановка задачи. Описание моделируемой сети (рис. 2.4):

- сеть состоит из 4 узлов (n0, n1, n2, n3);
- между узлами n0 и n2, n1 и n2 установлено дуплексное соединение с пропускной способностью 2 Мбит/с и задержкой 10 мс;
- между узлами n2 и n3 установлено дуплексное соединение с пропускной способностью 1,7 Мбит/с и задержкой 20 мс;
- каждый узел использует очередь с дисциплиной DropTail для накопления пакетов, максимальный размер которой составляет 10;
- TCP-источник на узле n0 подключается к TCP-приёмнику на узле n3 (по-умолчанию, максимальный размер пакета, который TCP-агент может генерировать, равняется 1KByte)

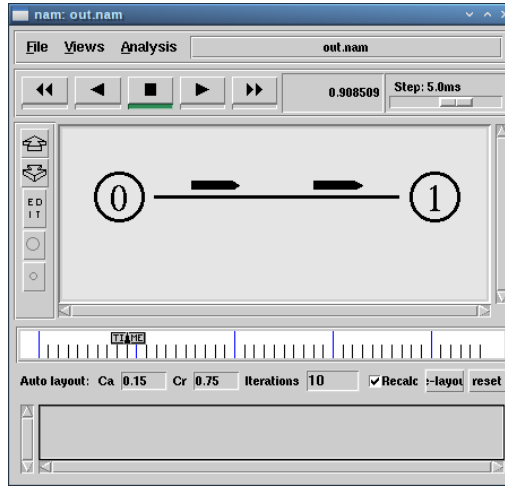


Рис. 1.1. Визуализация простой модели сети с помощью nam

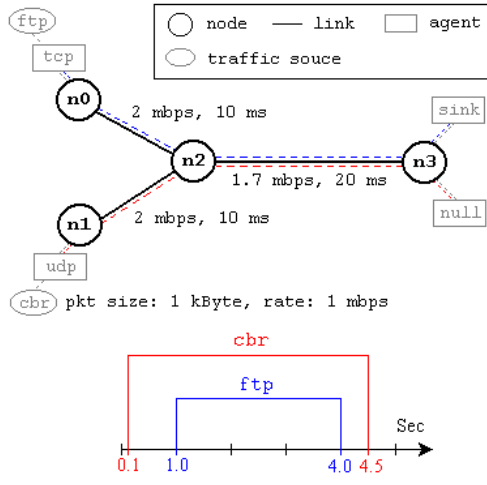


Рис. 1.2. Схема моделируемой сети

– TCP-приёмник генерирует и отправляет ACK пакеты отправителю и откидывает полученные пакеты;
– UDP-агент, который подсоединён к узлу n1, подключён к null-

агенту на узле n3 (null-агент просто откидывает пакеты);
– генераторы трафика ftp и cbr прикреплены к TCP и UDP агентам соответственно;
– генератор cbr генерирует пакеты размером 1 Кбайт со скоростью 1 Мбит/с;
– работа cbr начинается в 0,1 секунду и прекращается в 4,5 секунды, а ftp начинает работать в 1,0 секунду и прекращает в 4,0 секунды.

Реализация модели. Скопируем содержимое созданного шаблона в новый файл:

```
cp shablon.tcl example2.tcl
```

и откроем `example2.tcl` на редактирование.

Создадим 4 узла и 3 дуплексных соединения с указанием направления:

```
for {set i 0} {$i < 4} {incr i} {
  set n($i) [$ns node]
}

$ns duplex-link $n(0) $n(2) 2Mb 10ms DropTail
$ns duplex-link $n(1) $n(2) 2Mb 10ms DropTail
$ns duplex-link $n(3) $n(2) 2Mb 10ms DropTail

$ns duplex-link-op $n(0) $n(2) orient right-down
$ns duplex-link-op $n(1) $n(2) orient right-up
$ns duplex-link-op $n(2) $n(3) orient right
```

Создадим агент UDP с прикреплённым к нему источником CBR и агент TCP с прикреплённым к нему приложением FTP:

```
# создание агента UDP и присоединение его к узлу n(0)
set udp0 [new Agent/UDP]
$ns attach-agent $n(0) $udp0

# создание источника CBR-трафика
# и присоединение его к агенту udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0

# создание агента TCP и присоединение его к узлу n(1)
set tcp1 [new Agent/TCP]
$ns attach-agent $n(1) $tcp1

# создание приложения FTP
```

```
# и присоединение его к агенту tcp1
set ftp [new Application/FTP]
$ftp attach-agent $tcp1
```

Создадим агенты-получатели:

```
# создание агента-получателя для udp0
set null0 [new Agent/Null]
$ns attach-agent $n(3) $null0

# создание агента-получателя для tcp1
set sink1 [new Agent/TCPSink]
$ns attach-agent $n(3) $sink1
```

Соединим агенты udp0 и tcp1 и их получателей:

```
$ns connect $udp0 $null0
$ns connect $tcp1 $sink1
```

Зададим описание цвета каждого потока:

```
$ns color 1 Blue
$ns color 2 Red

$udp0 set class_ 1
$tcp1 set class_ 2
```

Отслеживание событий в очереди:

```
$ns duplex-link-op $n(2) $n(3) queuePos 0.5
```

Наложение ограничения на размер очереди:

```
$ns queue-limit $n(2) $n(3) 20
```

Добавление at-событий:

```
$ns at 0.5 "$cbr0 start"
$ns at 1.0 "$ftp start"
$ns at 4.0 "$ftp stop"
$ns at 4.5 "$cbr0 stop"
```

Сохранив изменения в отредактированном файле и запустив симулятор, получим анимированный результат моделирования (рис. 1.3).

При запуске скрипта можно заметить, что по соединениям между узлами n(0)–n(2) и n(1)–n(2) к узлу n(2) передаётся данных больше, чем способно передаваться по соединению от узла n(2) к узлу n(3). Действительно, мы передаём 200 пакетов в секунду от каждого источника данных в узлах n(0) и n(1), а каждый пакет имеет размер 500 байт. Таким образом, полоса каждого соединения $0,8$ Mb, а суммарная — $1,6$ Mb. Но соединение n(2)–n(3) имеет полосу лишь 1 Mb.

Следовательно, часть пакетов должна теряться. В окне аниматора можно видеть пакеты в очереди, а также те пакеты, которые отбрасываются при переполнении.

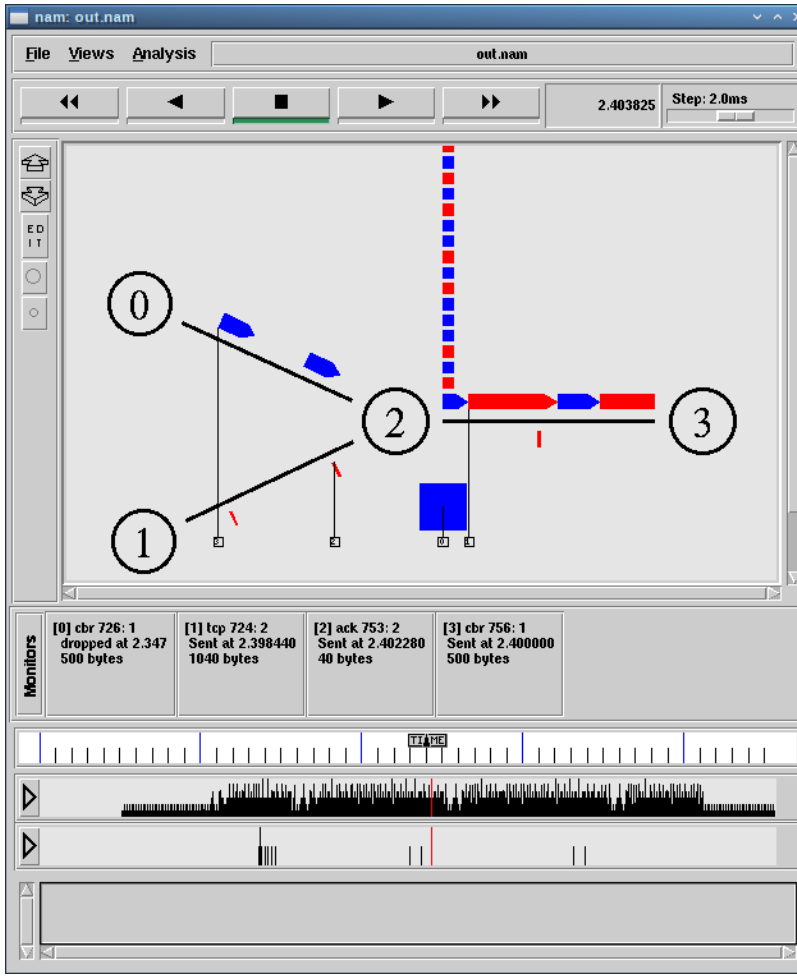


Рис. 1.3. Мониторинг очереди в визуализаторе nam

1.4. Пример с кольцевой топологией сети

Постановка задачи. Требуется построить модель передачи данных по сети с кольцевой топологией и динамической маршрутизацией пакетов:

- сеть состоит из 7 узлов, соединённых в кольцо;
- данные передаются от узла `n(0)` к узлу `n(3)` по кратчайшему пути;
- с 1 по 2 секунду модельного времени происходит разрыв соединения между узлами `n(1)` и `n(2)`;
- при разрыве соединения маршрут передачи данных должен измениться на резервный.

Реализация модели. Скопируем содержимое созданного шаблона в новый файл:

```
cp shablon.tcl example3.tcl
```

и откроем `example3.tcl` на редактирование.

Опишем топологию моделируемой сети:

```
for {set i 0} {$i < 7} {incr i} {
  set n($i) [$ns node]
}
```

Далее соединим узлы так, чтобы создать круговую топологию:

```
for {set i 0} {$i < 7} {incr i} {
  $ns duplex-link $n($i) $n([expr ($i+1)%7]) 1Mb 10ms DropTail
}
```

Каждый узел, за исключением последнего, соединяется со следующим, последний соединяется с первым. Для этого в цикле использован оператор %, означающий остаток от деления нацело.

Зададим передачу данных от узла `n(0)` к узлу `n(3)`:

```
set udp0 [new Agent/UDP]
$ns attach-agent $n(0) $udp0
set cbr0 [new Agent/CBR]
$ns attach-agent $n(0) $cbr0
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005

set null0 [new Agent/Null]
$ns attach-agent $n(3) $null0

$ns connect $cbr0 $null0
```

Данные передаются по кратчайшему маршруту от узла `n(0)` к узлу `n(3)`, через узлы `n(1)` и `n(2)` (рис. 1.4).

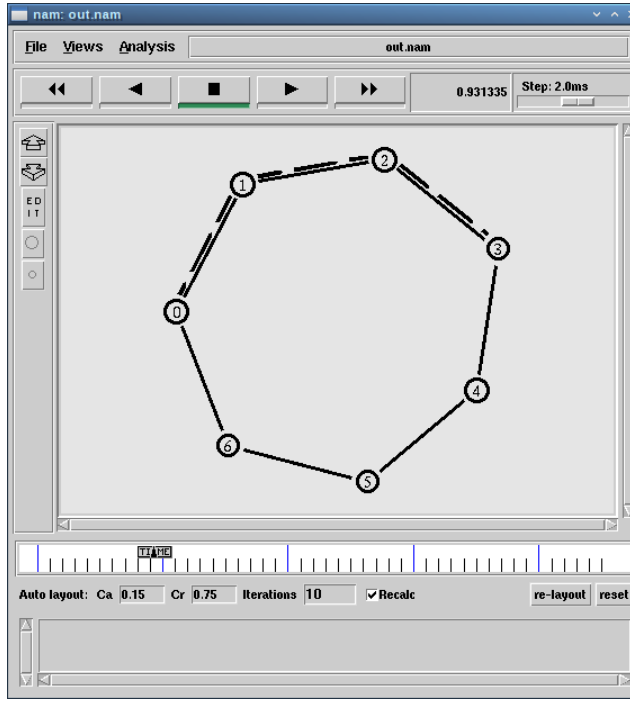


Рис. 1.4. Передача данных по кратчайшему пути сети с кольцевой топологией

Добавим команду разрыва соединения между узлами `n(1)` и `n(2)` на время в одну секунду:

```
$ns rtmodel-at 1.0 down $n(1) $n(2)
$ns rtmodel-at 2.0 up $n(1) $n(2)
```

Передача данных при кольцевой топологии сети в случае разрыва соединения представлена на рис. 1.5.

Добавив в начало скрипта после команды создания объекта Simulator:

```
$ns rtproto DV
```

увидим, что сразу после запуска в сети отправляется небольшое количество маленьких пакетов, используемых для обмена информацией, необходимой для маршрутизации между узлами (рис. 1.6). Когда соединение будет разорвано, информация о топологии будет обновлена,

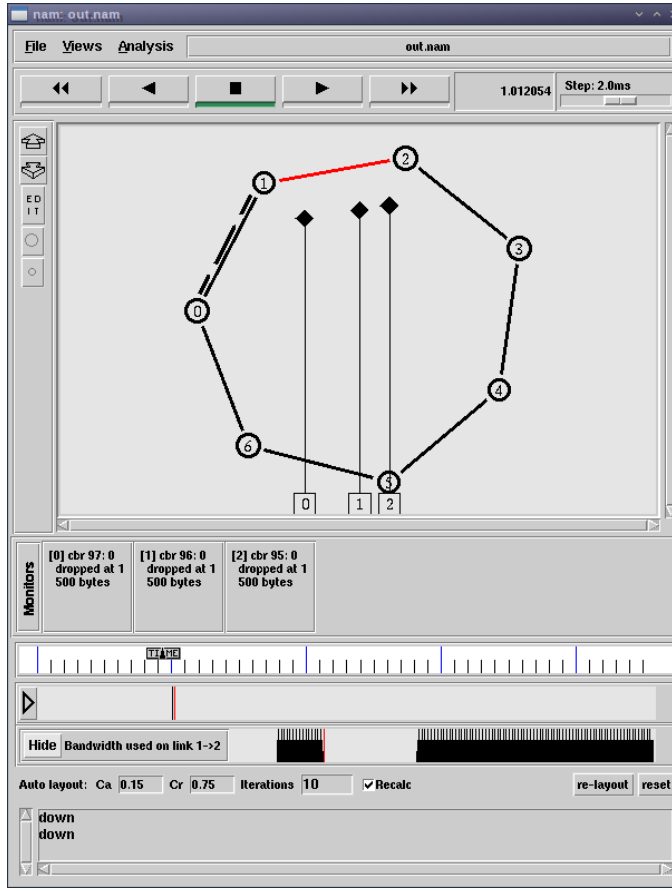


Рис. 1.5. Передача данных по сети с кольцевой топологией в случае разрыва соединения

и пакеты будут отсылаться по новому маршруту через узлы `n(6)`, `n(5)` и `n(4)`.

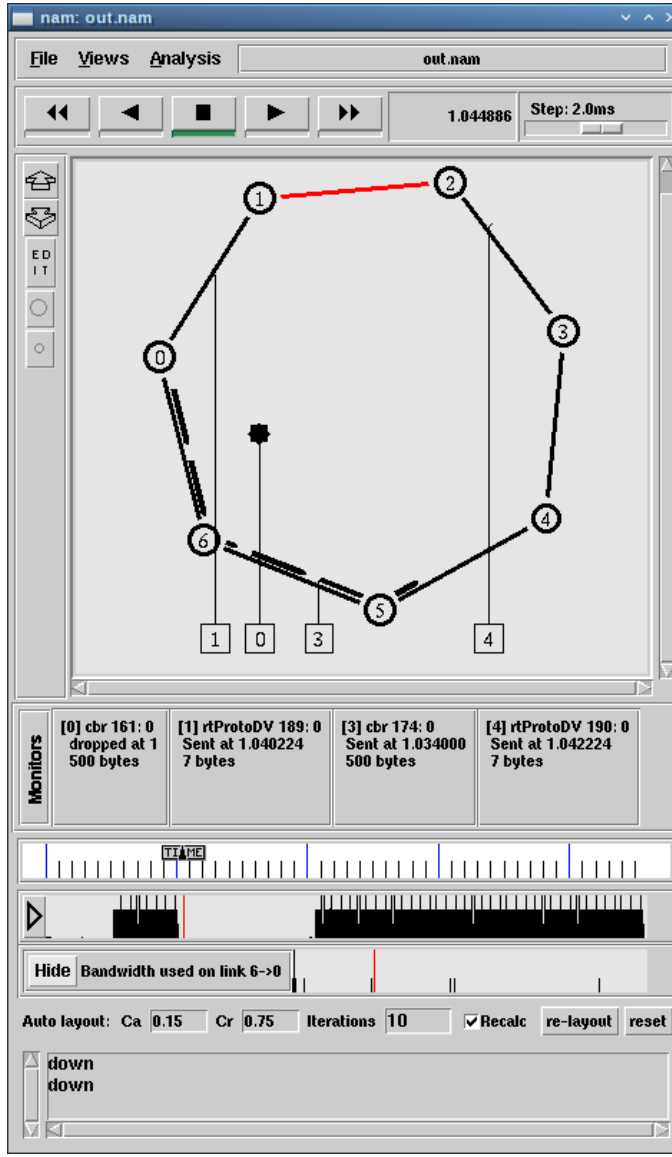


Рис. 1.6. Маршрутизация данных по сети с кольцевой топологией в случае разрыва соединения

Лабораторная работа 2. Исследование протокола TCP и алгоритма управления очередью RED

2.1. Предварительные сведения.

2.1.1. Протокол TCP

Протокол управления передачей (Transmission Control Protocol, **TCP**) имеет средства управления потоком и коррекции ошибок, ориентирован на установление соединения.

Рис. 2.1. Формат заголовка пакета TCP

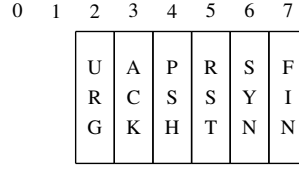


Рис. 2.2. Поле *Флаги* заголовка пакета TCP

Флаг *Указатель срочности (Urgent Pointer, URG)* устанавливается в 1 в случае использования поля *Указатель на срочные данные*.

Флаг *Подтверждение (Acknowledgment, ACK)* устанавливается в 1 в случае, если поле *Номер подтверждения (Acknowledgement Number)* содержит данные. В противном случае это поле игнорируется.

Флаг *Выталкивание (Push, PSH)* означает, что принимающий стек TCP должен немедленно информировать приложение о поступивших данных, а не ждать, пока буфер заполниться.

Флаг *Сброс (Reset, RST)* используется для отмены соединения из-за ошибки приложения, отказа от неверного сегмента, попытки создать соединение при отсутствии затребованного сервиса.

Флаг *Синхронизация (Synchronize, SYN)* устанавливается при инициировании соединения и синхронизации порядкового номера.

Флаг *Завершение (Finished, FIN)* используется для разрыва соединения. Он указывает, что отправитель закончил передачу данных.

Управление потоком в протоколе TCP осуществляется при помощи *скользящего окна* переменного размера:

- поле *Размер окна (Window)* (длина 16 бит) содержит количество байт, которое может быть послано после байта, получение которого уже подтверждено;
- если значение этого поля равно нулю, это означает, что все байты, вплоть до байта с номером *Номер подтверждения* - 1, получены, но получатель отказывается принимать дальнейшие данные;
- разрешение на дальнейшую передачу может быть выдано отправкой сегмента с таким же значением поля *Номер подтверждения* и ненулевым значением поля *Размер окна*.

Регулирование трафика в TCP:

- *контроль доставки* — отслеживает заполнение входного буфера получателя с помощью параметра *Размер окна (Window)*;
- *контроль перегрузки* — регистрирует перегрузку канала и связанные с этим потери, а также понижает интенсивность трафика с помощью *Окна перегрузки (Congestion Window, CWnd)* и *Порога медленного старта (Slow Start Threshold, SSThreth)*.

В ns-2 поддерживает следующие TCP-агенты односторонней передачи:

- Agent/TCP
- Agent/TCP/Reno
- Agent/TCP/Newreno
- Agent/TCP/Sack1 — TCP с выборочным повтором (RFC2018)
- Agent/TCP/Vegas
- Agent/TCP/Fack — Reno TCP с «последующим подтверждением»
- Agent/TCP/Linux — TCP-передатчик с поддержкой SACK, который использует TCP с перезагрузкой контрольных модулей из ядра Linux

Односторонние агенты приёма:

- Agent/TCPSink
- Agent/TCPSink/DelAck
- Agent/TCPSink/Sack1
- Agent/TCPSink/Sack1/DelAck

Двунаправленный агент:

- Agent/TCP/FullTcp

TCP Tahoe:

- медленный старт (Slow-Start);
- контроль перегрузки (Congestion Avoidance);
- быстрый повтор передачи (Fast Retransmit);
- метод оценки длительности цикла передачи (Round Trip Time, RTT), используемой для установки таймера повторной передачи (Retransmission TimeOut, RTO).

Схема работы TCP Tahoe:

- при переполнении буфера все сегменты теряются;
- при потере сегмента или с наступлением таймаута запускается *процедура медленного старта* — потерянный пакет и все, посланные после него пакеты (вне зависимости от того, подтверждено их получение или нет) пересылаются повторно;
- контроль перегрузки и оценка RTT: окно перегрузки увеличивается на 1 пакет с каждым ACK, полученным в течение медленного старта (`cwnd_ < ssthresh_`), и увеличивается на `1/cwnd_` для каждого нового ACK, полученного при избежании перегрузки (когда `cwnd_ >= ssthresh_`);
- реакция на перегрузку: при получении трёх дублированных ACK устанавливается `ssthresh_=min(cwnd_,window_)/2`

TCP Reno:

- медленный старт (Slow-Start);
- контроль перегрузки (Congestion Avoidance);
- быстрый повтор передачи (Fast Retransmit);
- процедура быстрого восстановления (Fast Recovery);
- метод оценки длительности цикла передачи (Round Trip Time, RTT), используемой для установки таймера повторной передачи (Retransmission TimeOut, RTO).

Схема работы TCP Reno:

- размер окна увеличивается до тех пор, пока не произойдёт потеря сегмента (аналогично TCP Tahoe):
  - фаза медленного старта;
  - фаза избежания перегрузки;
- алгоритм не требует освобождения канала и его медленного (slow-start) заполнения после потери одного пакета;
- отправитель переходит в режим быстрого восстановления, после получения некоторого предельного числа дублирующих подтверждений — отправитель повторяет передачу одного пакета и уменьшает окно насыщения (cwnd) в два раза и устанавливает `ssthresh_` в соответствии с этим значением.

2.1.2. Мониторинг очередей

Объект мониторинга очереди оповещает диспетчера очереди о поступлении пакета. Диспетчер очереди осуществляет мониторинг очереди.

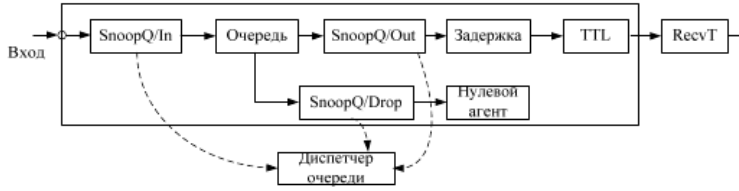


Рис. 2.3. Звено с объектами мониторинга очереди

SnoopQ/In — объект мониторинга очереди на входе.

SnoopQ/Out — объект мониторинга очереди на выходе.

SnoopQ/Drop — объект мониторинга отбрасываемых из очереди пакетов.

RecvT (receive tracing) — объект мониторинга принятых узлом пакетов.

Объекты очереди:

- `qlim_` — максимально разрешённое число пакетов в очереди;
- `limit_` — размер очереди в пакетах;
- `blocked_` — принимает значение true, если очередь заблокирована;
- `unblock_on_resume_` — принимает значение true, указывая, что очередь должна быть разблокирована после отправки последнего пакета;
- `bytes_` — принимает значение true, если используется режим передачи в байтах, а не в пакетах;
- `queue-in-bytes_` — принимает значение true, если используется режим измерения среднего размера очереди в байтах, а не пакетах;
- `thresh_` — минимальный порог среднего размера очереди (в пакетах);
- `maxthresh_` — максимальный порог среднего размера очереди (в пакетах);
- `mean_pktsize_` — грубая оценка среднего размера пакета (в байтах);
- `q_weight_` — вес очереди (используется при расчёте экспоненциально-взвешенного скользящего среднего размера очереди;
- `wait_` — интервал времени между сброшенными пакетами.

Объекты мониторинга очереди:

- `size_` — размер мгновенной длины очереди (в байтах);

- pkts_ — размер мгновенной длины очереди (в пакетах);
- parrivals_ — промежуточная сумма поступивших пакетов;
- barrivals_ — промежуточная сумма байт в поступивших пакетах
- pdepartures_ — промежуточная сумма обслуженных пакетов (не отброшенных);
- bdepartures_ — промежуточная сумма байт обслуженных пакетов (не отброшенных);
- pdrops_ — общая сумма отброшенных пакетов;
- bdrops_ — общая сумма байт отброшенных пакетов;
- bytesInt_ — заполненность очереди в байтах;
- pktsInt_ — заполненность очереди в пакетах;
- epdrops_ — число сброшенных по алгоритму RED пакетов;
- ebdrops_ — число байт в сброшенных по алгоритму RED пакетах;
- enable_in_ — устанавливается значение true, если требуется мониторинг потока на входе;
- enable_out_ — устанавливается значение true, если требуется мониторинг потока на выходе;
- enable_drop_ — устанавливается значение true, если требуется мониторинг сброшенных из потока пакетов;
- enable_edrop_ — устанавливается значение true, если требуется мониторинг сброшенных из потока пакетов по алгоритму RED;
- src_ — адрес источника пакетов, принадлежащих потоку;
- dst_ — адрес получателя пакетов, принадлежащих потоку;
- flowid_ — идентификатор потока.

```
# Пример задания множества объектов мониторинга:
SimpleLink instproc \
  attach-monitors { insnoop outsnoop dropsnoop qmon } {
   $self instvar queue_ head_ snoopIn_ snoopOut_ snoopDrop_
   $self instvar drophead_ qMonitor_
   set snoopIn_ $insnoop
   set snoopOut_ $outsnoop
   set snoopDrop_ $dropsnoop
   $snoopIn_ target $head_
   set head_ $snoopIn_
   $snoopOut_ target [$queue_ target]
   $queue_ target $snoopOut_
   $snoopDrop_ target [$drophead_ target]
   $drophead_ target $snoopDrop_
   $snoopIn_ set-monitor $qmon
   $snoopOut_ set-monitor $qmon
   $snoopDrop_ set-monitor $qmon
   set qMonitor_ $qmon
}
# Пример использования объектов мониторинга очереди соединения.
```

```
# Возвращает имя объекта, требуемого для определения
# среднего размера очереди
SimpleLink instproc init-monitor { ns qtrace sampleInterval} {
    $self instvar qMonitor_ ns_ qtrace_ sampleInterval_

    set ns_ $ns
    set qtrace_ $qtrace
    set sampleInterval_ $sampleInterval
    set qMonitor_ [new QueueMonitor]

    $self attach-monitors [new SnoopQueue/In]
        [new SnoopQueue/Out] [new SnoopQueue/Drop] $qMonitor_

    set bytesInt_ [new Integrator]
    $qMonitor_ set-bytes-integrator $bytesInt_
    set pktsInt_ [new Integrator]
    $qMonitor_ set-pkts-integrator $pktsInt_
    return $qMonitor_ }
```

2.2. Пример с дисциплиной RED

Постановка задачи Описание моделируемой сети:

- сеть состоит из 6 узлов;
- между всеми узлами установлено дуплексное соединение с различными пропускной способностью и задержкой 10 мс (см. рис. 2.4);
- узел r1 использует очередь с дисциплиной RED для накопления пакетов, максимальный размер которой составляет 25;
- TCP-источники на узлах s1 и s2 подключаются к TCP-приёмнику на узле s3;
- генераторы трафика ftp прикреплены к TCP-агентам.

На рис. 2.4 приведена схема моделируемой сети.

На рис. 2.5 приведена схема работы модуля RED.

На рис. 2.5 q — число пакетов в очереди, $\hat{q}$ — экспоненциально взвешенное скользящее среднее значение длины очереди, $p(\hat{q})$ — функция сброса пакетов.

Функция сброса алгоритма RED имеет вид (рис. 2.6):

$$p^{\mathrm{RED}}(\hat{q}) = \begin{cases} 0, & 0 < \hat{q} \leqslant q_{\min}, \\ \dfrac{\hat{q} - q_{\min}}{q_{\max} - q_{\min}} p_{\max}, & q_{\min} < \hat{q} \leqslant q_{\max}, \\ 1, & \hat{q} > q_{\max}, \end{cases}$$

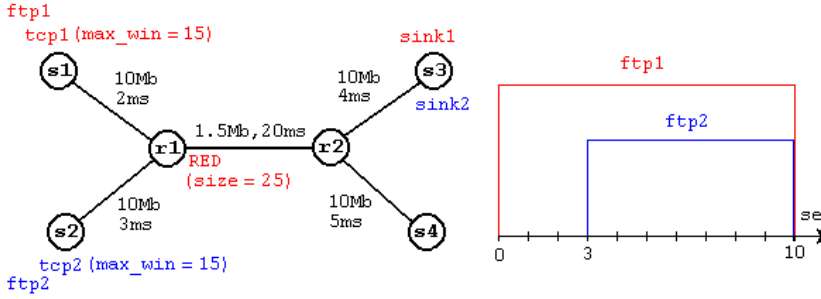


Рис. 2.4. Схема сети

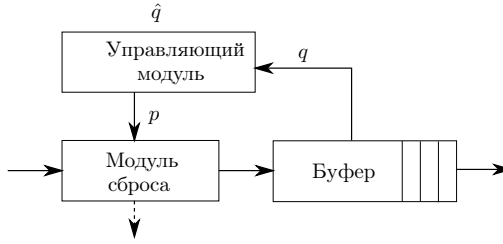


Рис. 2.5. Схема работы модуля RED

где $q_{\min}$, $q_{\max}$ — пороговые значения очереди; $p_{\max}$ — параметр максимального сброса.

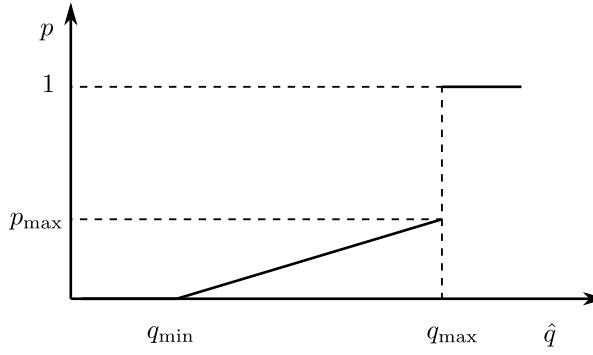


Рис. 2.6. Функция сброса алгоритма RED

Требуется разработать сценарий, реализующий модель согласно рис. 2.4, построить в Xgraph график изменения длины очереди и средней длины очереди.

Реализация модели

```
# Узлы сети:
for {set i 1} {$i < 5} {incr i} {
  set node_(s$i) [$ns node]
}
set node_(r1) [$ns node]
set node_(r2) [$ns node]

# Соединения:
$ns duplex-link $node_(s1) $node_(r1) 10Mb 2ms DropTail
$ns duplex-link $node_(s2) $node_(r1) 10Mb 3ms DropTail
$ns duplex-link $node_(r1) $node_(r2) 1.5Mb 20ms RED
$ns queue-limit $node_(r1) $node_(r2) 25
$ns queue-limit $node_(r2) $node_(r1) 25
$ns duplex-link $node_(s3) $node_(r2) 10Mb 4ms DropTail
$ns duplex-link $node_(s4) $node_(r2) 10Mb 5ms DropTail

# Агенты и приложения:
set tcp1 [$ns create-connection TCP/Reno
  $node_(s1) TCPSink $node_(s3) 0]
$tcp1 set window_ 15
set tcp2 [$ns create-connection TCP/Reno
  $node_(s2) TCPSink $node_(s3) 1]
$tcp2 set window_ 15
set ftp1 [$tcp1 attach-source FTP]
set ftp2 [$tcp2 attach-source FTP]
```

Здесь `window_` — верхняя граница окна приёмника (Advertisment Window) TCP соединения.

```
# Мониторинг очереди:
set redq [[$ns link $node_(r1) $node_(r2)] queue]
set tchan_ [open all.q w]
$redq trace curq_
$redq trace ave_
$redq attach $tchan_
```

Здесь `curq_` — текущий размер очереди, `ave_` — средний размер очереди.

```
# Добавление at-событий:
$ns at 0.0 "$ftp1 start"
```

```
$ns at 3.0 "$ftp2 start"
$ns at 10 "finish"

# Процедура finish:
proc finish {} {
 global tchan_

 # подключение кода AWK:
 set awkCode {
     {
         if ($1 == "Q" && NF>2) {
             print $2, $3 >> "temp.q";
             set end $2
         }
         else if ($1 == "a" && NF>2)
         print $2, $3 >> "temp.a";
     }
 }

 set f [open temp.queue w]
 puts $f "TitleText: red"
 puts $f "Device: Postscript"

 if { [info exists tchan_] } {
     close $tchan_
 }

 exec rm -f temp.q temp.a
 exec touch temp.a temp.q

 exec awk $awkCode all.q # выполнение кода AWK
 puts $f \"queue
 exec cat temp.q >@ $f
 puts $f \n\"ave_queue
 exec cat temp.a >@ $f
 close $f

 exec xgraph -bb -tk -x time -y queue temp.queue &
 exit 0
}
```

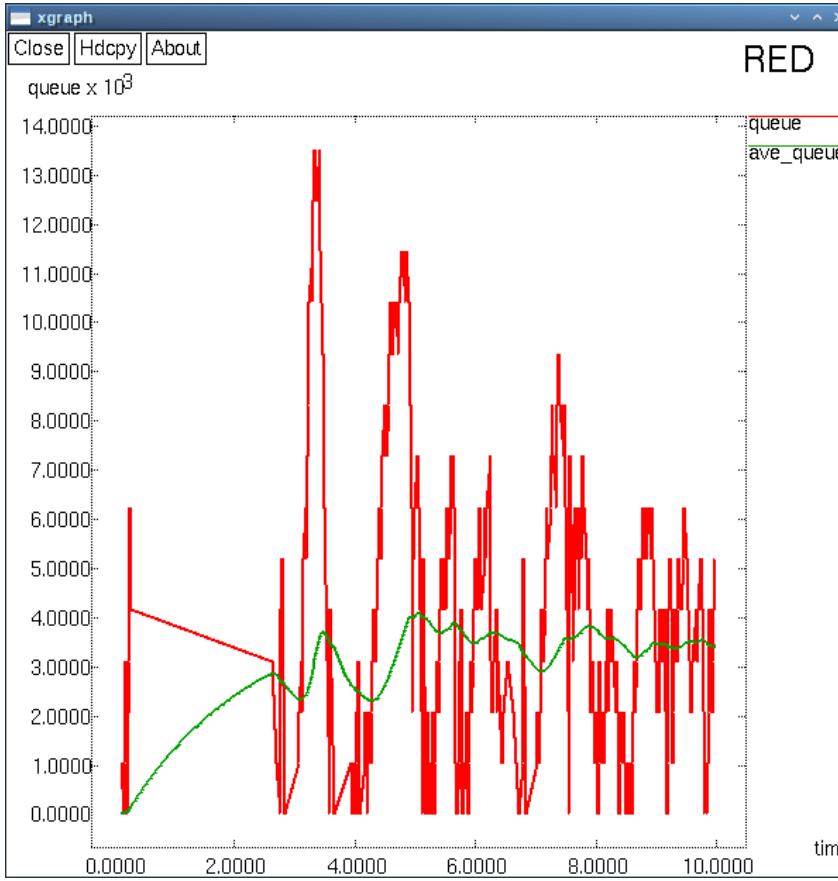


Рис. 2.7. График динамики длины очереди и средней длины очереди

Лабораторная работа 3. Моделирование стохастических процессов

В данном разделе использованы материалы [17].

3.1. Предварительные сведения. СМО $M|M|1$

$M|M|1$ — однолинейная СМО с накопителем бесконечной ёмкости. Поступающий поток заявок — пуассоновский с интенсивностью λ. Времена обслуживания заявок — независимые в совокупности случайные величины, распределённые по экспоненциальному закону с параметром μ.

Система дифференциальных уравнений Колмогорова:

$$\begin{cases} p_0'(t) = -\lambda p_0(t) + \mu p_1(t), \\ p_i'(t) = -(\lambda+\mu)p_i(t) + \lambda p_{i-1}(t) + \mu p_{i+1}(t), \quad i \geqslant 1, \end{cases} \tag{3.1}$$

$p_i(t) = P\{\nu(t) = i\}$ — вероятность того, что в момент времени t в системе находится i заявок.

Стационарные вероятности удовлетворяют СУР:

$$\begin{cases} 0 = -\lambda p_0 + \mu p_1, \\ 0 = -(\lambda+\mu)p_i + \lambda p_{i-1} + \mu p_{i+1}, \quad i \geqslant 1. \end{cases} \tag{3.2}$$

Уравнение локального баланса:

$$\lambda p_i = \mu p_{i+1}, \quad i \geqslant 0, \tag{3.3}$$

с условием нормировки: $\sum\limits_{i=0}^{\infty} p_i = 1$.

Решение уравнения (3.2):

$$\begin{cases} p_i = p_0 \rho^i, \quad i \geqslant 0, \quad \rho = \dfrac{\lambda}{\mu}, \\ p_0 = \dfrac{1}{\sum\limits_{i=0}^{\infty} \rho^i} = \left[\dfrac{1}{1-\rho}\right]^{-1} = 1-\rho, \quad \rho < 1 \end{cases} \tag{3.4}$$

Окончательно стационарное распределение числа заявок:

$$p_i = (1-\rho)\rho^i, \quad \rho < 1, \tag{3.5}$$

$\rho = \frac{\lambda}{\mu}$ — загрузка системы.

Характеристики системы:

Коэффициент использования системы: $u = 1 - p_0 = \rho$.

Стационарное среднее число заявок в системе: $N = \sum\limits_{i=0}^{\infty} i p_i = \dfrac{\rho}{1-\rho}$.

Стационарное среднее число заявок в очереди:

$$Q = \sum_{i=0}^{\infty} (i-1) p_i = \frac{\rho^2}{1-\rho}.$$

Стационарное среднее время ожидания начала обслуживания:

$$w = \frac{\rho}{\mu(1-\rho)}.$$

Стационарное среднее время пребывания заявки в системе:

$$v = \frac{1}{\mu(1-\rho)}.$$

3.2. Предварительные сведения. СМО $M|M|n|R$

$M|M|n|R$ — однолинейная СМО с накопителем конечной ёмкости R. Поступающий поток заявок — пуассоновский с интенсивностью λ. Времена обслуживания заявок — независимые в совокупности случайные величины, распределённые по экспоненциальному закону с параметром μ.

Система дифференциальных уравнений Колмогорова:

$$\begin{cases} p_0'(t) = -\lambda p_0(t) + \mu p_1(t), \\ p_i'(t) = -(\lambda + i\mu) p_i(t) + \lambda p_{i-1}(t) + (i+1)\mu p_{i+1}(t), \quad i = \overline{1, n-1}, \\ p_i'(t) = -(\lambda + n\mu) p_i(t) + \lambda p_{i-1}(t) + n\mu p_{i+1}(t), \quad i = \overline{n, n+R-1}, \\ p_{n+R}'(t) = -n\mu p_{n+R}(t) + \lambda p_{n+R-1}(t), \end{cases}$$

$p_i(t) = P\{\nu(t) = i\}$ — вероятность того, что в момент времени t в системе находится i заявок.

Стационарные вероятности удовлетворяют СУР:

$$\begin{cases} 0 = -\lambda p_0 + \mu p_1, \\ 0 = -(\lambda + i\mu)p_i + \lambda p_{i-1} + (i+1)\mu p_{i+1}, \quad i = \overline{1, n-1}, \\ 0 = -(\lambda + n\mu)p_i + \lambda p_{i-1} + n\mu p_{i+1}, \quad i = \overline{n, n+R-1}, \\ 0 = -n\mu p_{n+R} + \lambda p_{n+R-1}. \end{cases} \tag{3.6}$$

Уравнения локального баланса:

$$\begin{aligned} \lambda p_i &= (i+1)\mu p_{i+1}, \quad i = \overline{1, n-1}, \\ \lambda p_i &= n\mu p_{i+1}, \quad i = \overline{n, n+R-1} \end{aligned} \tag{3.7}$$

с условием нормировки: $\sum\limits_{i=0}^{\infty} p_i = 1$.

Решение уравнения (3.6):

$$\begin{cases} p_i = \dfrac{\rho^i}{i!} p_0, \quad i = \overline{1, n-1}, \\ p_i = \dfrac{\rho^i}{n! n^{i-n}} p_0, \quad i = \overline{n, n+R}, \\ p_0 = \left[\sum\limits_{i=0}^{n-1} \dfrac{\rho^i}{i!} + \dfrac{\rho^n}{n!} \dfrac{1 - \left(\frac{\rho}{n}\right)^{R+1}}{1 - \frac{\rho}{n}} \right]^{-1}, \end{cases} \tag{3.8}$$

$\rho = \dfrac{\lambda}{\mu}$ — загрузка системы.

При $n = 1$: $p_i = \dfrac{1-\rho}{1-\rho^{R+2}} \rho^i, \quad i = \overline{0, R+1}$.

Характеристики системы: Стационарная вероятность немедленного обслуживания заявки: $P_{w=0} = p_0 \sum\limits_{i=0}^{n-1} \dfrac{\rho^i}{i!}$.

Стационарная вероятность потери заявки: $\pi = p_{n+R} = \dfrac{\rho^{n+R}}{n! n^R} p_0$.

Стационарное среднее число заявок в очереди:

$$Q = \sum_{i=1}^{R} i p_{n+i} = \frac{\rho^{n-1}}{(n-1)!} \frac{1 + r\left(\frac{\rho}{n}\right)^{R+1} - (R+1)\left(\frac{\rho}{n}\right)^R}{\left(\frac{n}{\rho} - 1\right)^2} p_0.$$

3.3. Реализация модели на NS-2

```
# создание объекта  Simulator
set ns [new Simulator]

# открытие на запись файла out.tr для регистрации событий
set tf [open out.tr w]
$ns trace-all $tf

# задаём значения параметров системы
set lambda 30.0
set mu     33.0

# размер очереди для M|M|1 (для M|M|1|R: set qsize R)
set qsize     100000

# устанавливаем длительность эксперимента
set duration  1000.0

# задаём узлы и соединяем их симплексным соединением
# с полосой пропускания 100 Кб/с и задержкой 0 мс,
# очередью с обслуживанием типа  DropTail
set n1 [$ns node]
set n2 [$ns node]
set link [$ns simplex-link $n1 $n2 100kb 0ms DropTail]

# наложение ограничения на размер очереди:
$ns queue-limit $n1 $n2 $qsize

# задаём распределения интервалов времени
# поступления пакетов и размера пакетов
set InterArrivalTime [new RandomVariable/Exponential]
$InterArrivalTime set avg_ [expr 1/$lambda]
set pktSize [new RandomVariable/Exponential]
$pktSize set avg_ [expr 100000.0/(8*$mu)]

# задаём агент UDP и присоединяем его к источнику,
# задаём размер пакета
set src [new Agent/UDP]
$src set packetSize_ 100000
$ns attach-agent $n1 $src

# задаём агент-приёмник и присоединение его
set sink [new Agent/Null]
$ns attach-agent $n2 $sink
$ns connect $src $sink
```

```
# мониторинг очереди
set qmon [$ns monitor-queue $n1 $n2 [open qm.out w] 0.1]
$link queue-sample-timeout

# процедура finish закрывает файлы трассировки
proc finish {} {
  global ns tf
  $ns flush-trace
  close $tf
  exit 0
}

# процедура случайного генерирования пакетов
proc sendpacket {} {
  global ns src InterArrivalTime pktSize
  set time [$ns now]
 $ns at [expr $time +[$InterArrivalTime value]] "sendpacket"
  set bytes [expr round ([$pktSize value])]
  $src send $bytes
}

# планировщик событий
$ns at 0.0001 "sendpacket"
$ns at $duration "finish"

# расчет загрузки системы и вероятности потери пакетов
set rho [expr $lambda/$mu]
set ploss
 [expr (1-$rho)*pow($rho,$qsize)/(1-pow($rho,($qsize+1)))]
puts "Теоретическая вероятность потери = $ploss"
set aveq [expr $rho*$rho/(1-$rho)]
puts "Теоретическая средняя длина очереди = $aveq"

# запуск модели
$ns run
```

3.4. График в GNUplot (рис. 3.1)

```
#!/usr/bin/gnuplot -persist

# задаём текстовую кодировку,
# тип терминала, тип и размер шрифта
set encoding utf8
set term pdfcairo font "Arial,9"
```

```
# задаём выходной файл графика
set out 'qm.pdf'

# задаём название графика
set title "График средней длины очереди"

# задаём стиль линии
set style line 2

# подписи осей графика
set xlabel "t"
set ylabel "Пакеты"

# построение графика, используя значения
# 1-го и 5-го столбцов файла qm.out
plot "qm.out" using ($1):($5) with lines
        title "Размер очереди (в пакетах)",\
     "qm.out" using ($1):($5) smooth csplines
        title " Приближение сплайном ", \
     "qm.out" using ($1):($5) smooth bezier
        title " Приближение Безье "
```

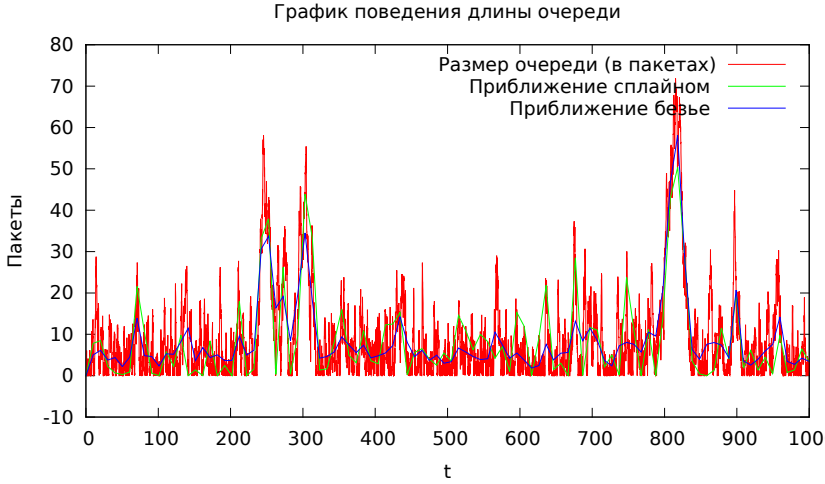


Рис. 3.1. График поведения длины очереди

Лабораторная работа 4. Задание для самостоятельного выполнения

4.1. Постановка задачи

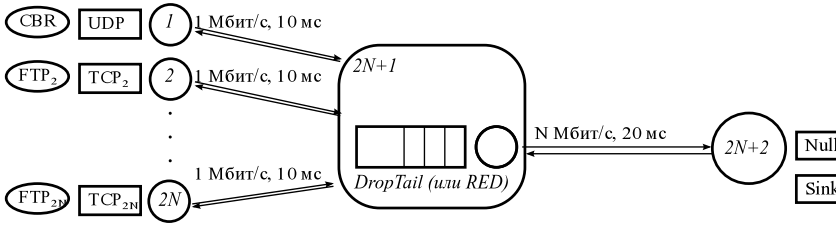


Рис. 4.1. Схема моделируемой сети

Описание моделируемой сети:
- сеть состоит из $2N + 2$ узлов;
- между узлами $1, 2, \ldots, 2N$ и узлом $2N + 1$ установлены дуплексные соединения с пропускной способностью 1 Мбит/с и задержкой 10 мс;
- между узлами $2N + 1$ и $2N + 2$ установлено дуплексное соединение с пропускной способностью N Мбит/с и задержкой 20 мс;
- узел $2N + 1$ использует очередь с дисциплиной DropTail (или RED) для накопления пакетов, максимальный размер которой составляет $10 \cdot N$ пакетов;
- TCP-источник на узлах $2, \ldots, 2N$ подключается к TCP-приёмнику на узле $2N + 2$;
- TCP-приёмник генерирует и отправляет ACK-пакеты отправителю и откидывает полученные пакеты;
- UDP-агент, который подсоединён к узлу 1, подключён к null-агенту на узле $2N + 2$;
- генераторы трафика FTP и CBR прикреплены к TCP и UDP агентам соответственно;
- генератор CBR генерирует пакеты размером 1 Кбайт со скоростью 1 Мбит/с.

4.2. Задание

1. Для приведённой схемы разработать имитационную модель в пакете NS-2.

2. Посчитать (с выводом значения на экран) число пакетов, поступивших в очередь с первого узла.
3. Посчитать (с выводом значения на экран) среднюю длину очереди и среднее время пребывания в очереди пакетов, поступивших с первого узла.
4. Построить график изменения длины очереди и средней длины очереди на узле $2N + 1$.
5. Оформить отчёт о выполненной работе.

Требования к отчёту

1. Отчёт должен быть аккуратно оформлен: иметь титульный лист с указанием идентифицирующих работу данных; содержать формулировку задачи; иметь единообразный шрифт (основной текст: 13 pt, Times NewRoman, 1,5 интервал, выравнивание по ширине; текст листингов: 10 Courier, 1 интервал; заголовки: 14 pt, Times NewRoman).
2. В отчёт включаются описания выполнения всех лабораторных работ раздела и заданий для самостоятельного выполнения.
3. Отчёт должен содержать листинг разработанной программы с пояснениями на русском языке, скриншоты nam с пояснением, что на них изображено, полученные в результате моделирования графики и/или скриншоты консоли экрана с посчитанными значениями.

Часть II
Компонентное моделирование. Scilab, подсистема xcos

II.1. Теоретические сведения

II.1.1. Общее описание Scilab и xcos

Scilab — система компьютерной математики, предназначенная для решения вычислительных задач.

Основное окно Scilab содержит *обозреватель файлов*, *командное окно*, *обозреватель переменных* и *журнал команд* (рис. II.1.1).

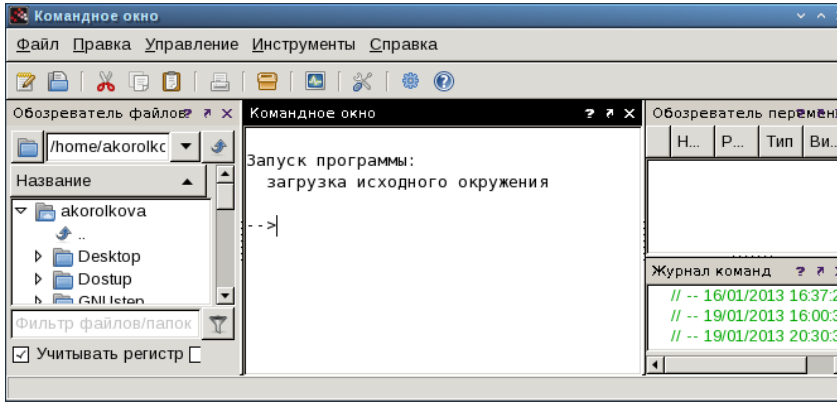


Рис. II.1.1. Основное окно Scilab

Программа *xcos* является приложением к пакету Scilab. Для вызова окна xcos необходимо в меню основного окна Scilab выбрать *Инструменты*, *Визуальное моделирование xcos*.

При моделировании с использованием xcos реализуется принцип визуального программирования, в соответствии с которым пользователь на экране из *палитры блоков* (рис. II.1.2) создаёт модель и осуществляет расчёты.

На рис. II.1.3 в качестве примера приведена модель функционирования двух источников синусоидального сигнала, позволяющая в зависимости от задаваемых параметров построить различные фигуры Лиссажу.

Математическое выражение для кривой Лиссажу:

$$\begin{cases} x(t) = A\sin(at + \delta), \\ y(t) = B\sin(bt), \end{cases}$$

где A, B — амплитуды колебаний, a, b — частоты, δ — сдвиг фаз.

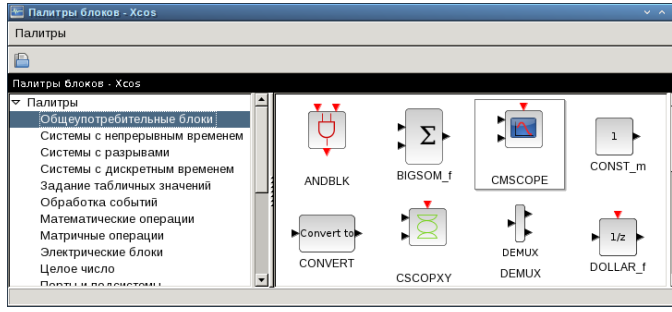


Рис. II.1.2. Палитры в xcos

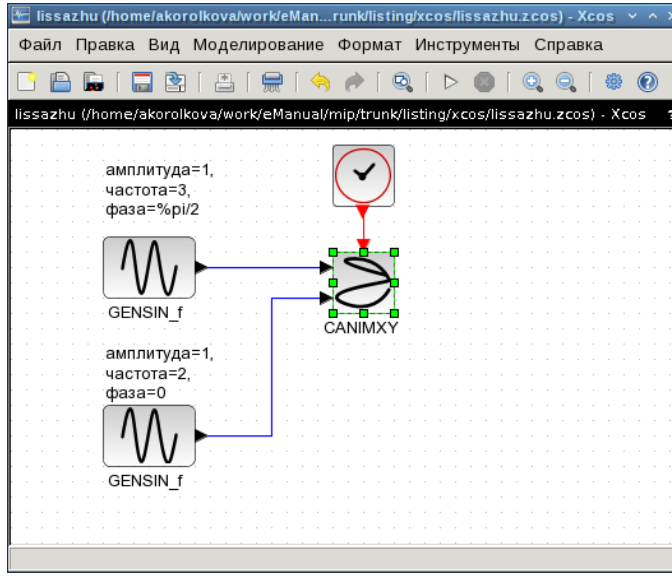


Рис. II.1.3. Пример модели в xcos

В модели, изображённой на рис. II.1.3, использованы следующие блоки xcos:

- `CLOCK_c` — запуск часов модельного времени;
- `GENSIN_f` — блок генератора синусоидального сигнала;
- `CANIMXY` — анимированное регистрирующее устройство для построения графика типа $y = f(x)$;
- `TEXT_f` — задаёт текст примечаний.

Предположим, что в модели заданы следующие параметры: $A = B = 1$, $a = 3$, $b = 2$, $\delta = \pi/2$. Получим график, изображённый на рис. II.1.4.

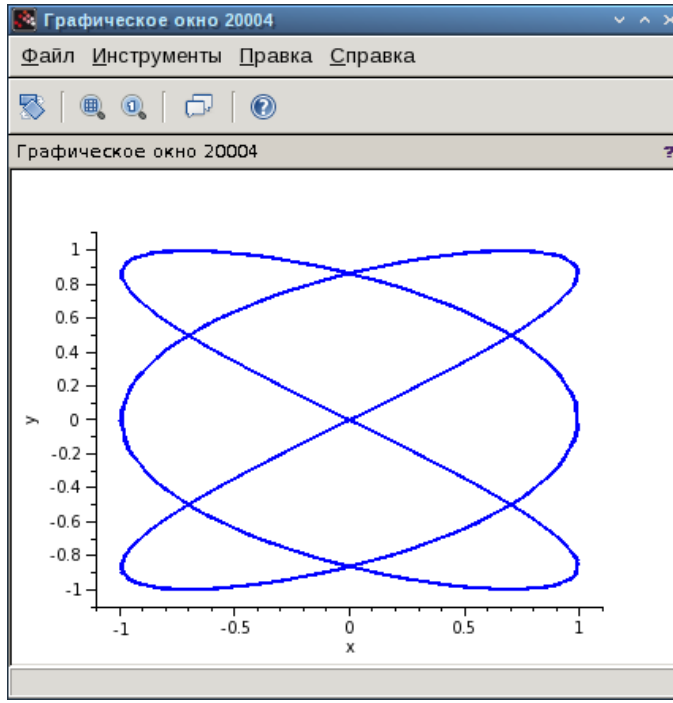


Рис. II.1.4. Фигура Лиссажу: $A = B = 1$, $a = 3$, $b = 2$, $\delta = \pi/2$

II.1.2. Упражнение

Постройте с помощью xcos фигуры Лиссажу со следующими параметрами:

1) $A = B = 1$, $a = 2$, $b = 2$, $\delta = 0;\ \pi/4;\ \pi/2;\ 3\pi/4;\ \pi$;
2) $A = B = 1$, $a = 2$, $b = 4$, $\delta = 0;\ \pi/4;\ \pi/2;\ 3\pi/4;\ \pi$;
3) $A = B = 1$, $a = 2$, $b = 6$, $\delta = 0;\ \pi/4;\ \pi/2;\ 3\pi/4;\ \pi$;
4) $A = B = 1$, $a = 2$, $b = 3$, $\delta = 0;\ \pi/4;\ \pi/2;\ 3\pi/4;\ \pi$.

II.1.3. Modelica

Modelica – свободно распространяемый объектно-ориентированный язык для моделирования сложных физических систем. В основе языка Modelica лежит концепция соединяемых блоков. При соединении в соответствии с требуемой схемой автоматически генерируются соответствующие уравнения.

Язык Modelica в чем-то похож на императивные объектно-ориентированные языки. В нём есть выражения, классы, наследование, функции. В основу языка положена конструкция «уравнение» (equation).

Общая структура класса решения некоторого уравнения на языке Modelica:

```
class имя_класса
   объявления;
   equation
   уравнение_1;
   ...
   уравнение_n;
end имя_класса;
```

Дифференциальное уравнение типа $\dot{x} = -x$ на языке Modelica описывается следующим кодом:

```
class DU "Решение ДУ"
   Real x(start=1);
   equation
   \der(x)=-x;
end DU;
```

Здесь DU — название объявляемого нами класса решения дифференциального уравнения, что отражено в соответствующей записи комментария. Далее объявлена переменная типа `Real` с именем `x` и её начальное значение. Ключевое слово `equation` начинает секцию объявления уравнений. Затем задано решаемое дифференциальное уравнение.

Документацию по языку Modelica можно найти на сайте разработчика: `https://www.modelica.org/`.

Лабораторная работа 5. Модель эпидемии (SIR)

5.1. Математическая модель

Модель SIR предложена в 1927 г. (W. O. Kermack, A. G. McKendrick).

Предполагается, что особи популяции размера N могут находиться в трёх различных состояниях:

- S (susceptible, уязвимые) — здоровые особи, которые находятся в группе риска и могут подхватить инфекцию;
- I (infective, заражённые, распространяющие заболевание) — заразившиеся переносчики болезни;
- R (recovered/removed, вылечившиеся) — те, кто выздоровел и перестал распространять болезнь (в эту категорию относят, например, приобретших иммунитет или умерших).

Внутри каждой из выделенных групп особи считаются неразличимыми по свойствам. Типичная эволюция особи популяции описывается следующей диаграммой:

$$S \to I \to R.$$

Считаем, что система замкнута, т.е.

$$N = S + I + R.$$

Если предположить, что каждый член популяции может контактировать с каждым, то задача о распространении эпидемии описывается системой дифференциальных уравнений:

$$\begin{cases} \dot{s} = -\beta s(t) i(t); \\ \dot{i} = \beta s(t) i(t) - \nu i(t); \\ \dot{r} = \nu i(t). \end{cases} \tag{5.1}$$

где β — скорость заражения, ν — скорость выздоровления.

В представленной модели сделано предположение, что все особи популяции могут равновероятно заразиться со скоростью β.

Первое уравнение описывает динамику численности уязвимых к болезни особей: заражённая особь с некоторой скоростью β заражает уязвимую особь. Таким образом, инфицированная особь вступает в контакт и может передавать болезнь другим со скоростью βN. При этом доля контактов с инфицированными составляет

$s(t)/N$. Число новых инфицированных в единицу времени составляет $\beta N(s(t)/N)i(t) = \beta s(t)i(t)$.

Третье уравнение описывает динамику выздоровления заражённой особи: с некоторой скоростью ν инфицированная особь выздоравливает.

Второе уравнение описывает динамику численности заражённых особей: разность числа заражённых особей и числа выздоровевших особей.

5.2. Реализация модели в xcos

Зафиксируем начальные данные: $\beta = -1$, $\nu = 0,3$, $s(0) = 0,999$, $i(0) = 0,001$, $r(0) = 0$.

В меню *Моделирование*, *Задать переменные окружения* зададим значения переменных β и ν (рис. 5.1).

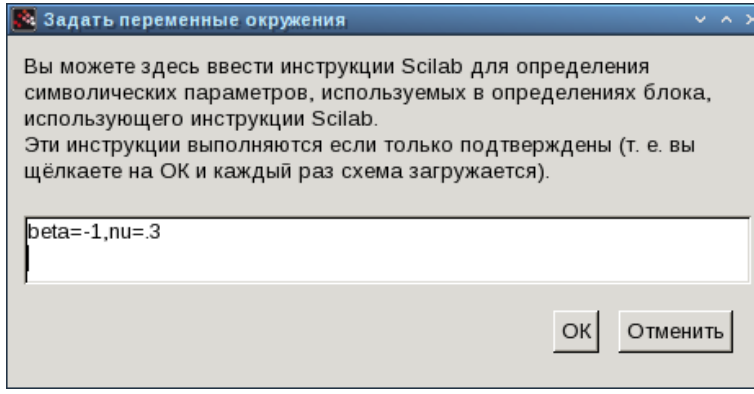


Рис. 5.1. Задать переменные окружения в xcos

Для реализации модели (5.1) потребуются следующие блоки xcos:

- `CLOCK_c` — запуск часов модельного времени;
- `CSCOPE` — регистрирующее устройство для построения графика;
- `TEXT_f` — задаёт текст примечаний;
- `MUX` — мультиплексер, позволяющий в данном случае вывести на графике сразу несколько кривых;
- `INTEGRAL_m` — блок интегрирования: $y(t) = \int\limits_{t_0}^{t} u(t)dt + y_0$;
- `GAINBLK_f` — в данном случае позволяет задать значения коэффициентов β и ν;
- `SUMMATION` — блок суммирования;

– `PROD_f` — поэлементное произведение двух векторов на входе блока.

Готовая модель SIR представлена на рис. 5.2.

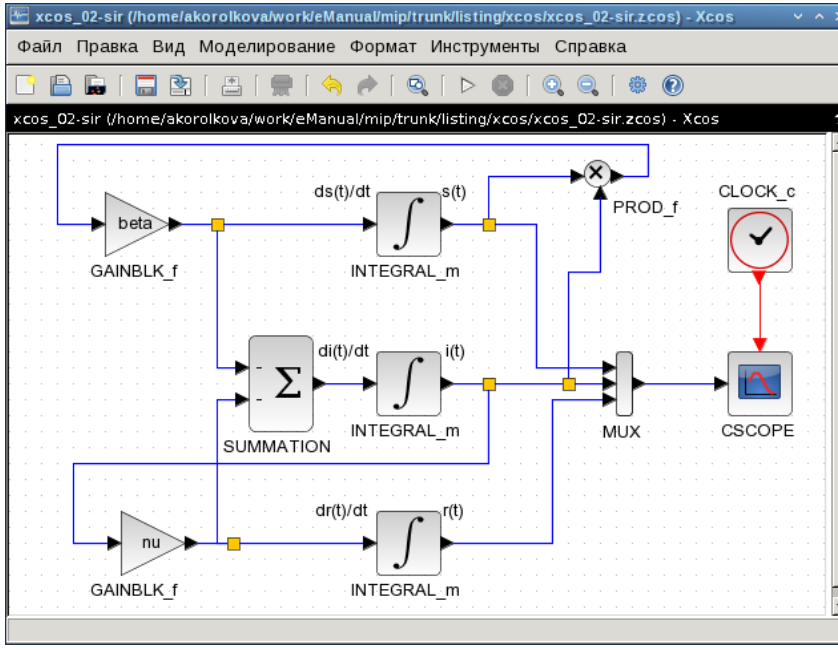


Рис. 5.2. Модель SIR в xcos

Первое уравнение модели (5.1) задано верхним блоком интегрирования, блоком произведения и блоком задания коэффициента β. Блок произведения соединён с выходами верхнего и среднего блоков интегрирования и блоком коэффициента β, что реализует математическую конструкцию $s(t)i(t)\beta$.

Третье уравнение модели (5.1) задано нижним блоком интегрирования и блоком задания коэффициента ν. Для реализации математической конструкции $\nu i(t)$ соединяем выход среднего блока интегрирования и вход блока задания коэффициента ν, а результат передаём на вход нижнего блока интегрирования.

Средний блок интегрирования и блок суммирования определяют второе уравнение модели (5.1), которое по сути является суммой правых частей первого и третьего уравнений (5.1). Для реализации соединяем входы верхнего и нижнего блоков интегрирования с входами блока суммирования, меняя при этом в его параметрах оба знака на

минус. Выход блока суммирования соединяем с входом среднего блока интегрирования.

Выходы трёх блоков интегрирования соединяем с мультиплексором.

В параметрах верхнего и среднего блока интегрирования необходимо задать начальные значения $s(0) = 0,999$ и $i(0) = 0,001$ (рис. 5.3).

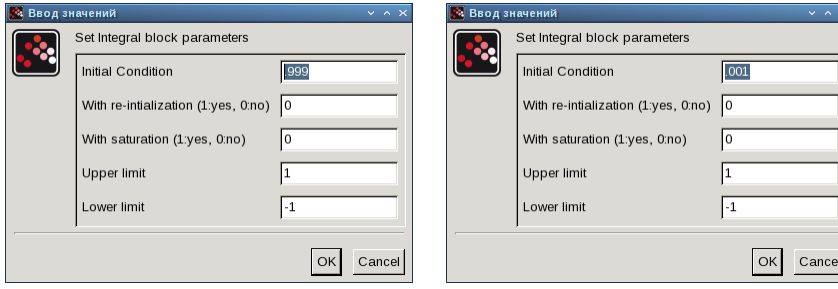


Рис. 5.3. Задать начальные значения в блоках интегрирования

В меню *Моделирование*, *Установка* необходимо задать *конечное время интегрирования*, равным времени моделирования (в данном случае 30, см. рис. 5.4).

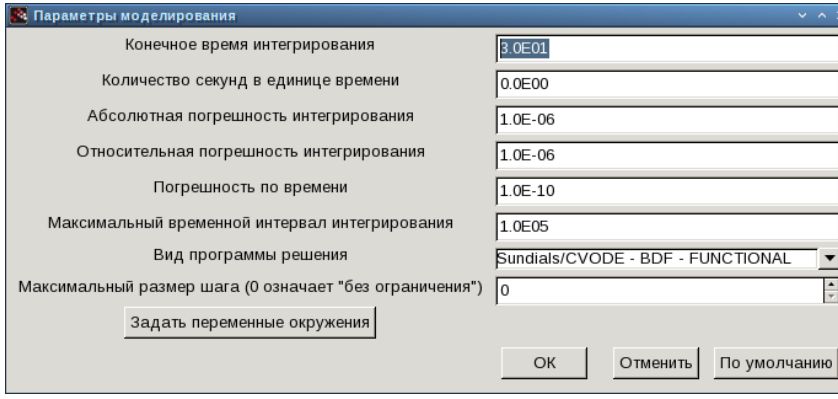


Рис. 5.4. Задать конечное время интегрирования в xcos

Результат моделирования представлен на рис. 5.5, где сплошной линией обозначен график $s(t)$ (динамика численности уязвимых к болезни особей), пунктирная линия определяет $r(t)$ — динамику численности выздоровевших особей, наконец, пунктирная с точкой линия

определяет $i(t)$ — динамику численности заражённых особей. Пересечение трёх линий определяет порог эпидемии.

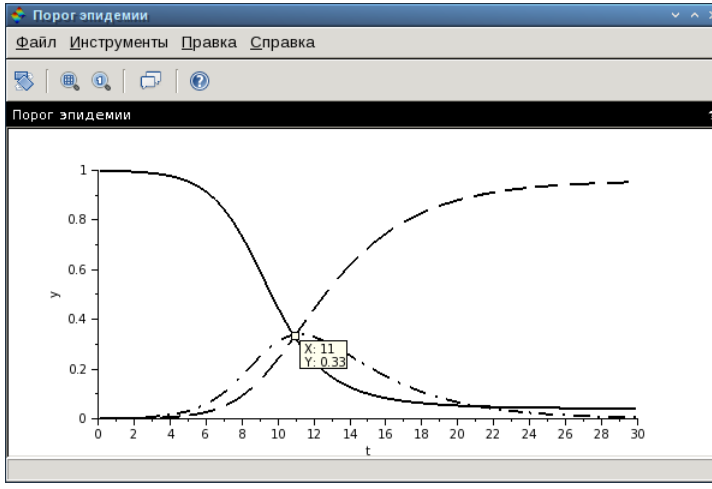


Рис. 5.5. Эпидемический порог модели SIR 5.1 при $\beta = 1$, $\nu = 0.3$

5.3. Реализация модели с помощью блока Modelica в xcos

Готовая модель SIR представлена на рис. 5.6.

Для реализации модели (5.1) с помощью языка Modelica помимо блоков `CLOCK_c`, `CSCOPE`, `TEXT_f` и `MUX` требуются блоки `CONST_m` — задаёт константу; `MBLOCK` (Modelica generic) — блок реализации кода на языке Modelica. Задаём значения переменных β и ν (см. рис. 5.1).

Параметры блока Modelica представлены на рис. 5.7. Переменные на входе (“beta”, “nu”) и выходе (“s”, “i”, “r”) блока заданы как внешние (“E”).

Код на языке Modelica:

```
class generic
////automatically generated ////
    //input variables
    Real beta,nu;
    //output variables  (комментируем, т.к.
    // начальные значения задаем в самом блоке):
    // Real s,i,r;
////do not modif above this line ////
```

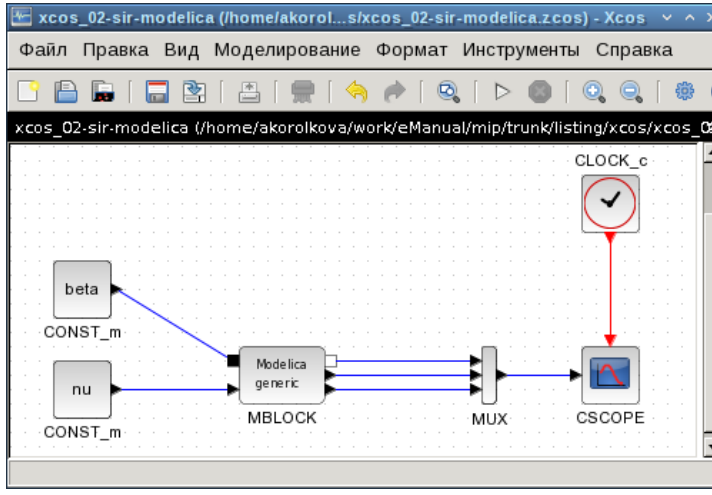


Рис. 5.6. Модель SIR в xcos с применением блока Modelica

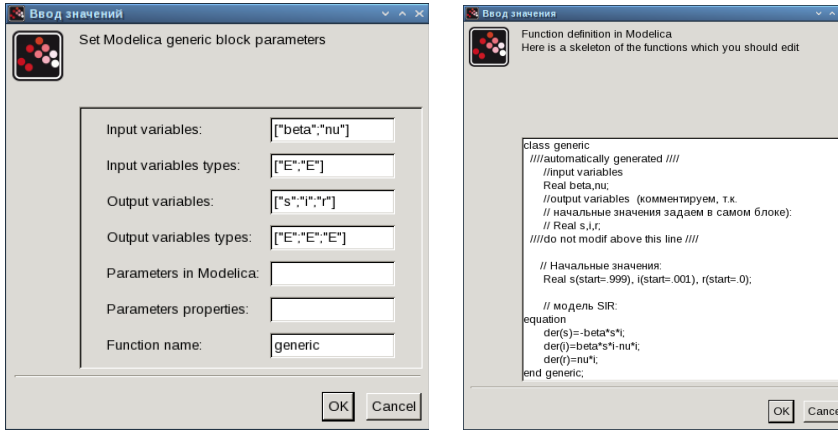


Рис. 5.7. Параметры блока Modelica для модели (5.1)

```
    // Начальные значения:
     Real s(start=.999), i(start=.001), r(start=.0);
     // модель SIR:
equation
     der(s)=-beta*s*i;
```

```
    der(i)=beta*s*i-nu*i;
    der(r)=nu*i;
end generic;
```

Результат моделирования совпадёт с рис. 5.5.

5.4. Задание для самостоятельного выполнения

Реализовать модель SIR с учётом процесса рождения / гибели особей:

$$\begin{cases} \dot{s} = -\beta s(t)i(t) + \mu(N - s(t)); \\ \dot{i} = \beta s(t)i(t) - \nu i(t) - \mu i(t); \\ \dot{r} = \nu i(t) - \mu r(t), \end{cases} \tag{5.2}$$

где μ — скорость умирания особей (предполагается равной скорости рождения особей).

Построить графики эпидемического порога при различных значениях параметров модели. Сделать анализ полученных графиков в зависимости от выбранных значений параметров модели.

Лабораторная работа 6. Модель «хищник–жертва»

6.1. Математическая модель

Модель «хищник–жертва» (модель Лотки — Вольтерра) представляет собой модель межвидовой конкуренции. В математической форме модель имеет вид:

$$\begin{cases} \dot{x} = ax - bxy; \\ \dot{y} = cxy - dy, \end{cases} \tag{6.1}$$

где x — количество жертв; y — количество хищников; a, b, c, d — коэффициенты, отражающие взаимодействия между видами: a — коэффициент рождаемости жертв; b — коэффициент убыли жертв; c — коэффициент рождения хищников; d — коэффициент убыли хищников.

6.2. Реализация модели в xcos

Зафиксируем начальные данные: $a = 2$, $b = 1$, $c = 0,3$, $d = 1$, $x(0) = 2$, $y(0) = 1$.

В меню *Моделирование*, *Задать переменные окружения* зададим значения коэффициентов a, b, c, d (рис. 6.1).

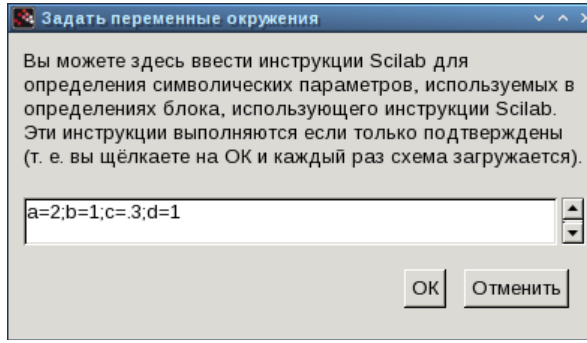


Рис. 6.1. Задать переменные окружения в xcos для модели (6.1)

Для реализации модели (6.1) в дополнение к блокам `CLOCK_c`, `CSCOPE`, `TEXT_f`, `MUX`, `INTEGRAL_m`, `GAINBLK_f`, `SUMMATION`, `PROD_f` по-

требуется блок CSCOPXY — регистрирующее устройство для построения фазового портрета.

Готовая модель «хищник–жертва» представлена на рис. 6.2.

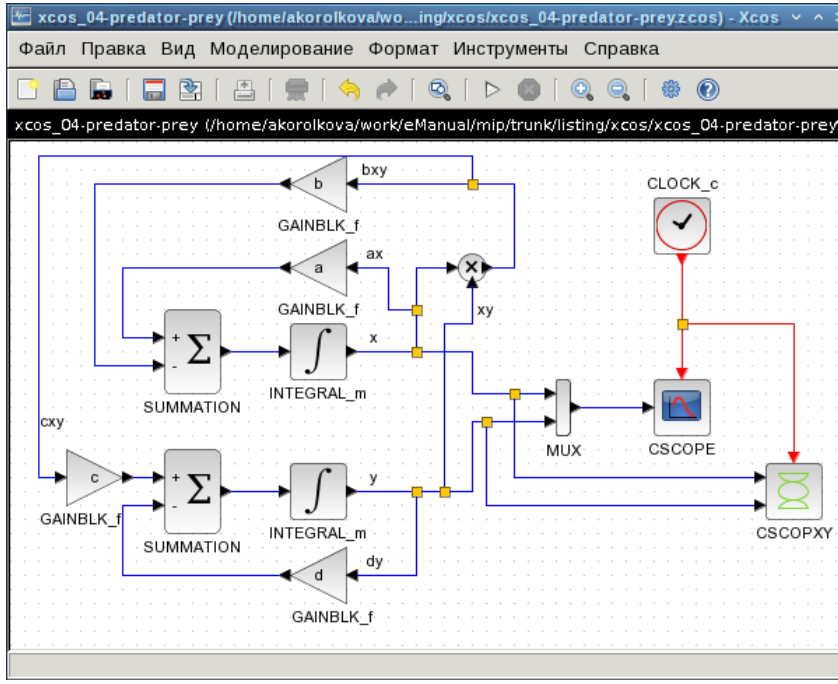


Рис. 6.2. Модель «хищник–жертва» в xcos

Первое уравнение модели (6.1) задано верхним блоком интегрирования, блоком произведения и блоками задания коэффициентов a и b.

Второе уравнение модели (6.1) задано нижним блоком интегрирования и блоками задания коэффициентов c и d.

Для суммирования слагаемых правых частей уравнений (6.1) используем блоки суммирования с соответствующими знаками перед коэффициентами. Выходы блоков суммирования соединяем с входами блоков интегрирования. Выходы блоков интегрирования соединяем с мультиплексором, который в свою очередь позволяет вывести на один график сразу обе кривые: динамику численности жертв и динамику численности хищников.

В параметрах блоков интегрирования необходимо задать начальные значения $x(0) = 2$, $y(0) = 1$ (рис. 6.3)

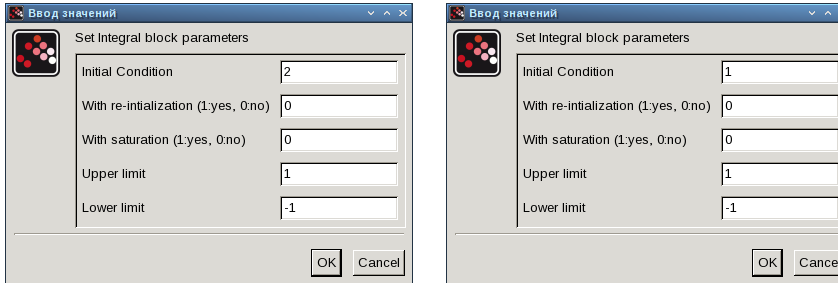


Рис. 6.3. Задать начальные значения в блоках интегрирования

В меню *Моделирование*, *Установка* необходимо задать *конечное время интегрирования*, равным времени моделирования: 30.

Результат моделирования представлен на рис. 6.4.

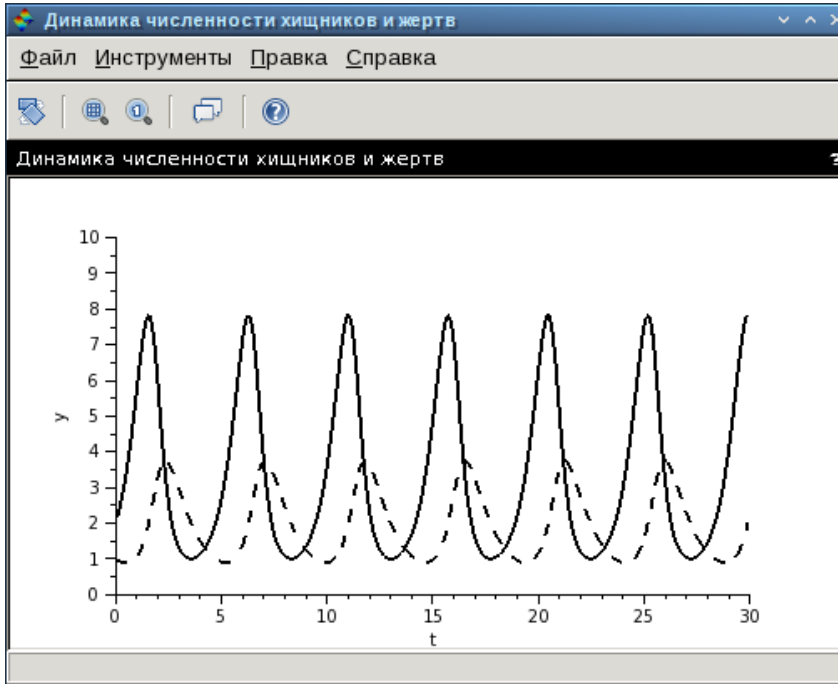


Рис. 6.4. Динамика изменения численности хищников и жертв модели 6.1 при $a = 2$, $b = 1$, $c = 0,3$, $d = 1$, $x(0) = 2$, $y(0) = 1$

На рис. 6.4 сплошной линией обозначен график $x(t)$ (динамика численности жертв), пунктирная линия определяет $y(t)$ — динамику численности хищников.

На рис. 6.5 приведён фазовый портрет модели 6.1.

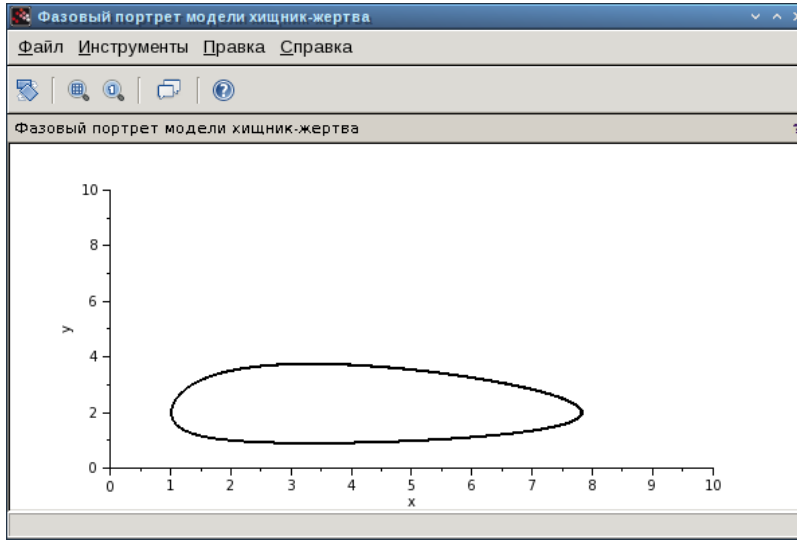


Рис. 6.5. Фазовый портрет модели 6.1 при $a = 2$, $b = 1$, $c = 0,3$, $d = 1$, $x(0) = 2$, $y(0) = 1$

Фазовый портрет — графическое изображение системы на фазовой плоскости (или в многомерном пространстве), по координатным осям которого отложены значения величин переменных системы. Поведение переменных во времени при таком способе представления для каждой начальной точки описывается фазовой траекторией. Совокупность таких фазовых траекторий для любых начальных условий представляет собой фазовый портрет.

6.3. Реализация модели с помощью блока Modelica в xcos

Для реализации модели (6.1) с помощью языка Modelica потребуются следующие блоки xcos: `CLOCK_c`, `CSCOPE`, `CSCOPXY`, `TEXT_f`, `MUX`, `CONST_m` и `MBLOCK` (Modelica generic).

Как и ранее, задаём значения коэффициентов a, b, c, d (см. рис. 6.1).

Готовая модель «хищник–жертва» представлена на рис. 6.6.

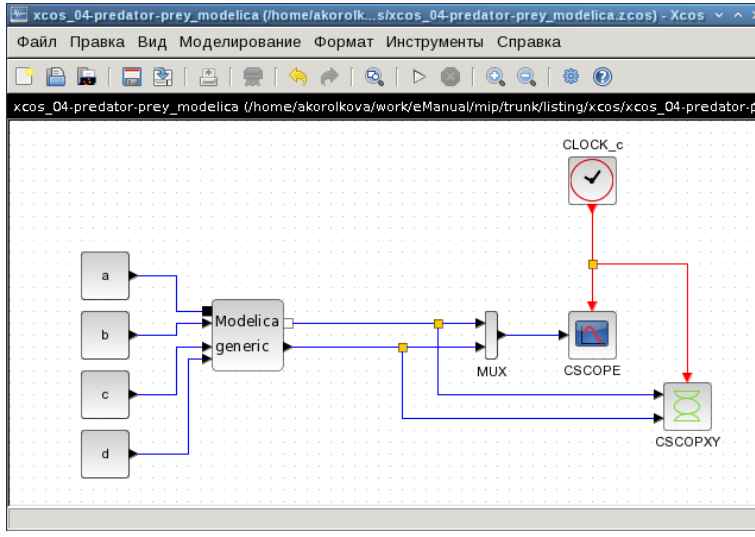


Рис. 6.6. Модель «хищник–жертва» в xcos с применением блока Modelica

Параметры блока Modelica представлены на рис. 6.7. Переменные на входе (“a”, “b”, “c”, “d”) и выходе (“x”, “y”) блока заданы как внешние (“E”).

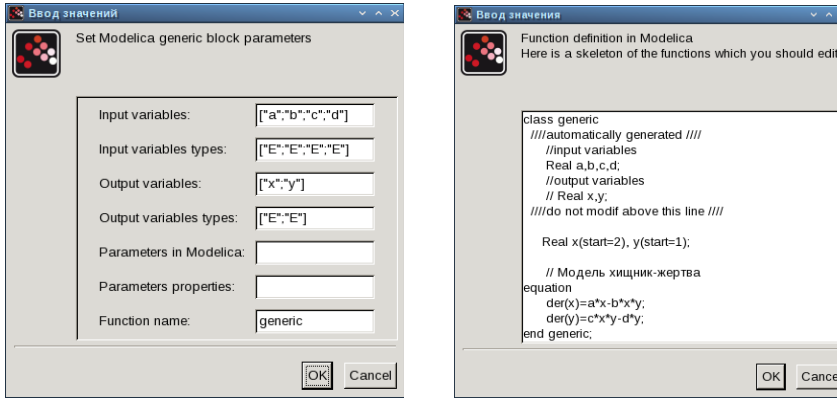


Рис. 6.7. Параметры блока Modelica для модели (6.1)

Результат моделирования совпадёт с рис. 6.4 и 6.5.

Код на языке Modelica:

```
class generic
////automatically generated ////
    //input variables
    Real a,b,c,d;
    //output variables
    // Real x,y;
////do not modif above this line ////

   Real x(start=2), y(start=1);
// Модель хищник-жертва
equation
    der(x)=a*x-b*x*y;
    der(y)=c*x*y-d*y;
end generic;
```

Лабораторная работа 7. Модель $M|M|1|\infty$

Рассмотрим пример моделирования в xcos системы массового обслуживания типа $M|M|1|\infty$.

Зафиксируем начальные данные: $\lambda = 0,3$, $\mu = 0,35$, $z_0 = 6$. Суперблок, моделирующий поступление заявок, представлен на рис. 7.1.

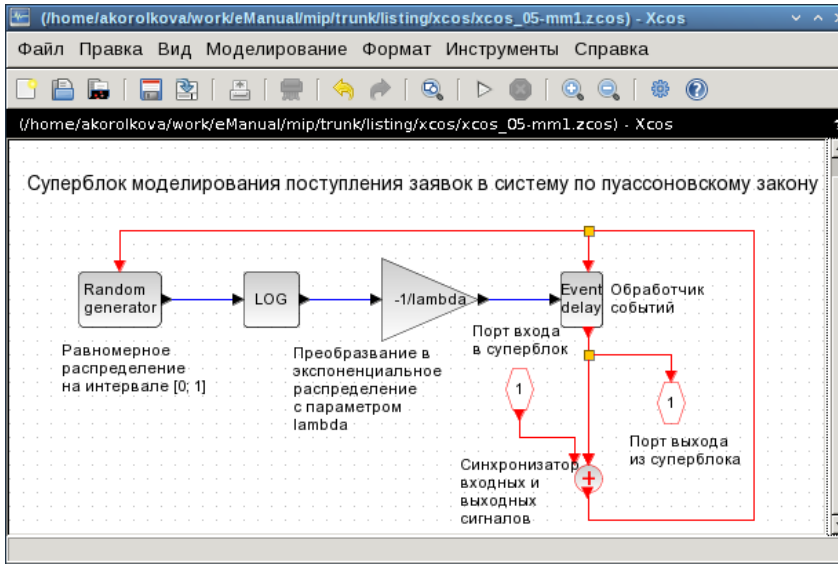


Рис. 7.1. Суперблок, моделирующий поступление заявок

Суперблок, моделирующий процесс обработки заявок, представлен на рис. 7.2.

Готовая модель $M|M|1|\infty$ представлена на рис. 7.3.

Результат моделирования представлен на рис. 7.4 и 7.5.

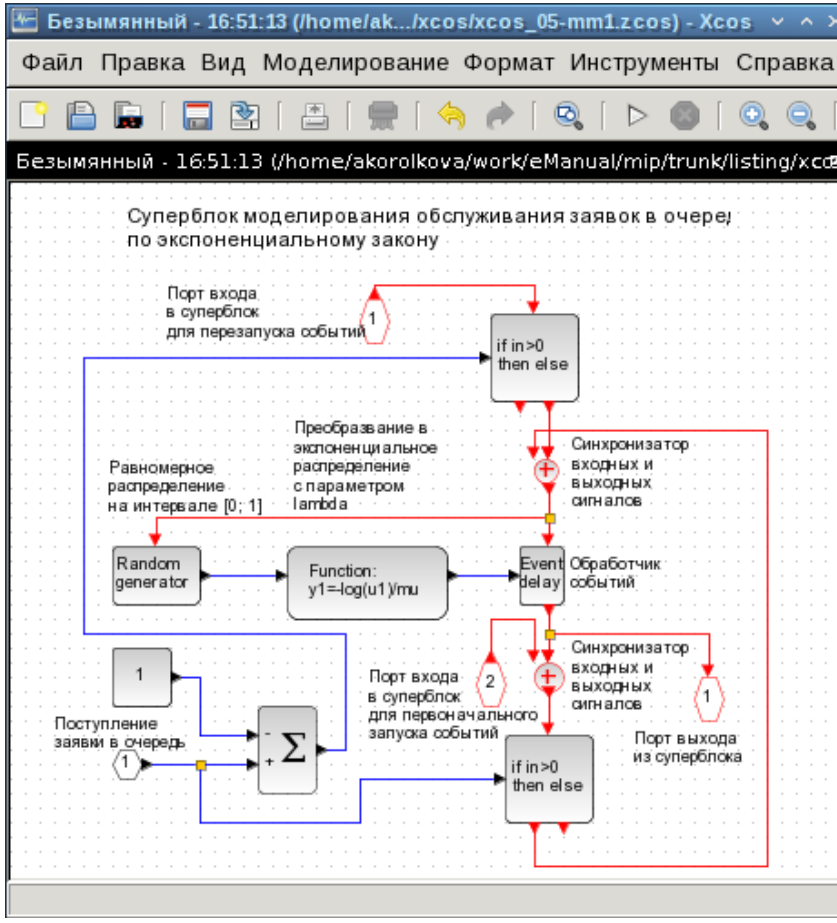


Рис. 7.2. Суперблок, моделирующий обработку заявок

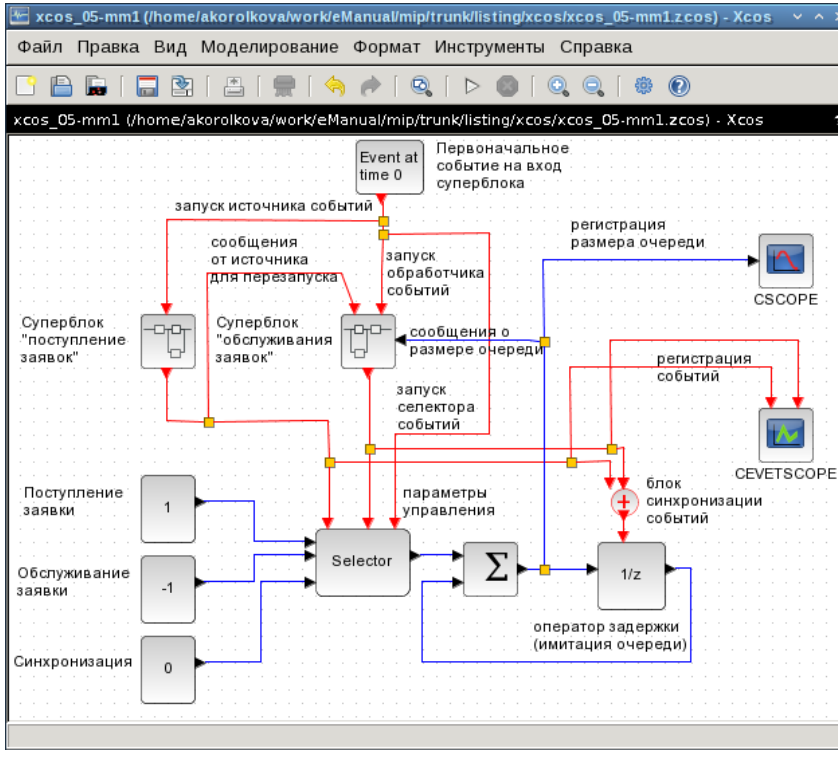


Рис. 7.3. Модель $M|M|1|\infty$ в xcos

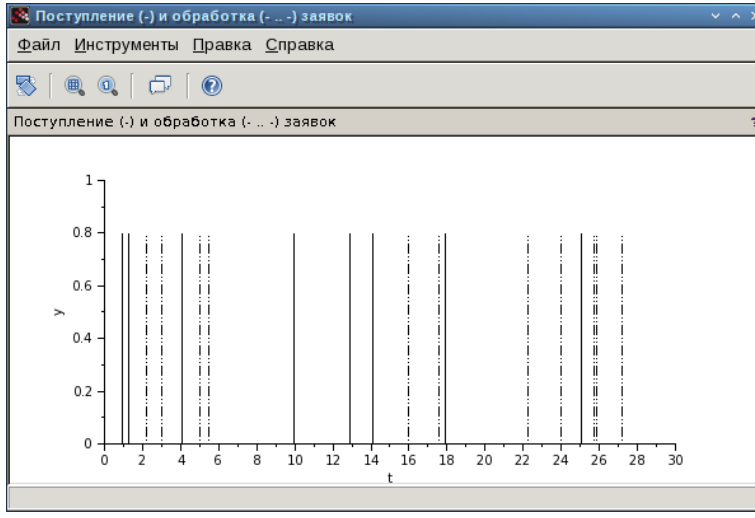


Рис. 7.4. Поступление (—) и обработка (– · · –) заявок

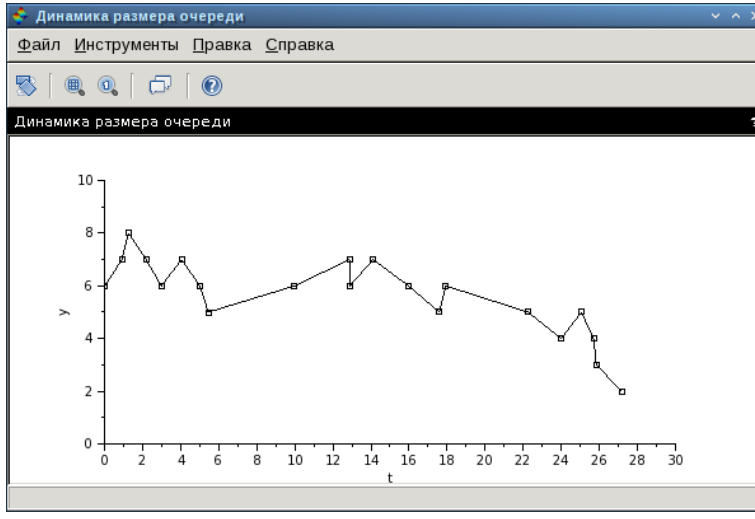


Рис. 7.5. Динамика размера очереди

Лабораторная работа 8. Модель TCP/AQM

8.1. Математическая модель

Рассмотрим упрощённую модель поведения TCP-подобного трафика с регулируемой некоторым AQM алгоритмом динамической интенсивностью потока:

$$\dot{W}(t) = \frac{1}{R(t)} - \frac{1}{2}\frac{W(t)W(t-R(t))}{R(t-R(t))}p(t-R(t)), \tag{8.1}$$

$$\dot{Q}(t) = \begin{cases} N(t)\dfrac{W(t)}{R(t)} - C, & Q(t) > 0, \\ \max\left(N(t)\dfrac{W(t)}{R(t)} - C, 0\right), & Q(t) = 0, \end{cases} \tag{8.2}$$

где $W(t)$ — средний размер TCP-окна (в пакетах), $Q(t)$ — средний размер очереди (в пакетах), $R(t)$ — время двойного оборота (Round Trip Time, сек.), C — скорость обработки пакетов в очереди (пакетов в секунду), $N(t)$ — число TCP-сессий, $p(\cdot)$ — вероятностная функция сброса (отметки на сброс) пакета (значения функции $p(\cdot)$ лежат на интервале $[0, 1]$).

Функции $W(t)$ и $Q(t)$ — положительны. Функция $R(t)$ может быть представлена в виде

$$R(t) = \frac{Q(t)}{C} + \tau_p, \tag{8.3}$$

где τ_p — задержка распространения пакета по сети (сек.).

Уравнение (8.1) описывает динамическое управление размером окна TCP. Первое слагаемое описывает фазу медленного старта TCP. Второе слагаемое учитывает фазу и алгоритм избежания перегрузок. На фазе избежания перегрузок размер окна увеличивается на $1/W$ при получении каждого подтверждения, а в случае потери пакета размер окна сокращается вдвое.

Уравнение (8.2) описывает поведение очереди, а именно разность средней интенсивности поступления пакетов $\dfrac{N(t)W(t)}{R(t)}$ и пропускной способностью звена сети C.

Жидкостную модель (8.1)–(8.3) управления потоком трафика принято называть задачей управления с обратной связью. Управление в такой задаче описывается вероятностной функцией $p(\cdot)$.

Сделаем упрощение модели, приняв, что $N(t) \equiv N$, $R(t) \equiv R$, т.е. указанные величины будем считать постоянными, не изменяющимися во времени. Кроме того, положим $p(\cdot) = KQ(t)$, т.е. функция сброса пакетов $p(\cdot)$ пропорциональна длине очереди $Q(t)$.

В результате получим следующую упрощённую модель управления TCP-подобным трафиком:

$$\dot{W}(t) = \frac{1}{R} - \frac{W(t)W(t-R)}{2R} KQ(t-R), \tag{8.4}$$

$$\dot{Q}(t) = \begin{cases} \dfrac{NW(t)}{R} - C, & Q(t) > 0, \\ \max\left(\dfrac{NW(t)}{R} - C, 0\right), & Q(t) = 0. \end{cases} \tag{8.5}$$

8.2. Реализация модели в xcos

Схема xcos, моделирующая систему (8.4)–(8.5), с начальными значениями параметров $N = 1$, $R = 1$, $K = 5,3$, $C = 1$, $W(0) = 0,1$, $Q(0) = 1$ приведена на рис. 8.1.

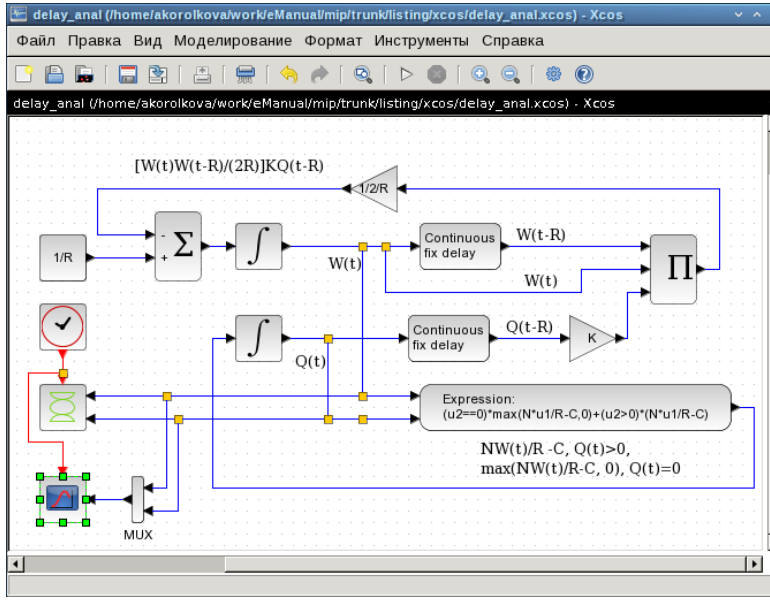


Рис. 8.1. Схема xcos, моделирующая систему (8.4)–(8.5)

Результат моделирования представлен на рис. 8.2 и 8.3.

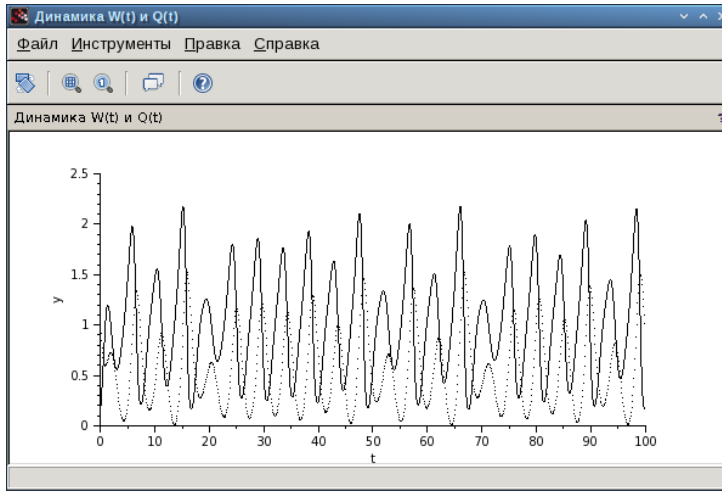


Рис. 8.2. Динамика изменения размера TCP окна $W(t)$ и размера очереди $Q(t)$

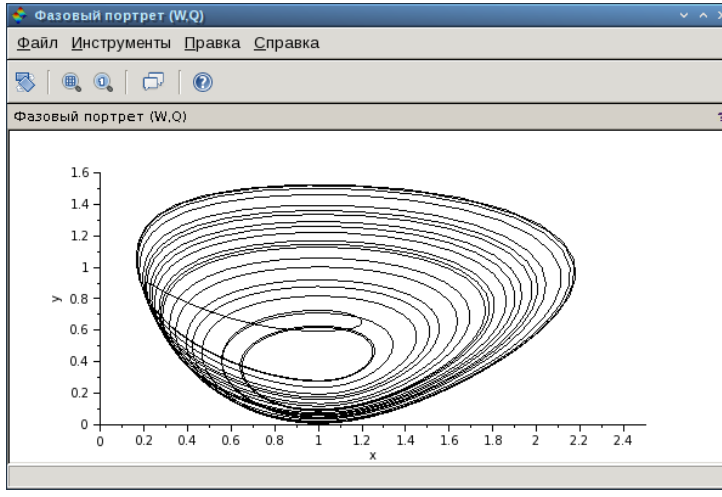


Рис. 8.3. Фазовый портрет (W, Q)

На рис. 8.2 представлена динамика изменения размера TCP окна $W(t)$ (сплошная линия) и размера очереди $Q(t)$ (пунктирная линия).

На рис. 8.3 представлен фазовый портрет (W, Q), который показывает наличие автоколебаний параметров системы — фазовая траектория осциллирует вокруг своей стационарной точки.

При $C = 0,9$ автоколебания более выраженные (рис. 8.4 и 8.5).

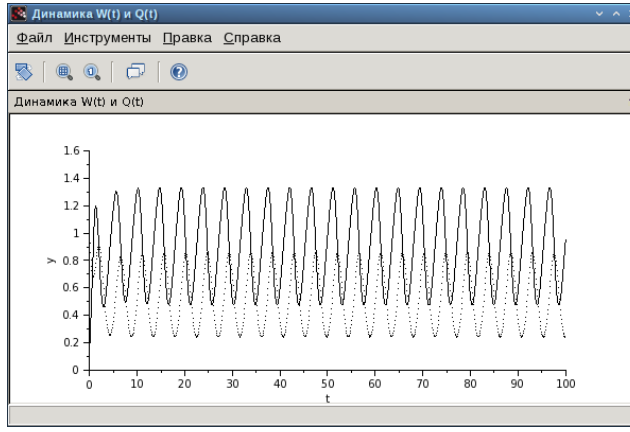


Рис. 8.4. Динамика изменения размера TCP окна $W(t)$ и размера очереди $Q(t)$ при $C = 0,9$

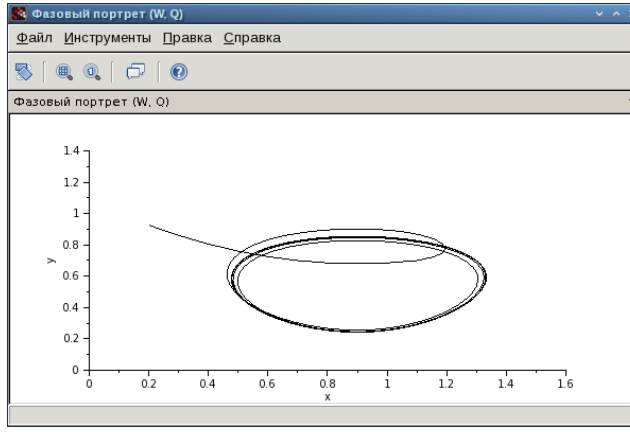


Рис. 8.5. Фазовый портрет (W, Q) при $C = 0,9$

8.3. Задание для самостоятельного выполнения

Реализовать модель (8.4)–(8.5) с использованием языка Modelica. Построить график динамики изменения размера TCP окна $W(t)$ и размера очереди $Q(t)$ и фазовый портрет (W, Q).

Требования к отчёту

1. Отчёт должен быть аккуратно оформлен: иметь титульный лист с указанием идентифицирующих работу данных; содержать формулировку задачи; иметь единообразный шрифт (основной текст: 13 pt, Times NewRoman, 1,5 интервал, выравнивание по ширине; текст листингов (если требуется): 10 Courier, 1 интервал; заголовки: 14 pt, Times NewRoman).
2. В отчёт включаются описания выполнения всех лабораторных работ раздела и заданий для самостоятельного выполнения.
3. Отчёт должен содержать скриншоты разработанных схем xcos с пояснениями в тексте на русском языке.
4. Отчёт должен содержать полученные в результате моделирования графики с пояснениями в тексте на русском языке.

Часть III
Сети Петри. Моделирование в CPN Tools

III.1. Теоретические сведения

В разделе использованы материалы из [5, 18].

CPN Tools — специальное программное средство, предназначенное для моделирования иерархических временных раскрашенных сетей Петри. Такие сети эквивалентны машине Тьюринга и составляют универсальную алгоритмическую систему, позволяющую описать произвольный объект.

CPN Tools позволяет визуализировать модель с помощью графа сети Петри и применить язык программирования CPN ML (Colored Petri Net Markup Language) для формализованного описания модели.

Назначение CPN Tools:

- разработка сложных объектов и моделирование процессов в различных прикладных областях, в том числе:
- моделирование производственных и бизнес-процессов;
- моделирование систем управления производственными системами и роботами;
- спецификация и верификация протоколов, оценка пропускной способности сетей и качества обслуживания, проектирование телекоммуникационных устройств и сетей.

Основные функции CPN Tools:

- создание (редактирование) моделей;
- анализ поведения моделей с помощью имитации динамики сети Петри;
- построение и анализ пространства состояний модели.

III.1.1. Интерфейс CPN Tools

Запустить CPN Tools можно в командной строке, введя:

```
cpntools &
```

После этого появится основное окно CPN Tools (рис. III.1.1).

Рис. III.1.1. Основное окно CPN Tools

Основное окно состоит из двух областей: рабочей области (справа) и области меню (слева), содержащего панель инструментов (Tool box), систему помощи (Help) и настройки опций (Options). В рабочую область можно вызвать как отдельные панели инструментов (рис. III.1.2), так и графическое представление модели в виде сети Петри (рис. III.1.3). Чтобы открыть палитру инструментов, необходимо перетащить её, удерживая нажатой левую кнопку мышки, из меню в рабочую область.

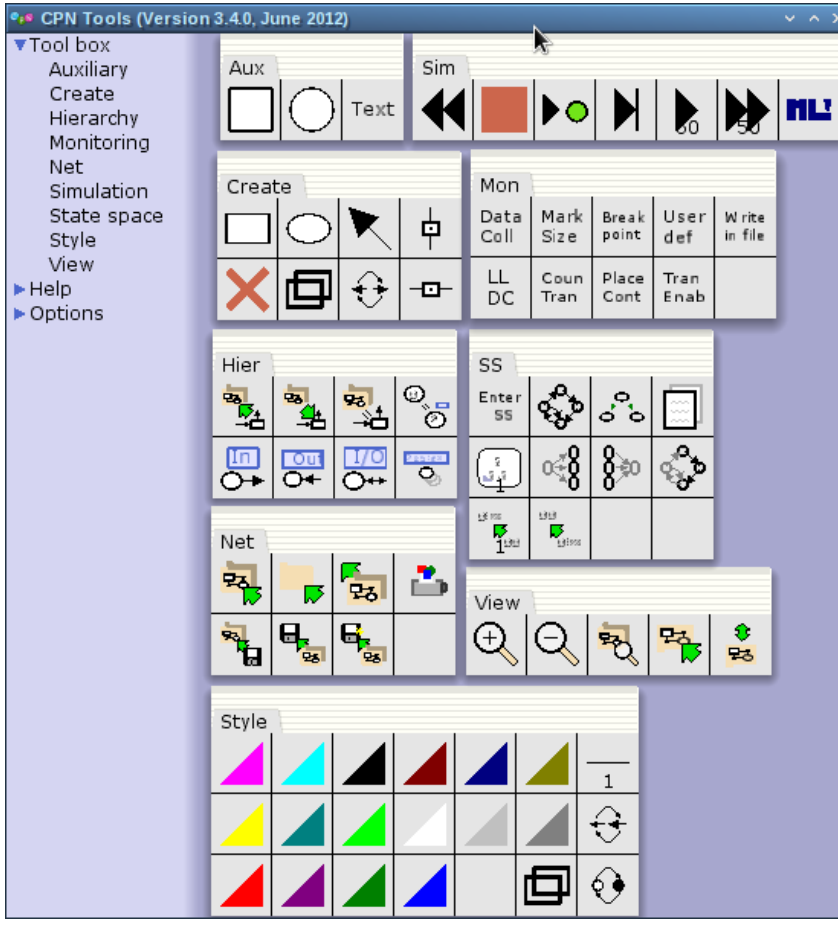


Рис. III.1.2. Панели инструментов в рабочей обоасти CPN Tools

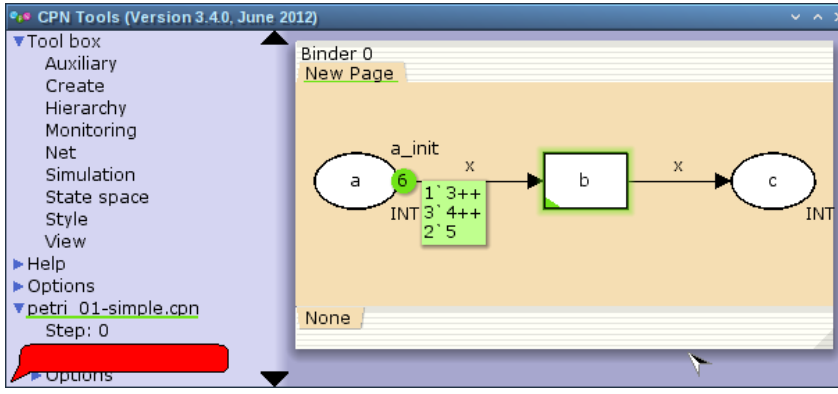


Рис. III.1.3. Графическое представление модели в виде сети Петри в рабочей области CPN Tools

Для взаимодействия элементами модели и палитрами инструментов CPN Tools предусматривает множество контекстно-зависимых меню, появляющихся на экране при нажатии на правую кнопку мыши. Меню имеет форму круга с названиями секторов (рис. III.1.4). Чтобы сохранять меню на экране, следует удерживать кнопку мыши, передвигая сектор для выбора требуемого элемента. В большинстве случаев элементы контекстно-зависимых меню дублируют инструменты в палитрах.

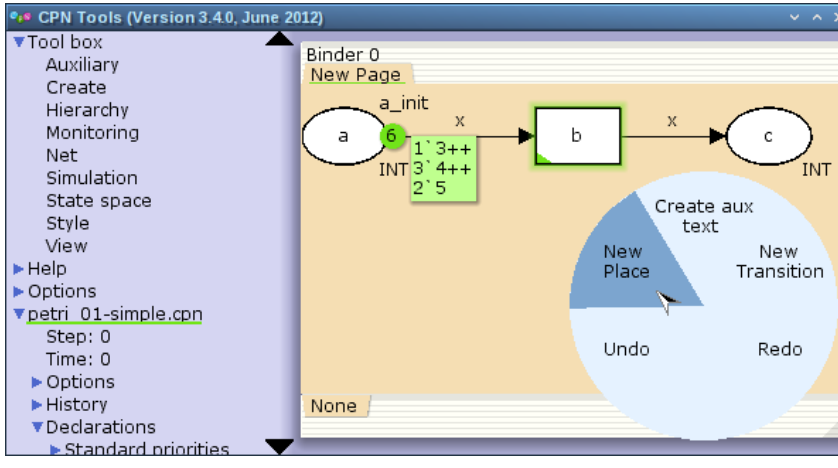


Рис. III.1.4. Вызов контекстно-зависимого меню CPN Tools

В CPN Tools модели называют сетями (net). Их описание расположено в меню под стандартными элементами. Каждая сеть в CPN Tools представлена следующими полями меню:
- имя — имя соответствующего файла с расширением `.cpn`;
- шаг (step) — количество шагов, выполненных при имитации;
- время (time) — текущее модельное время;
- опции (options) — опции сети;
- история (history) — список команд, которые были выполнены над сетью;
- описания (declarations) — описания множеств цветов, переменных, функций, констант;
- страницы (page) — названия страниц сети (рабочих областей модели).

Для создания новых описаний и новых страниц используются контекстно-зависимые меню. Чтобы открыть существующую страницу сети, необходимо перетащить её из бокового меню в рабочую область.

CPN Tools обеспечивает графическую обратную связь, которая отражает текущее состояние системы:
- *всплывающие сообщения* — прямоугольник, появляющийся при наведении мыши на объект и предоставляющий контекстно-зависимую информацию (ошибки во время проверки синтаксиса, ошибки при моделировании сети, подсказки для инструментов в палитрах, информация для индикатора состояний, результат применения инструмента оценки языка CPN ML, полный путь к сохранённой сети);
- *индикатор состояний* — показывает текущее состояние модели (зелёный — действие было успешно завершено; красный — при выполнении действия произошла ошибка; фиолетовый — в данный момент выполняется длительная операция);
- *подсветка элемента* — цветовая кодировка для выделения объектов с определенными свойствами:
  - ярко-красный — указывает на объекты с ошибками при проверке синтаксиса и моделировании сети;
  - темно-красный — указывает на повторяемые имена позиций и переходов при проверке синтаксиса;
  - зелёный — указывает разрешённые переходы при моделировании сети;
  - темно-синий — указывает зависимость между описаниями и другими элементами, такими как позиции, переходы и страницы;
  - цвет морской волны — указывает объект, который содержит описание;
  - оранжевый – указывает на то, что проверка синтаксиса объекта ещё не началась;
  - жёлтый — указывает на то, что проверка синтаксиса объекта выполняется в настоящее время;

  - розовый — указывает на то, какие объединенные позиции принадлежат множеству слияния;
  - цвет морской волны — указывает назначения порта/сокета и взаимосвязи страниц/подстраниц при работе с иерархическими сетями;
- *изменяющаяся форма курсора* – определяет действия, которые могут быть выполнены.

Инструменты CPN Tools (см. рис. III.1.2):

- *сетевые инструменты (Net)* — для операций с сетями (рис. III.1.5):
  - создать новую сеть;
  - создать новую страницу;
  - закрыть сеть;
  - загрузить сеть;
  - сохранить сеть;
  - сохранить сеть как;
  - печатать сеть;
- инструменты создания элементов сети (Create) — для рисования и редактирования сетей Петри (рис. III.1.6):
  - создать переход;
  - создать позицию;
  - создать дугу;
  - создать вертикальную магнитную опорную линию;
  - удалить элемент;
  - клонировать элемент;
  - переключение возможных направлений дуги;
  - создать горизонтальную магнитную опорную линию;

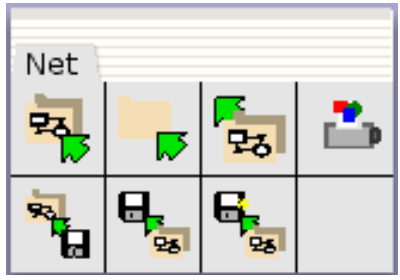


Рис. III.1.5. Сетевые инструменты CPN Tools

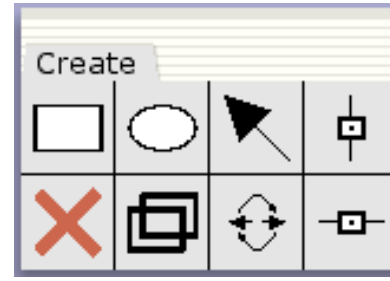


Рис. III.1.6. Инструменты создания элементов CPN Tools

- инструменты иерархии (Hierarchy) — для создания многоуровневых сетей (рис. III.1.7); содержат инструменты, как для восходящего, так и для нисходящего структурирования сети;

– инструменты пространства состояний (State Space) — для построения и анализа пространства состояний (рис. III.1.8):
  – войти в пространство состояний;
  – вычислить пространство состояний;
  – построить граф пространства состояний (сильно связанных состояний);
  – сохранить отчёт;
  – показать узел с определённым номером;
  – показать потомков узла;
  – показать родительский элемент узла;
  – показать часть графа сети;
  – запустить симуляцию на основе пространства состояний (передать состояние из пространства состояний в имитацию);
  – создать пространство состояний на основе симуляции сети (передать состояние из имитации в пространство состояний);

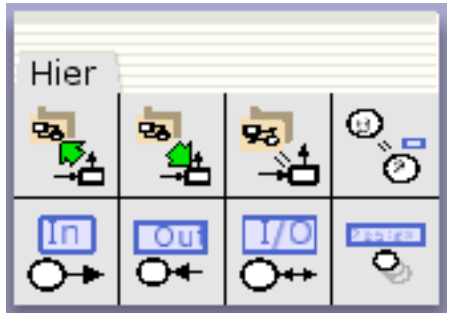


Рис. III.1.7. Инструменты иерархии CPN Tools

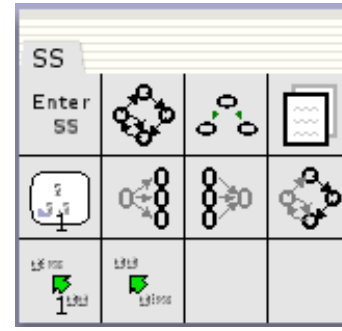


Рис. III.1.8. Инструменты пространства состояний CPN Tools

– инструменты моделирования (Simulate) — для имитации поведения сети (рис. III.1.9):
  – вернуться в начальное состояние;
  – остановить текущее моделирование;
  – выполнить переход с выбранными параметрами фишки;
  – выполнить переход;
  – выполнить указанное количество переходов, показывая промежуточные маркировки;
  – выполнить указанное количество переходов, не показывая промежуточные маркировки;
  – оценить текст как ML код.
– инструменты отображения (View) — для выбора масштаба и указания групп элементов (рис. III.1.10);

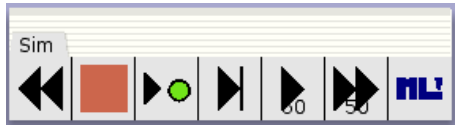


Рис. III.1.9. Инструменты моделирования CPN Tools

Рис. III.1.10. Инструменты отображения CPN Tools

– инструменты стиля (Style) — для указания особенностей внешнего вида сетей (рис. III.1.11);
– вспомогательные инструменты (Auxiliary) — для повышения наглядности модели (рис. III.1.12).

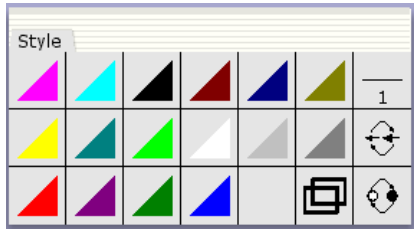


Рис. III.1.11. Инструменты стиля CPN Tools

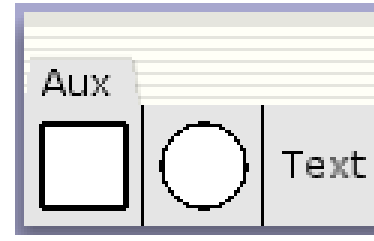


Рис. III.1.12. Вспомогательные инструменты CPN Tools

III.1.2. Основы языка CPN ML

CPN Tools использует язык CPN ML для создания описаний в меню и атрибутов элементов сетей (типов данных, переменных, функций, констант). У каждой позиции раскрашенной сети Петри должен быть обязательный атрибут – множество цветов, т.е. позиция может содержать фишки только из указанного множества. Переменные и функции используются для указания атрибутов переходов и дуг.

Стандартные предопределённые описания множеств цветов : `E` — элементарный тип, `INT` — целые числа, `BOOL` — логический тип, `STRING` — строковый тип.

CPN ML обеспечивает следующие *простые множества цветов*: `unit`, `boolean`, `integer`, `string`, `enumerated`, `index`.

Синтаксис описания `unit`:

```
colset name = unit [with new_unit];
```

Синтаксис описания `boolean`:

```
colset name = bool [with (new_false, new_true)];
```

К логическим переменным могут быть применены следующие операции: отрицание логической переменной b: `not b`; логическое умножение (конъюнкция): `b1 andalso b2`; логическое сложение (дизъюнкция): `b1 orelse b2`.

Синтаксис описания `integer`:

```
colset name = int [with int-exp1..int-exp2];
```

Опции позволяют ограничить множество цветов интервалом, заданным двумя выражениями: от `int-exp1` до `int-exp2`:

```
colset Dozen = int with 1..12;
```

К целым переменным могут быть применены следующие операции: `+`, `-`, `div`, `mod`, `abs`, `Int.min`, `Int.max`, `~` — унарный минус.

Синтаксис описания `string`:

```
colset name = string [with string-exp1..string-exp2
[and int-exp1..int-exp2]];
```

Опции определяют диапазоны допустимых символов:

```
colset LowerString = with "a".."z";
```

К строковым переменным могут быть применены следующие операции: `^` — конкатенация (операция последовательного объединения двух строк в одну), `String.size` — размер, `substring` — подстрока.

Синтаксис описания `enumerated`:

```
colset name = with id0 | id1 | ... | idn;
```

Значения явно перечисляются как идентификаторы в описании.

Синтаксис описания `index`:

```
colset name = index id with int-exp1..int-exp2;
```

Индексированные переменные — последовательности значений, состоящих из идентификатора и спецификатора индекса. Индексированные значения имеют вид: `id i` или `id (i)`, где `i` — целое число и `int-exp1 <= i <= int-exp2`.

Составные множества цветов состоят из комбинаций простых множеств цветов. CPN ML обеспечивает следующие составные множества цветов: `product`, `record`, `union`, `list`, `subset` и `alias`.

`Product` и `record` представляют собой кортежи данных, сформированные как результат декартового произведения множеств цветов. Компоненты множества цветов `product` являются безымянными, компоненты множества цветов `record` — поименованы.

Синтаксис описания `product`:

```
colset name = product name1 * name2 * ... * namen;
```

Значения этого множества цветов имеют вид: `(v1, v2, ..., vn)`, где `vi` имеет тип `namei` для `1<=i<=n`. Чтобы извлечь i–й элемент из кортежа, используется следующая операция:

```
#i name
```

Синтаксис описания `record`:

```
colset name = record id1:name1 * id2:name2 * ... *
idn:namen;
```

Значения этого множества цветов имеют вид

```
{id1=v1, id2=v2, ..., idn=vn}
```

Здесь `vi` — это переменные типа `namei` для `1<=i<=n`. Чтобы извлечь i–й элемент из кортежа, используется следующая операция:

```
#idi name
```

Синтаксис описания `alias`:

```
colset name = name0;
```

`alias` введено для того, чтобы использовать различные названия одного и того же множества цветов.

Переменная — это идентификатор, значение которого может быть изменено во время выполнения модели.

Синтаксис описания переменных:

```
var id1, id2, ..., idn : cs_name;
```

Здесь `idi` — идентификаторы, `cs_name` — имя предварительно описанного множества цветов.

Описание величины сопоставляет некоторое значение указанному идентификатору (задаёт константу).

Синтаксис описания величины:

```
val id = exp;
```

Здесь `id` — идентификатор и `exp` выражение CPN ML. Выражение задаёт значение, которое будет связано с идентификатором.

Функции CPN ML используют стандартные управляющие структуры языков программирования, такие как операторы «if» и «case».

Синтаксис описания функции:

```
fun id pat1 = exp1
  | id pat2 = exp2
  | ...
  | id patn = expn;
```

Здесь pat1, pat2, ..., patn — шаблоны и выражения exp1, exp2, ..., expn имеют один и тот же тип. Описание означает, что в случае, когда фактические аргументы удовлетворяют шаблону pati, значение функции будет вычислено как expi.

Пример вычисления факториала целого числа, используя рекурсию:

```
fun fact ( 0 ) = 1
  | fact ( i ) = i * fact( i-1);
```

Управляющие структуры if-then-else и case:

```
if bool-exp then exp1 else exp2;
```

```
case exp of
     pat1 => exp1
   | pat2 => exp2
   | ...
   | patn => expn;
```

где exp1, exp2, ..., expn имеют тот же самый тип.

Структура let задаёт описания локальных переменных в функциях:

```
let
   val pat1 = exp1
   val pat2 = exp2
   .........
   val patn = expn
in
   exp
end;
```

Случайные функции обеспечивают возможность моделирования статистических характеристик:

- *свободные (несвязанные) переменные* являются переменными выходных дуг переходов, значения которых не определены входными дугами и другими атрибутами;
- *функция* ran генерирует случайную величину для больших множеств цветов (более 100);
- *специальные случайные функции с указанным законом распределения* (Бернулли, биномиальное, Эрланга, показательное, нормальное, Пуассоновское, Стьюдента, равномерное).

Мультимножества используются для представления маркировки позиций. Мультимножества содержат элементы с определенной кратностью. Символ обратной кавычки (`) является конструктором мультимножества. Например, 3`5 – это мультимножество, состоящее из трех элементов типа 5.

Конструктор мультимножества, дополненный операторами сложения (++) и вычитания (--) мультимножеств, обеспечивает лаконичный способ задания мультимножеств.

Для работы с мультимножествами используются следующие константы, операции и функции:

- `empty` — константа empty создает пустое мультимножество, которое сопоставимо с любыми типами мультимножеств;
- `ms1 == ms2` — равенство мультимножеств;
- `ms1 <><> ms2` — неравенство мультимножеств;
- `ms1 >> ms2` — мультимножество больше, чем;
- `ms1 >>== ms2` — мультимножество больше или равно;
- `ms1 << ms2` — мультимножество меньше, чем;
- `ms1 <<== ms2` — мультимножество меньше или равно;
- `ms1 ++ ms2` — сложение мультимножеств;
- `ms1 -- ms2` — вычитание мультимножеств (`ms2` должно быть меньше или равно `ms1`), генерирует прерывание, если `ms2` не меньше или равно `ms1`;
- `i ** ms` — скалярное произведение;
- `size ms` — размер мультимножества;
- `random ms` — возвращает псевдослучайный цвет из мультимножества;
- `cf(c,ms)` — возвращает количество появлений цвета (типа) в мультимножестве;
- `filter p ms` — берет предикат `p` и мультимножество `ms` и генерирует мультимножество всех элементов `ms`, удовлетворяющих предикату `p`.

Например,

```
m1 = 2`5 ++ 3`4 ++ 4`5;
m2 = 1`5 ++ 2`4 ++ 3`5;
```

тогда

```
m1 ++ m2 = 3`5 ++ 5`4 ++ 7`5
m1 - m2 = 1`5 ++ 1`4 ++ 1`3
m1 >> m2  имеет значение true
size m1 = 9
cf(4,m1) = 3
```

В CPN Tools начальная и текущая маркировки позиции представлены мультимножеством, заданным на множестве цветов позиции. При выборе фишки в переменную надписи дуги CPN Tools обеспечивает случайный выбор фишки, как с помощью функции `random`.

Временные мультимножества используются, чтобы представить временные задержки в модели. Описание соответствующего множества цветов должно быть дополнено модификатором `timed`. Операторы `@`, `@+`, и `@@ +` используются, чтобы добавить временные метки

к цветам. Добавление временной задержки `x` к цвету `c` присоединяет временную метку с величиной, равной текущему времени модели `+ x`, к цвету `c`.

Операции, применимые к временным мультимножествам:

- `c @ t` — присоединение временной метки (с типом `Time.time`) к цвету `c`;
- `ms @+ i` — добавление целой временной задержки `i` к каждому из цветов мультимножества `ms`, возвращает временное мультимножество;
- `tms1 +++ tms2` — сложение временных мультимножеств.

Например, описание

```
colset tint = int timed;
var t : tint;
t = 1`2@100 ++ 1`3@200 ++ 1`4@300;
```

означает, что фишка `2@100` может быть использована только тогда, когда модельное время больше или равно 100, фишка `3@200` – только когда модельное время больше или равно 200 и т.д. До наступления момента активации фишка не может быть извлечена ни одним переходом модели.

III.1.3. Язык описания моделей

В CPN Tools каждый элемент сети Петри имеет свои атрибуты, описанные на языке CPN ML. Используя инструменты для создания элементов сети, можно поместить элемент на страницу модели. Затем добавляются атрибуты элементов. Для этого необходимо щелкнуть мышью на соответствующем элементе и использовать кнопку `Tab` на клавиатуре для переключения атрибутов. Нажатие на клавиатуре клавиши `Esc` позволяет покинуть выбранный элемент; такой же результат может быть получен щелчком мыши в любой другой области модели.

Атрибуты позиций (рис. III.1.13):

- цвет фишек;
- начальная маркировка;
- имя.

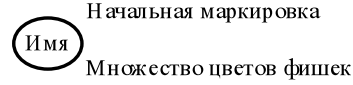


Рис. III.1.13. Атрибуты позиций CPN Tools

В качестве *атрибута дуги* выступает выражение `expr`, которое должно соответствовать множеству цветов позиции, связанной с ду-

гой (рис. III.1.14). Выражение входной дуги перехода является шаблоном для выбора фишки. Этот шаблон описывается как предикат, который может быть применен в функции `filter`, для соответствующего множества цветов. Выражение выходной дуги перехода представляет собой конструктор для создания новых фишек. Такой конструктор часто использует переменные надписей входных дуг и в простых случаях может совпадать с одной из них.

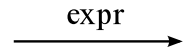


Рис. III.1.14. Атрибут дуги CPN Tools

Атрибуты переходов (рис. III.1.15):

- имя перехода;
- условие запуска перехода (логическое выражение, которое оценивается как `true` (истина) или `false` (ложь));
- время задержки (положительное целое число);
- сегмент кода (процедура языка ML, которая используется для более сложной обработки входных фишек).

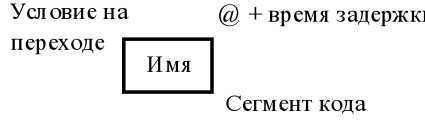


Рис. III.1.15. Атрибуты переходов CPN Tools

Задержки переходов применяются ко всем выходным фишкам перехода.

Лабораторная работа 9. Модель «Накорми студентов»

Рассмотрим пример студентов, обедающих пирогами. Голодный студент становится сытым после того, как съедает пирог.

Таким образом, имеем:
- два типа фишек: «пироги» и «студенты»;
- три позиции: «голодный студент», «пирожки», «сытый студент»;
- один переход: «съесть пирожок».

1. Рисуем граф сети. Для этого с помощью контекстного меню создаём новую сеть, добавляем позиции, переход и дуги (рис. 9.1).

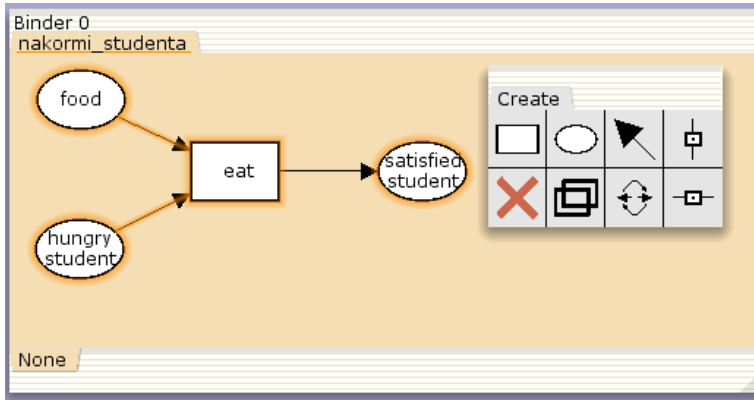


Рис. 9.1. Граф сети модели «Накорми студентов»

2. В меню задаём новые декларации модели: типы фишек, начальные значения позиций, выражения для дуг. Для этого наведя мышку на меню Standart declarations, правой кнопкой вызываем контекстное меню и выбираем New Decl (рис. 9.2).

После этого задаем тип `s` фишкам, относящимся к студентам, тип `p` — фишкам, относящимся к пирогам, задаём значения переменных `x` и `y` для дуг и начальные значения мультимножеств `init_stud` и `init_food` (рис. 9.3):

```
colset s=unit with student;
colset p=unit with pasty;
var x:s;
var y:p;
val init_stud = 3`student;
val init_food = 5`pasty;
```

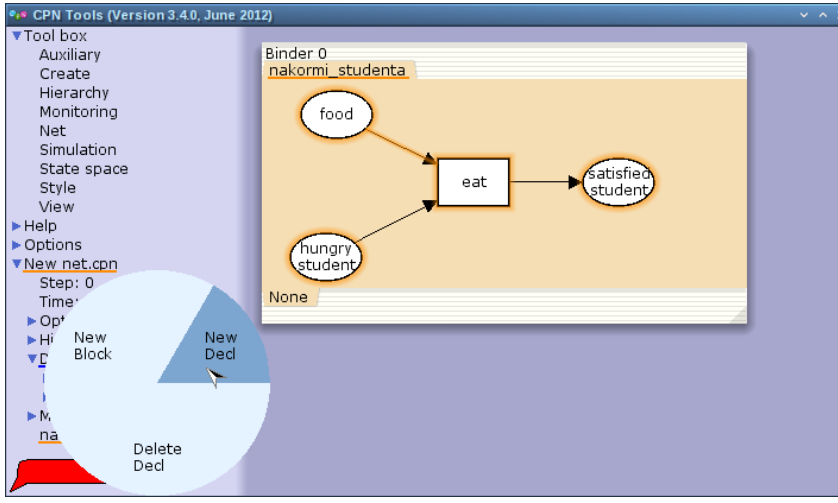


Рис. 9.2. Задание деклараций модели «Накорми студентов»

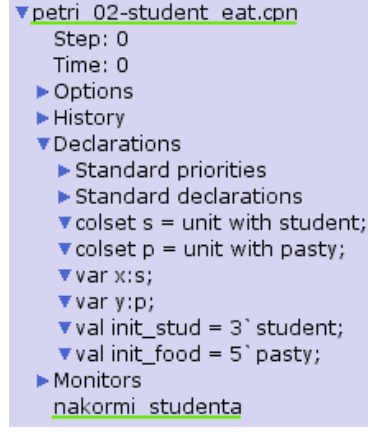


Рис. 9.3. Декларации модели «Накорми студентов»

В результате получаем работающую модель (рис. 9.4).

После запуска фишки типа «пирожки» из позиции «еда» и фишки типа «студенты» из позиции «голодный студент», пройдя через переход «кушать», попадают в позицию «сытый студент» и преобразуются в тип «студенты» (рис. 9.5).

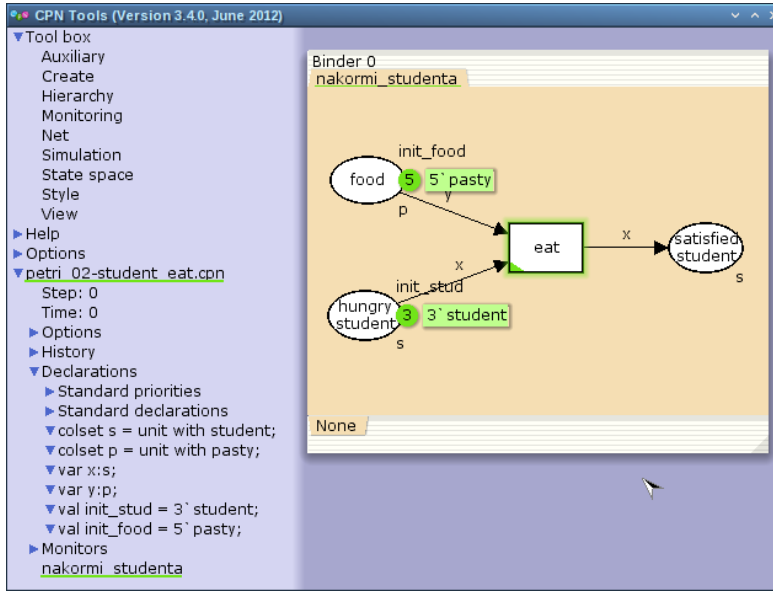


Рис. 9.4. Модель «Накорми студентов»

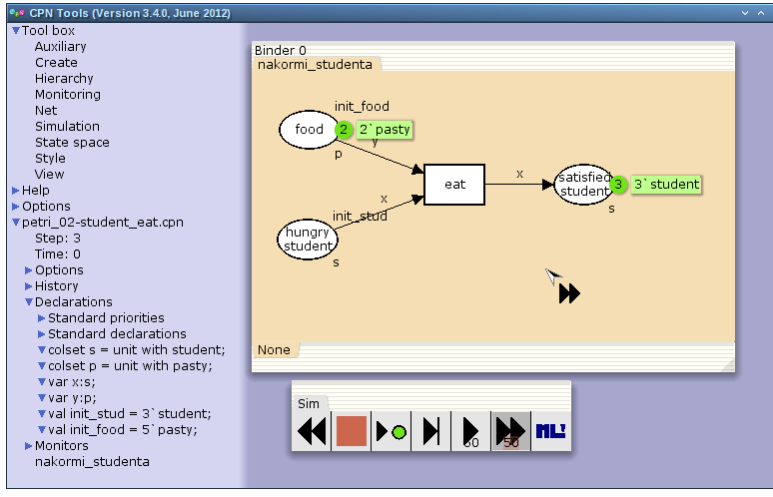


Рис. 9.5. Запуск модели «Накорми студентов»

Лабораторная работа 10. Задача об обедающих мудрецах

Задача об обедающих мудрецах — классическая задача о блокировках и синхронизации процессов.

10.1. Постановка задачи

Пять мудрецов сидят за круглым столом и могут пребывать в двух состояниях — думать и есть. Между соседями лежит одна палочка для еды. Для приёма пищи необходимы две палочки. Палочки — пересекающийся ресурс. Необходимо синхронизировать процесс еды так, чтобы мудрецы не умерли с голода.

10.2. Построение модели с помощью CPNTools

1. Рисуем граф сети. Для этого с помощью контекстного меню создаём новую сеть, добавляем позиции, переходы и дуги (рис. 10.1).

Начальные данные:

- позиции: мудрец размышляет (philosopher thinks), мудрец ест (philosopher eats), палочки находятся на столе (sticks on the table)
- переходы: взять палочки (take sticks), положить палочки (put sticks)

2. В меню задаём новые декларации модели: типы фишек, начальные значения позиций, выражения для дуг:

- n — число мудрецов и палочек ($n = 5$);
- p — фишки, обозначающие мудрецов, имеют перечисляемый тип `PH` от 1 до n;
- s — фишки, обозначающие палочки, имеют перечисляемый тип `ST` от 1 до n;
- функция `ChangeS(p)` ставит в соответствие мудрецам палочки (возвращает номера палочек, используемых мудрецами); по условию задачи мудрецы сидят по кругу и мудрец $p(i)$ может взять i и $i+1$ палочки, поэтому функция `ChangeS(p)` определяется следующим образом:

```
fun ChangeS (ph(i))=
1`st(i)++st(if = n then 1 else i+1)
```

В результате получаем работающую модель (рис. 10.3).

После запуска модели наблюдаем, что одновременно палочками могут воспользоваться только два из пяти мудрецов (рис. 10.4).

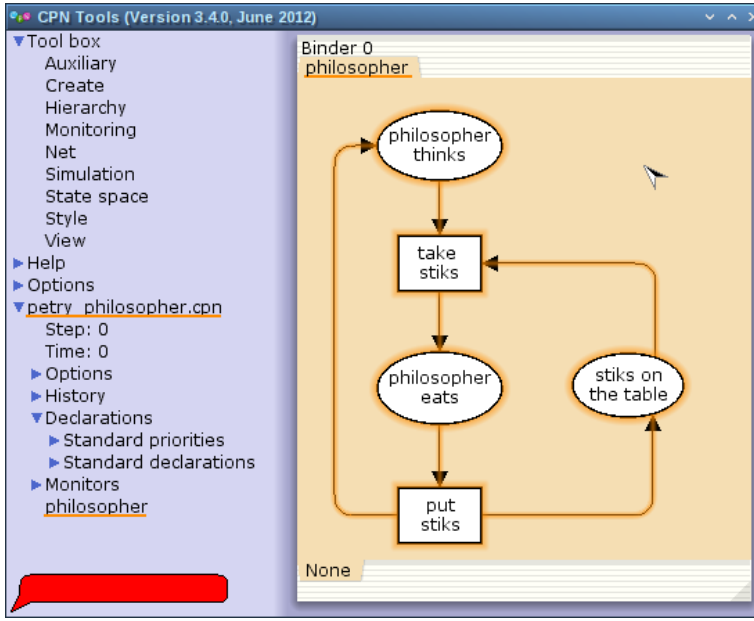


Рис. 10.1. Граф сети задачи об обедающих мудрецах

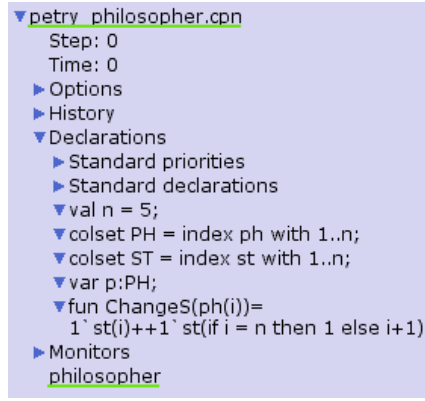


Рис. 10.2. Задание деклараций задачи об обедающих мудрецах

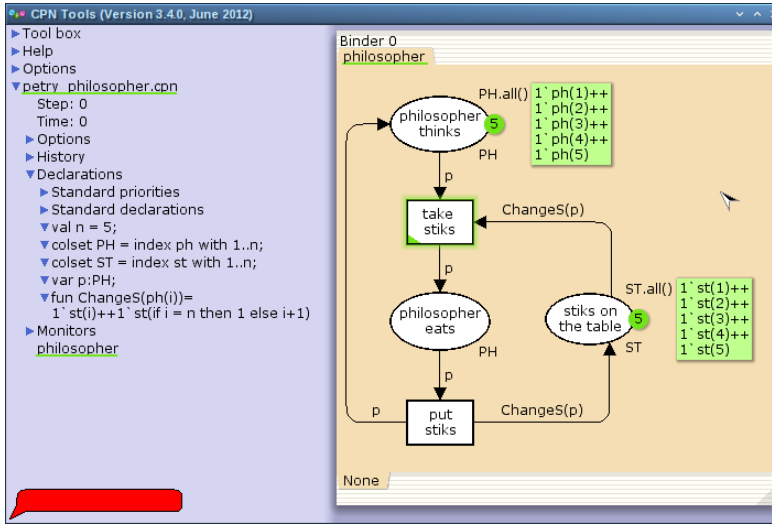


Рис. 10.3. Модель задачи об обедающих мудрецах

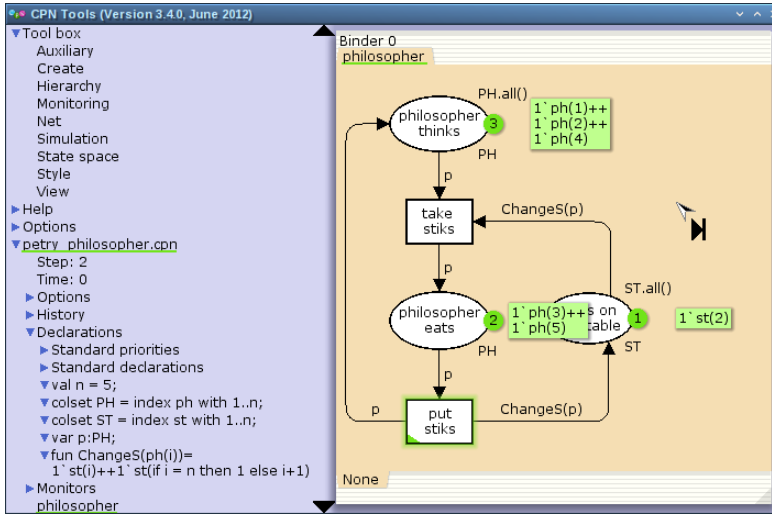


Рис. 10.4. Запуск модели задачи об обедающих мудрецах

Лабораторная работа 11. Модель системы массового обслуживания $M|M|1$

11.1. Постановка задачи

В систему поступает поток заявок двух типов, распределённый по пуассоновскому закону. Заявки поступают в очередь сервера на обработку. Дисциплина очереди - FIFO. Если сервер находится в режиме ожидания (нет заявок на сервере), то заявка поступает на обработку сервером.

11.2. Построение модели с помощью CPNTools

1. Будем использовать три отдельных листа: на первом листе опишем граф системы (рис. 11.1), на втором — генератор заявок (рис. 11.2), на третьем — сервер обработки заявок (рис. 11.3).

1.1. Сеть имеет 2 позиции (очередь — `Queue`, обслуженные заявки — `Complited`) и два перехода (генерировать заявку — `Arrivals`, передать заявку на обработку серверу — `Server`). Переходы имеют сложную иерархическую структуру, задаваемую на отдельных листах модели (с помощью соответствующего инструмента меню — Hierarchy).

Между переходом `Arrivals` и позицией `Queue`, а также между позицией `Queue` и переходом `Server` установлена дуплексная связь. Между переходом `Server` и позицией `Complited` — односторонняя связь.

1.2. Граф генератора заявок имеет 3 позиции (текущая заявка — `Init`, следующая заявка — `Next`, очередь — `Queue` из листа `System`) и 2 перехода (`Init` — определяет распределение поступления заявок по экспоненциальному закону с интенсивностью 100 заявок в единицу времени, `Arrive` — определяет поступление заявок в очередь).

1.3. Граф процесса обработки заявок на сервере имеет 4 позиции (`Busy` — сервер занят, `Idle` — сервер в режиме ожидания, `Queue` и `Complited` из листа `System`) и 2 перехода (`Start` — начать обработку заявки, `Stop` — закончить обработку заявки).

2. Зададим декларации системы.

Определим множества цветов системы (colorset):

- фишки типа `UNIT` определяют моменты времени;
- фишки типа `INT` определяют моменты поступления заявок в систему.
- фишки типа `JobType` определяют 2 типа заявок — `A` и `B`;

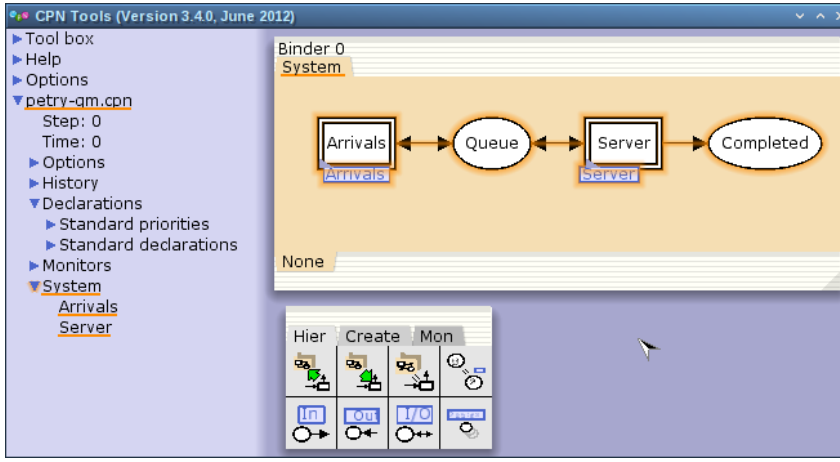


Рис. 11.1. Граф сети системы обработки заявок в очереди

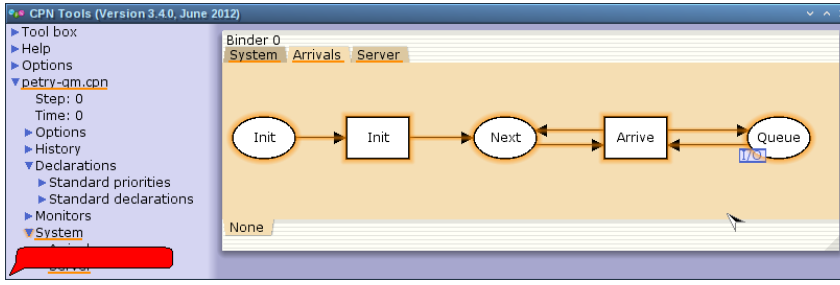


Рис. 11.2. Граф генератора заявок системы

– кортеж `Job` имеет 2 поля: `jobType` определяет тип работы (соответственно имеет тип `JobType`, поле `AT` имеет тип `INT` и используется для хранения времени нахождения заявки в системе;
– фишки `Jobs` — список заявок;
– фишки типа `ServerxJob` — определяют состояние сервера, занятого обработкой заявок.

```
colset UNIT = unit timed;
colset INT = int;
colset Server = with server timed;
colset JobType = with A | B;
colset Job = record jobType : JobType *
                    AT : INT;
```

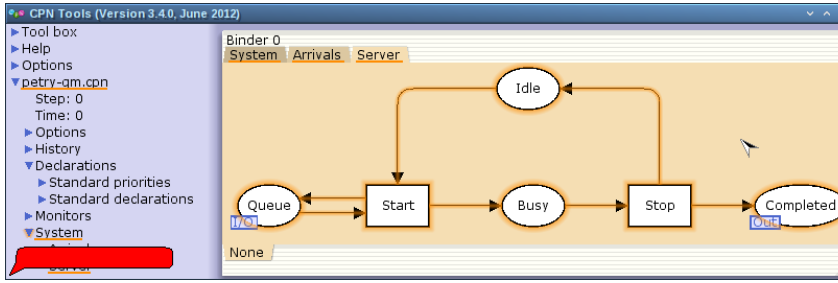


Рис. 11.3. Граф процесса обработки заявок на сервере системы

```
colset Jobs = list Job;
colset ServerxJob = product Server * Job timed;
```

Переменные модели:

- `proctime` — определяет время обработки заявки;
- `job` — определяет тип заявки;
- `jobs` — определяет поступление заявок в очередь.

```
var proctime : INT;
var job: Job;
var jobs: Jobs;
```

Определим функции системы:

- функция `expTime` описывает генерацию целочисленных значений через интервалы времени, распределённые по экспоненциальному закону;
- функция `intTime` преобразует текущее модельное время в целое число;
- функция `newJob` возвращает значение из набора `Job` — случайный выбор типа заявки (`A` или `B`).

```
fun expTime (mean: int) =
    let
        val realMean = Real.fromInt mean
        val rv = exponential((1.0/realMean))
    in
        floor (rv+0.5)
    end;

fun intTime() = IntInf.toInt (time());

fun newJob() = {jobType = JobType.ran(),
                AT      = intTime()}
```

3. Зададим параметры модели на графах сети.

3.1. На листе `System` (рис. 11.4):

– у позиции `Queue` множество цветов фишек — `Jobs`; начальная маркировка `1`[]` определяет, что изначально очередь пуста.
– у позиции `Completed` множество цветов фишек — `Job`.

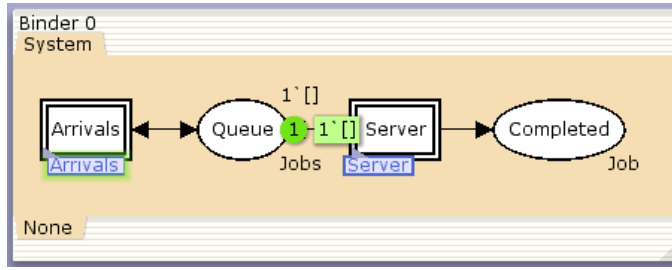


Рис. 11.4. Параметры элементов основного графа системы обработки заявок в очереди

3.2. На листе `Arrivals` (рис. 11.5):

– у позиции `Init`: множество цветов фишек — `UNIT`; начальная маркировка `1`()@0` определяет, что поступление заявок в систему начинается с нулевого момента времени;
– у позиции `Next`: множество цветов фишек — `UNIT`;

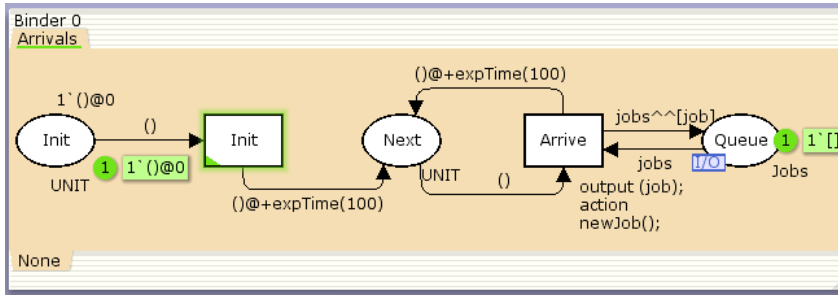


Рис. 11.5. Параметры элементов генератора заявок системы

– на дуге от позиции `Init` к переходу `Init` выражение `()` задаёт генерацию заявок;
– на дуге от переходов `Init` и `Arrive` к позиции `Next` выражение `()@+expTime(100)` задаёт экспоненциальное распределение времени между поступлениями заявок;
– на дуге от позиции `Next` к переходу `Arrive` выражение `()` задаёт перемещение фишки;

– на дуге от перехода `Arrive` к позиции `Queue` выражение `jobs^^[job]` задает поступление заявки в очередь;
– на дуге от позиции `Queue` к переходу `Arrive` выражение `jobs` задаёт обратную связь.

3.3. На листе `Server` (рис. 11.6):

– у позиции `Busy`: множество цветов фишек — `Server`, начальное значение маркировки — `1`server@0` определяет, что изначально на сервере нет заявок на обслуживание;
– у позиции `Idle`: множество цветов фишек — `ServerxJob`;
– переход `Start` имеет сегмент кода `output (proctime); action expTime(90);` определяющий, что время обслуживания заявки распределено по экспоненциальному закону со средним временем обработки в 90 единиц времени;
– на дуге от позиции `Queue` к переходу `Start` выражение `job::jobs` определяет, что сервер может начать обработку заявки, если в очереди есть хотя бы одна заявка;
– на дуге от перехода `Start` к позиции `Busy` выражение `(server,job)@+proctime` запускает функцию расчёта времени обработки заявки на сервере;
– на дуге от позиции `Busy` к переходу `Stop` выражение `(server,job)` говорит о завершении обработки заявки на сервере;
– на дуге от перехода `Stop` к позиции `Completed` выражение `job` показывает, что заявка считается обслуженной;
– выражение `server` на дугах от и к позиции `Idle` определяет изменение состояние сервера (обрабатывает заявки или ожидает);
– на дуге от перехода `Start` к позиции `Queue` выражение `jobs` задаёт обратную связь.

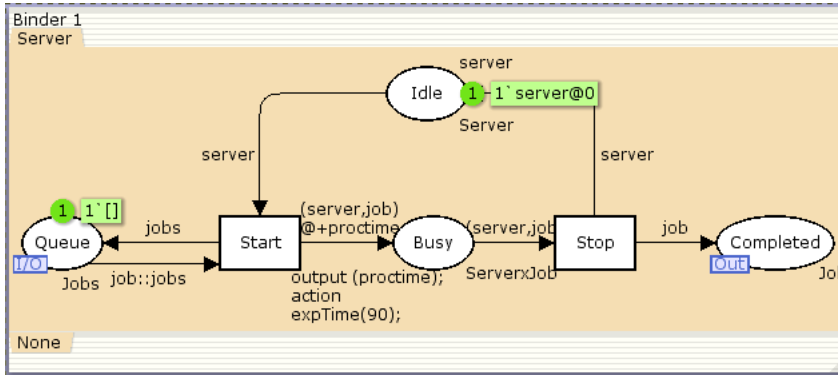


Рис. 11.6. Параметры элементов обработчика заявок системы

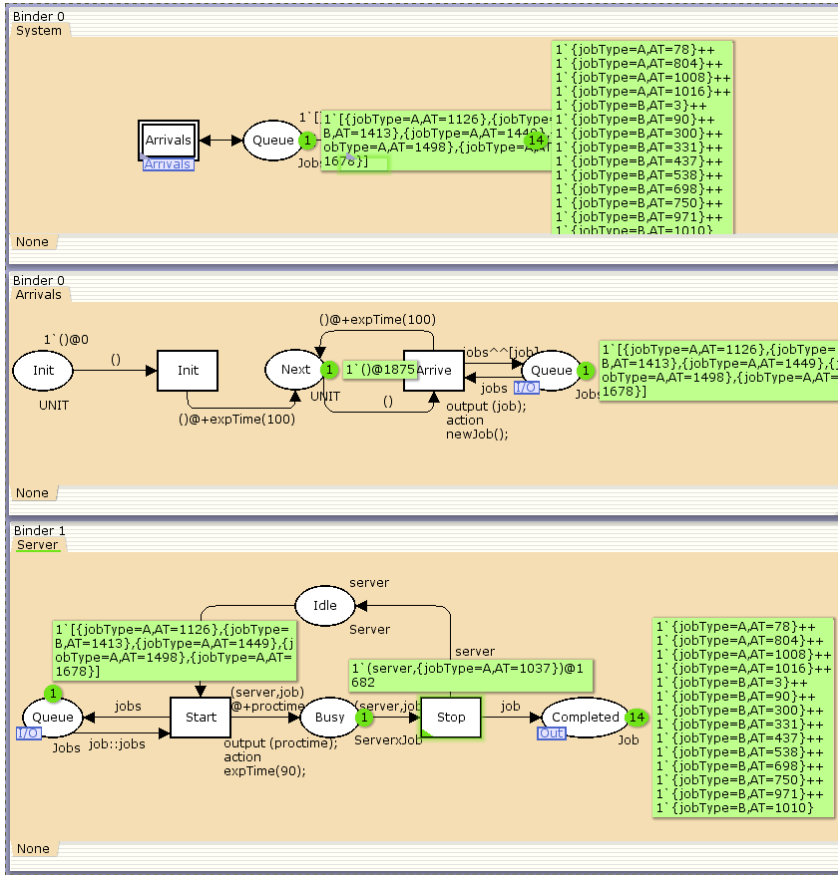


Рис. 11.7. Запуск системы обработки заявок в очереди

11.3. Мониторинг параметров моделируемой системы

Цель: мониторинг параметров очереди системы $M|M|1$.

Потребуется палитра *Monitoring*. Выбираем *Break Point* (точка останова) и устанавливаем её на переход *Start*. После этого в разделе меню *Monitor* появится новый подраздел, который назовём *Ostanovka*. В этом подразделе необходимо внести изменения в функцию `Predicate`, которая будет выполняться при запуске монитора:

```
fun pred (bindelem) =
let
  fun predBindElem (Server'Start (1, {job,jobs,proctime}))
    = true
    | predBindElem _ = false
in
  predBindElem bindelem
end
```

Изначально, когда функция начинает работать, она возвращает значение `true`, в противном случае — `false`. В теле функции вызывается процедура `predBindElem`, которую определяем в предварительных декларациях.

Зададим число шагов, через которое будем останавливать мониторинг. Для этого `true` заменим на `Queue_Delay.count()=200` (рис. 11.8):

```
fun pred (bindelem) =
let
  fun predBindElem (Server'Start (1, {job,jobs,proctime}))
    = Queue_Delay.count()=200
    | predBindElem _ = false
in
  predBindElem bindelem
end
```

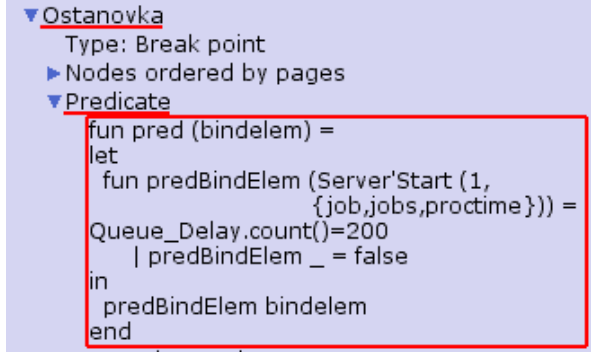


Рис. 11.8. Функция Predicate монитора Ostanovka

Необходимо определить конструкцию `Queue_Delay.count()`. С помощью палитры *Monitoring* выбираем `Data Call` и устанавливаем на переходе `Start`. Появившийся в меню монитор называем `Queue Delay` (без подчеркивания).

Функция `Observer` выполняется тогда, когда функция предикатора выдаёт значение `true`. По умолчанию функция выдаёт 0 или унарный минус (`~1`), подчёркивание обозначает произвольный аргумент.

```
fun obs (bindelem) =
let
 fun obsBindElem (Server'Start (1, {job, jobs, proctime})) = 0
     | obsBindElem _ = ~1
in
  obsBindElem bindelem
end
```

Изменим её так, чтобы получить значение задержки в очереди. Для этого необходимо из текущего времени `intTime()` вычесть временную метку AT, означающую приход заявки в очередь (рис. 11.9):

```
fun obs (bindelem) =
let
  fun obsBindElem (Server'Start (1, {job, jobs, proctime}))
     = (intTime() - (#AT job))
     | obsBindElem _ = ~1
in
  obsBindElem bindelem
end
```

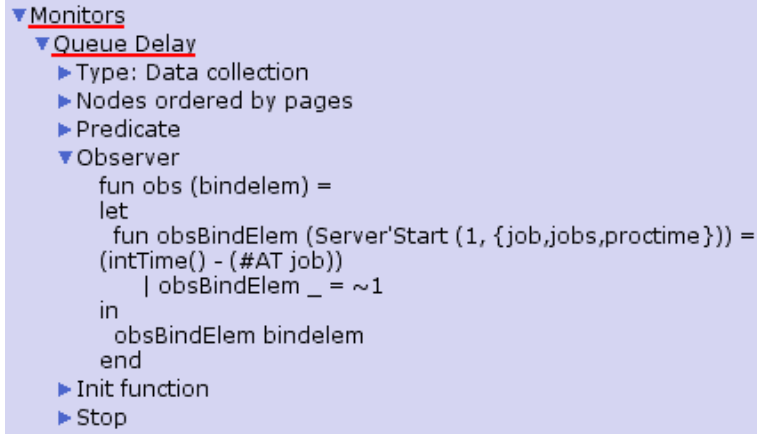


Рис. 11.9. Функция Observer монитора Queue Delay

После запуска программы на выполнение в каталоге с кодом программы появится файл `Queue_Delay.log`, содержащий в первой колонке — значение задержки очереди, во второй — счётчик, в третьей —

шаг, в четвёртой — время. С помощью gnuplot можно построить график значений задержки в очереди, выбрав по оси x время, а по оси y — значения задержки:

```
gnuplot
plot "Queue_Delay.log" using ($4):($1) with lines
quit
```

Посчитаем задержку в действительных значениях. С помощью палитры *Monitoring* выбираем `Data Call` и устанавливаем на переходе `Start`. Появившийся в меню монитор называем `Queue Delay Real`.

Функцию `Observer` изменим следующим образом (рис. 11.10):

```
fun obs (bindelem) =
let
  fun obsBindElem (Server'Start (1, {job, jobs, proctime}))
     = Real.fromInt(intTime() - (#AT job))
     | obsBindElem _ = ~1.0
in
  obsBindElem bindelem
end
```

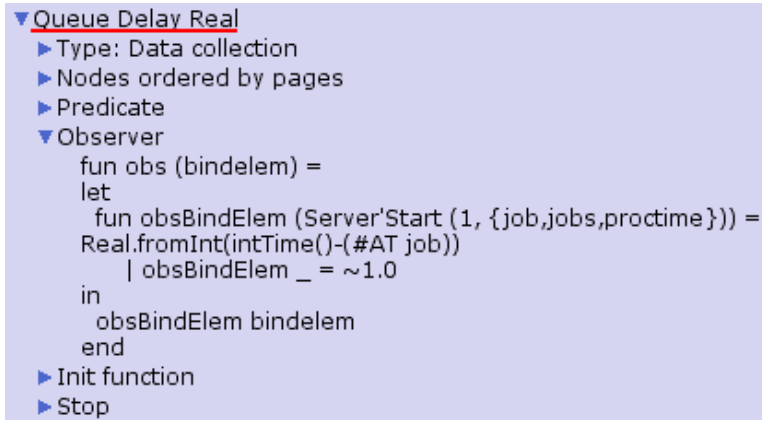


Рис. 11.10. Функция Observer монитора Queue Delay Real

По сравнению с предыдущим описанием функции добавлено преобразование значения функции из целого в действительное, при этом `obsBindElem _` принимает значение `~1.0`.

После запуска программы на выполнение в каталоге с кодом программы появится файл `Queue_Delay_Real.log` с содержимым, аналогичным содержимому файла `Queue_Delay.log`, но значения задержки имеют действительный тип.

Посчитаем, сколько раз задержка превысила заданное значение. С помощью палитры *Monitoring* выбираем `Data Call` и устанавливаем на переходе `Start`. Монитор называем `Long Delay Time`.

Функцию `Observer` изменим следующим образом (рис. 11.11):

```
fun obs (bindelem) =
  if IntInf.tiInt(Queue_Delay.last())>=(!longdelaytime)
  then 1
  else 0
```

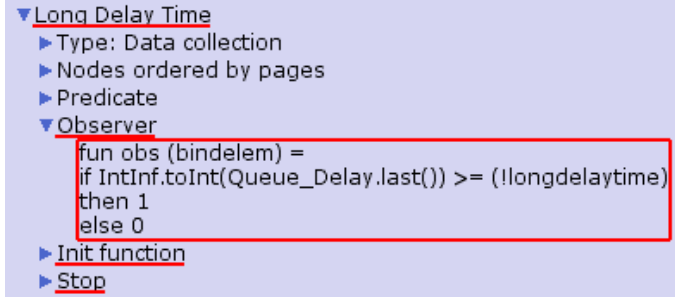


Рис. 11.11. Функция Observer монитора Long Delay Time

Если значение монитора `Queue Delay` превысит некоторое заданное значение, то функция выдаст 1, в противном случае — 0. Восклицательный знак означает разыменование ссылки.

При этом необходимо в декларациях (рис. 11.12) задать глобальную переменную (в форме ссылки на число 200): `longdelaytime`:

```
globref longdelaytime = 200;
```

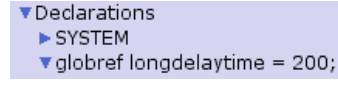


Рис. 11.12. Определение longdelaytime в декларациях

С помощью gnuplot можно построить график, демонстрирующий, в какие периоды времени значения задержки в очереди превышали заданное значение 200:

```
gnuplot
plot [0:][0:1.2] "Long_Delay_Time.log" using ($4):($1) with lines
quit
```

Лабораторная работа 12. Пример моделирования простого протокола передачи данных

12.1. Постановка задачи

Рассмотрим ненадёжную сеть передачи данных, состоящую из источника, получателя.

Перед отправкой очередной порции данных источник должен получить от получателя подтверждение о доставке предыдущей порции данных.

Считаем, что пакет состоит из номера пакета и строковых данных. Передавать будем сообщение «Modelling and Analysis by Means of Coloured Petry Nets», разбитое по 8 символов.

12.2. Построение модели с помощью CPNTools

Основные состояния: источник (Send), получатель (Receiver).

Действия (переходы): отправить пакет (Send Packet), отправить подтверждение (Send ACK).

Промежуточное состояние: следующий посылаемый пакет (NextSend).

Зададим декларации модели:

```
colset INT = int;
colset DATA = string;
colset INTxDATA = product INT * DATA;
var n, k: INT;
var p, str: DATA;
val stop = "########";
```

Состояние `Send` имеет тип `INTxDATA` и следующую начальную маркировку (в соответствии с передаваемой фразой):

```
1`(1,"Modellin")++
1`(2,"g and An")++
1`(3,"alysis b")++
1`(4,"y Means ")++
1`(5,"of Colou")++
1`(6,"red Petr")++
1`(7,"y Nets##")++
1`(8,"########")
```

Стоповый байт (`"########"`) определяет, что сообщение закончилось.

Состояние `Receiver` имеет тип `DATA` и начальное значение `1`""` (т.е. пустая строка, поскольку состояние собирает данные и номер пакета его не интересует). Состояние `NextSend` имеет тип `INT` и начальное значение `1`1`.

Поскольку пакеты представляют собой кортеж, состоящий из номера пакета и строки, то выражение у двусторонней дуги будет иметь значение `(n,p)`.

Кроме того, необходимо взаимодействовать с состоянием, которое будет сообщать номер следующего посылаемого пакета данных. Поэтому переход `Send Packet` соединяем с состоянием `NextSend` двумя дугами с выражениями `n` (рис. 12.1).

Также необходимо получать информацию с подтверждениями о получении данных. От перехода `Send Packet` к состоянию `NextSend` дуга с выражением `n`, обратно — `k`.

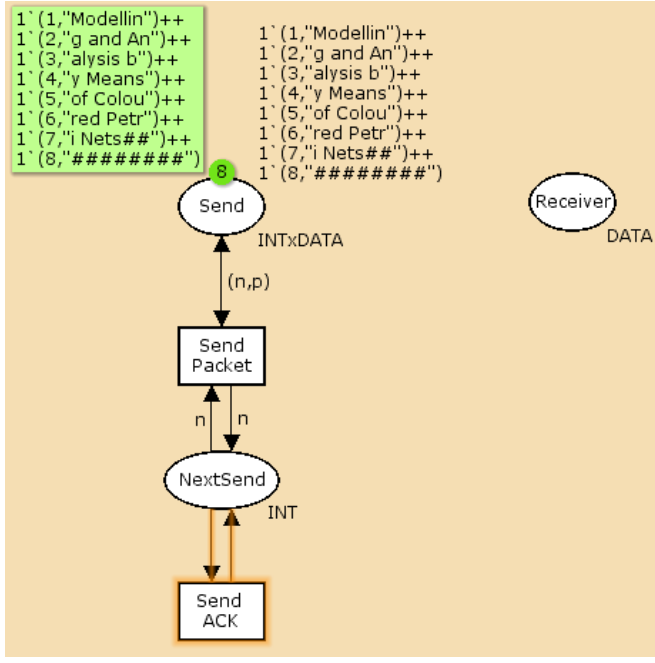


Рис. 12.1. Начальный граф

Зададим промежуточные состояния (`A`, `B` с типом `INTxDATA`, `C`, `D` с типом `INTxDATA`) для переходов (рис. 12.2): передать пакет `Transmit Packet` (передаём `(n,p)`), передать подтверждение

`Transmit ACK` (передаём целое число `k`).

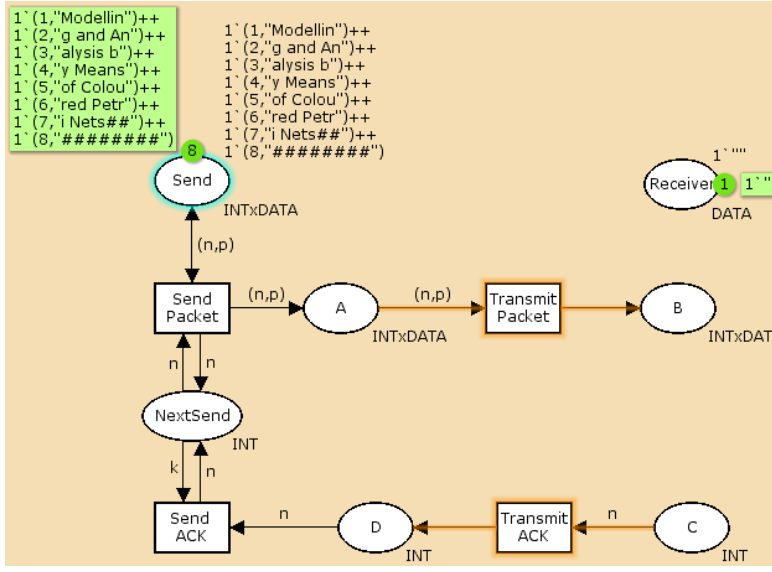


Рис. 12.2. Добавление промежуточных состояний

Добавляем переход получения пакета (`Receive Packet`).

От состояния `Receiver` идёт дуга к переходу `Receive Packet` со значением той строки (`str`), которая находится в состоянии `Receiver`. Обратно: проверяем, что номер пакета новый и строка не равна стоп-биту. Если это так, то строку добавляем к полученным данным.

Кроме того, необходимо знать, каким будет номер следующего пакета. Для этого добавляем состояние `NextRec` с типом `INT` и начальным значением `1`1` (один пакет), связываем его дугами с переходом `Receive Packet`. Причём к переходу идёт дуга с выражением `k`, от перехода — `if n=k then k+1 else k`.

Связываем состояния `B` и `C` с переходом `Receive Packet`. От состояния `B` к переходу `Receive Packet` — выражение `(n,p)`, от перехода `Receive Packet` к состоянию `C` — выражение `if n=k then k+1 else k`.

От перехода `Receive Packet` к состоянию `Receiver`:

```
if n=k andalso p<>stop then str^p else str
```

(если `n=k` и мы не получили стоп-байт, то направляем в состояние строку и к ней прикрепляем `p`, в противном случае посылаем толко строку).

На переходах `Transmit Packet` и `Transmit ACK` зададим потерю пакетов. Для этого на интервале от 0 до 10 зададим пороговое значение и, если передаваемое значение превысит этот порог, то считаем, что произошла потеря пакета, если нет, то передаём пакет дальше. Для этого задаём вспомогательные состояния `SP` и `SA` с типом `Ten0` и начальным значением `1`8`, соединяем с соответствующими переходами.

В декларациях задаём:

```
colset  Ten0 = int with 0..10;
colset  Ten1 = int with 0..10;
var s: Ten0;
var r: Ten1;
```

и определяем функцию (если нет превышения порога, то истина, если нет — ложь):

```
fun Ok(s:Ten0, r:Ten1)=(r<=s);
```

Задаём выражение от перехода `Transmit Packet` к состоянию `B`:

```
if Ok(s,r) then 1`(n,p) else empty
```

Задаём выражение от перехода `Transmit ACK` к состоянию `D`:

```
if Ok(s,r) then 1`n else empty
```

Таким образом, получим модель простого протокола передачи данных (рис. 12.3).

Пакет последовательно проходит: состояние `Send`, переход `Send Packet`, состояние `A`, с некоторой вероятностью переход `Transmit Packet`, состояние `B`, попадает на переход `Receive Packet`, где проверяется номер пакета и если нет совпадения, то пакет направляется в состояние `Received`, а номер пакета передаётся последовательно в состояние `C`, с некоторой вероятностью в переход `Transmit ACK`, далее в состояние `D`, переход `Receive ACK`, состояние `NextSend` (увеличивая на 1 номер следующего пакета), переход `Send Packet`. Так продолжается до тех пор, пока не будут переданы все части сообщения. Последней будет передана стоп-последовательность.

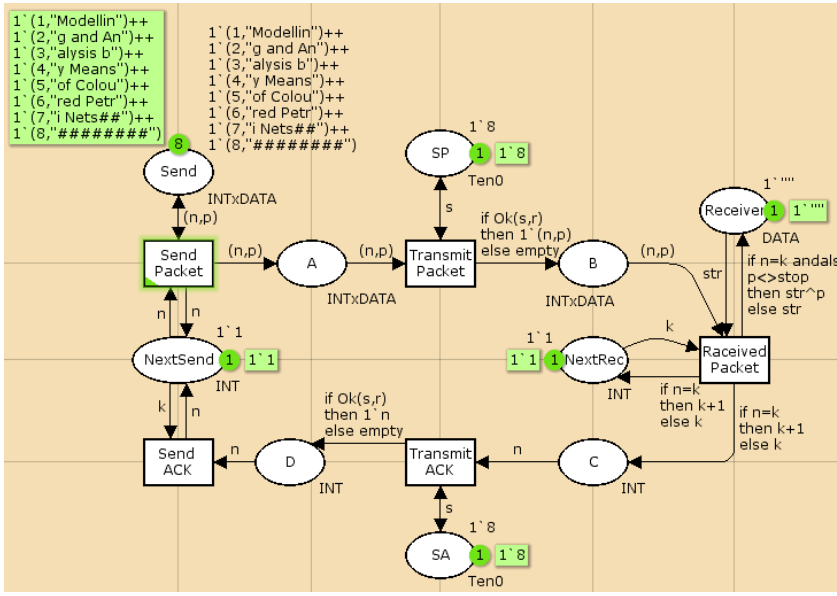


Рис. 12.3. Модель простого протокола передачи данных

Лабораторная работа 13. Задание для самостоятельного выполнения

13.1. Схема модели

Заявка (команды программы, операнды) поступает в оперативную память (ОП), затем передается на прибор (центральный процессор, ЦП) для обработки. После этого заявка может равновероятно обратиться к оперативной памяти или к одному из двух внешних запоминающих устройств (B1 и B2). Прежде чем записать информацию на внешний накопитель, необходимо вторично обратиться к центральному процессору, определяющему состояние накопителя и выдающему необходимую управляющую информацию. Накопители (B1 и B2) могут работать в 3-х режимах:

1) B1 — занят, B2 — свободен;
2) B2 — свободен, B1 — занят;
3) B1 — занят, B2 — занят.

Схема модели представлена на рис. 13.1.

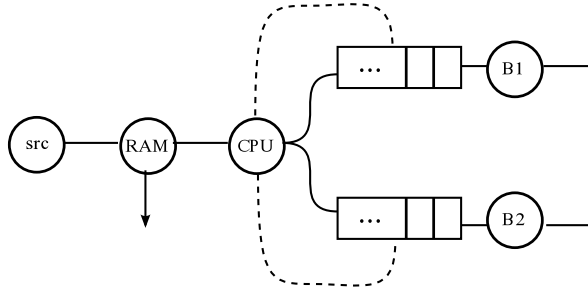


Рис. 13.1. Схема модели для выполнения домашнего задания

На схеме:

- src — источник заявок;
- B1 и B2 — накопители для хранения заявок;
- RAM — оперативная память;
- CPU — центральный процессор;
- B1, B1 — внешние запоминающие устройства.

13.2. Описание модели

Сеть Петри моделируемой системы представлена на рис. 13.2.

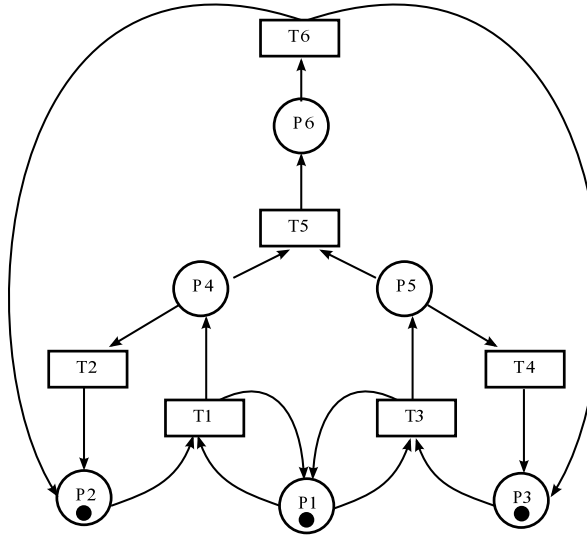


Рис. 13.2. Сеть для выполнения домашнего задания

Множество позиций:

P1 — состояние оперативной памяти (свободна / занята);

P2 — состояние внешнего запоминающего устройства B1 (свободно / занято);

P3 — состояние внешнего запоминающего устройства B2 (свободно / занято);

P4 — работа на ОП и B1 закончена;

P5 — работа на ОП и B2 закончена;

P6 — работа на ОП, B1 и B2 закончена;

Множество переходов:

T1 — ЦП работает только с RAM и B1;

T2 — обрабатываются данные из RAM и с B1 переходят на устройство вывода;

T3 — CPU работает только с RAM и B2;

T4 — обрабатываются данные из RAM и с B2 переходят на устройство вывода;

T5 — CPU работает только с RAM и с B1, B2;

T6 — обрабатываются данные из RAM, B1, B2 и переходят на устройство вывода.

Функционирование сети Петри можно расматривать как срабатывание переходов, в ходе которого происходит перемещение маркеров по позициям:

– работа CPU с RAM и B1 отображается запуском перехода T1 (удаление маркеров из P1, P2 и появление в P1, P4), что влечет за собой срабатывание перехода T2, т.е. передачу данных с RAM и B1 на устройство вывода;
– работа CPU с RAM и B2 отображается запуском перехода T3 (удаление маркеров из P1 и P3 и появление в P1 и P5), что влечет за собой срабатывание перехода T4, т.е. передачу данных с RAM и B2 на устройство вывода;
– работа CPU с RAM, B1 и B2 отображается запуском перехода T5 (удаление маркеров из P4 и P5 и появление в P6), далее срабатывание перехода T6, и данные из RAM, B1 и B2 передаются на устройство вывода;
– состояние устройств восстанавливается при срабатывании: RAM — переходов T1 или T2; B1 — переходов T2 или T6; B2 — переходов T4 или T6.

13.3. Постановка задачи

1. Используя теоретические методы анализа сетей Петри, провести анализ сети, изображённой на рис. 13.2 (с помощью построения дерева достижимости). Определить, является ли сеть безопасной, ограниченной, сохраняющей, имеются ли тупики.
2. Промоделировать сеть Петри (см. рис. 13.2) с помощью CPNTools.

Требования к отчёту

1. Отчёт должен быть аккуратно оформлен: иметь титульный лист с указанием идентифицирующих работу данных; содержать формулировку задачи; иметь единообразный шрифт (основной текст: 13 pt, Times NewRoman, 1,5 интервал, выравнивание по ширине; текст листингов (если требуется): 10 Courier, 1 интервал; заголовки: 14 pt, Times NewRoman).
2. В отчёт включаются описания выполнения всех лабораторных работ раздела и задания для самостоятельного выполнения.
3. Отчёт должен содержать скриншоты разработанных схем CPNTools с пояснениями в тексте на русском языке.
4. Отчёт должен содержать полученные в результате моделирования графики с пояснениями в тексте на русском языке.

Часть IV
Имитационное моделирование в GPSS

IV.1. Теоретические сведения

В разделе использованы материалы из [1, 6, 12, 14].

Пакет GPSS (General Purpose Simulation System — система моделирования общего назначения) предназначен для имитационного моделирования дискретных систем.

Имитационная модель в GPSS представляет собой последовательность текстовых строк, каждая из которых определяет правила создания, перемещения, задержки и удаления транзактов.

Транзакт — динамический объект, отождествляемый с заявкой на обслуживание, который перемещается между элементами системы.

IV.1.1. Объекты GPSS

Типы объектов GPSS:

- *транзакты* — динамические элементы моделируемой системы (заявки, пакеты, требования на обслуживание и т.д.)
- *блоки* — задают логику функционирования имитационной модели системы и определяют пути движения транзактов; обладают основными функциями:
  - создание / уничтожение транзактов;
  - изменение числовых атрибутов блоков и транзактов;
  - задержка транзакта на определённый интервал времени;
  - изменение маршрута движения транзакта.
- *одноканальные устройства (Facility)* — описание устройств, которые в любой момент времени могут обрабатывать лишь один транзакт; обеспечивают сбор основной статистической информации о своем функционировании:
  - текущее состояние устройства;
  - коэффициент использования устройства;
  - общее число вошедших транзактов;
  - среднее время использования устройства одним транзактом;
- *многоканальные устройства (Storage)* — описание устройств, которые могут использоваться несколькими транзактами одновременно; обеспечивают сбор основной статистической информации о своем функционировании:
  - текущее содержимое многоканального устройства;
  - число свободных единиц многоканального устройства;
  - коэффициент использования многоканального устройства;
  - среднее содержимое многоканального устройства;
  - максимальное содержимое многоканального устройства;
  - общее число транзактов, вошедших в многоканальное устройство;

  - среднее время пребывания транзактов в многоканальном устройстве;
  - признак пустоты многоканального устройства;
  - признак заполненности многоканального устройства;
- *логические ключи* — используются для блокировки или изменения направления движения транзактов в зависимости от ранее наступивших в модели событий;
- *арифметические переменные* — позволяют вычислять арифметические выражения;
- *логические (булевы) переменные* — используются для проверки условий функционирования объектов (например, для описания условий, определяющих движение транзактов);
- *функции* — задавать функциональные зависимости между несколькими переменными;
- *очереди* — обеспечивают сбор основной статистической информации о времени задержки транзактов из-за недоступности или занятости устройств:
  - текущая длина очереди;
  - средняя длина очереди;
  - максимальная длина очереди;
  - общее число входов в очередь;
  - число нулевых входов в очередь;
  - среднее время пребывания транзактов в очереди;
  - среднее время пребывания транзактов в очереди, исключая транзакты, прошедшие очередь без ожидания.
- *таблицы* — осуществляют сбор статистической информации о случайных величинах, заданных пользователем:
  - среднее арифметическое значение элементов таблицы;
  - общее число элементов в таблице;
  - среднеквадратическое отклонение элементов таблицы;
- *ячейки* — используются для сохранения некоторой числовой информации (например, длина очереди в какой-то конкретной точке модели);
- *матрицы ячеек* — используются для сохранения некоторой числовой информации;
- *списки пользователя* — позволяют организовать работу с очередями, дисциплина обслуживания в которых отличается от FIFO.

IV.1.2. Основные операторы и команды GPSS

Оператор — текстовая строка описания:
- объектов исследуемой системы;
- блоков модели, которые имитируют функционирование элементов системы;

– команд управления моделированием.

Строка оператора GPSS состоит из разделённых пробелами полей:

– *номер строки* — любое десятичное число из семи символов, в том числе и дробное;
– *метка* — содержимое зависит от типа оператора (например, имя объекта в операторах описания объектов)
– *операция* — символическое обозначение оператора;
– *операнд* — значение различно для разных операторов;
– *комментарий* — информация, поясняющая назначение оператора.

IV.1.2.1. Команды управления моделированием

Условия завершения процесса моделирования задаются командами `SIMULATE` и `START`:

```
SIMULATE [A]
```

где `A` — число минут реального времени, по истечении которого моделирование будет завершено;

```
START A
```

где `A` — начальное значение счетчика завершений.

Команда `RESET` обнуляет всю собранную статистику и значение относительного модельного времени.

Команда `CLEAR` дополнительно обнуляет значение абсолютного модельного времени, инициализирует генераторы случайных чисел и удаляет из модели все имеющиеся транзакты.

Команда `HALT` немедленно прерывает процесс моделирования, переводя его в приостановленное состояние и удаляя оставшиеся команды из очереди команд.

Команда `CONTINUE` позволяет продолжить моделирование после приостановки.

Команда `STEP` задает условие прерывания процесса моделирования при прохождении транзактами заданного числа блоков:

```
STEP A
```

Здесь `A` — положительное целое число пройденных транзактами блоков.

Команда `STOP` устанавливает или снимает условие останова моделирования:

```
STOP [A],[B]
```

Здесь `A` — номер транзакта, удовлетворяющего условию останова; `B` — номер или метка блока, удовлетворяющего условию останова.

Команда `STOP` без операндов вызывает немедленный останов процесса моделирования, который можно продолжить командой `CONTINUE`.

IV.1.2.2. Работа с транзактами

За создание и моделирование поступления транзактов в систему отвечает блок GENERATE:

```
GENERATE A,[B],[C],[D],[E],[F],[G],[H],[I]
```

Здесь A — среднее значение интервала времени между моделируемыми транзактами (по умолчанию — 0); B — величина разброса возможных значений времени; C — модельное время генерации первого транзакта; D — максимальное количество моделируемых транзактов; E — приоритет транзактов (по умолчанию — 0, т.е. самый низкий приоритет); F,..., I —количество и формат параметров транзактов (по умолчанию — 12).

Пример 1. Конструкция

```
GENERATE 10,2,5,,2
```

задаёт моделирование транзактов через интервалы времени, равномерно распределённые на отрезке $[8, 12]$ (или 10 ± 2), причём первый транзакт появляется через 5 единиц модельного времени после начала моделирования. Кроме того, общее число моделируемых транзактов не ограничено и все транзакты имеют приоритет 2 и 12 параметров.

Пример 2. Конструкция

```
GENERATE 75,FN$EXPON,,20,,3PB
```

задаёт моделирование транзактов через интервалы времени, имеющие экспоненциальное распределение со средним значением 75 единиц; первый транзакт моделируется в нулевой момент модельного времени; генерируется только 20 транзактов с нулевым приоритетом; каждый транзакт имеет по 3 параметра форматом «полуслово», т.е. способных принимать значения от -255 до 255.

Блок TERMINATE используется для удаления транзактов из модели:

```
TERMINATE [A]
```

Здесь A — указывает число (по умолчанию 0), на которое уменьшается содержимое счетчика завершений.

Пример 3. Конструкция

```
TERMINATE
```

задаёт уничтожение транзакта, поступившего в блок; значение счетчика завершений не изменяется.

Конструкция

```
TERMINATE 1
```

задаёт уничтожение транзакта, поступившего в блок; значение счетчика завершений изменяется на 1.

Блок ADVANCE используется для задания задержки транзакта:

```
ADVANCE A,[B]
```

Здесь A — среднее значение интервала времени между моделируемыми транзактами (по умолчанию — 0); B — величина разброса возможных значений времени.

Пример 4. Конструкция

```
ADVANCE 30,5
```

задаёт моделирование задержки транзактов в течение времени, которое имеет равномерное распределение на отрезке $[25, 35]$ (или 30 ± 5).

Блок ASSIGN используется для изменения значений параметров транзакта:

```
ASSIGN A,B,[C],[D]
```

Здесь A — номер изменяемого параметра с указанием режима изменения: накопление (+), вычитание (–), замещение (без дополнительных символов); B — число, изменяющее значение параметра; C — имя функции, применяемой для модификации значения параметра; D — формат изменяемого параметра: PF, PH, PB или PL (по умолчанию — PH).

Пример 5. Конструкция

```
ASSIGN 3+,5,,PB
```

задаёт увеличение значения параметра 3 форматом «байт» на 5 единиц.

Конструкция

```
ASSIGN 2-6,5.75,,PL
```

задаёт параметрам 2–6 (форматом «плавающая точка») значение $5,75$.

Блок PRIORITY используется для изменения приоритета транзакта:

```
PRIORITY A
```

Здесь A — указывает новое значение приоритета транзакта, вошедшего в блок (от 0 до 127 включительно).

Блок SPLIT используется для создания копий транзакта:

```
SPLIT A,[B],[C]
```

Здесь A — число создаваемых копий; B — метка блока, к которому отправляются копии исходного транзакта; C — номер параметра, используемого для присвоения копиям последовательных номеров.

Блок ASSEMBLE используется для объединения определенного числа транзактов одного семейства:

```
ASSEMBLE A
```

Здесь A — число объединяемых транзактов.

Для определения времени перемещения транзакта между двумя произвольными точками модели используется блок MARK:

```
MARK A
```

Здесь A — номер параметра транзакта, в который записывается текущее значение абсолютного модельного времени.

IV.1.2.3. Обслуживающие устройства

Блок SEIZE моделирует занятие транзактом одноканального устройства, блок RELEASE предназначен для освобождения устройства:

```
SEIZE A
RELEASE A
```

Здесь A — имя устройства, занимаемого (освобождаемого) транзактом.

Блок PREEMPT применяется для моделирования работы одноканальных устройств с прерываниями:

```
PREEMPT A,B,[C],D,E
```

Здесь A — имя устройства; B — режим прерывания (обычный или PR — по приоритету); C — метка блока, в который направляется транзакт, обслуживание которого было прервано; D — номер параметра прерванного транзакта, в который заносится остаток времени обслуживания; E — если в поле записывается «RE», то транзакт, обслуживание которого было прервано, не претендует на завершение своего обслуживания в устройстве.

Транзакт, вошедший в блок RETURN, снимает прерывание на устройстве, вызванное вхождением данного транзакта в блок PREEMPT:

```
RETURN A
```

Здесь A — имя устройства, с которого снимается прерывание.

Блоки ENTER и LEAVE предназначены соответственно для занятия и освобождения многоканального устройства:

```
ENTER A,[B]
LEAVE A,[B]
```

Здесь A — имя многоканального устройства; B — число занимаемых (освобождаемых) приборов многоканального устройства.

Оператор STORAGE необходим для указания количества приборов в многоканальном устройстве:

```
A STORAGE B
```

Здесь A — имя многоканального устройства; B — число приборов в устройстве.

IV.1.2.4. Очереди

Блок QUEUE применяется для сбора и обработки статистики по очередям, блок DEPART освобождает требуемое число мест в очереди:

```
QUEUE A,[B]
DEPART A,[B]
```

Здесь A — имя очереди; B — число мест в очереди, занимаемых (освобождаемых) транзактом.

IV.1.2.5. Управление перемещением транзактов

Блок TRANSFER изменяет маршрут движения транзактов:

```
TRANSFER [A],B,[C],[D]
```

Здесь A — режим перехода; B — метка первого альтернативного блока; C — метка второго альтернативного блока; D — константа, используемая для относительной переадресации транзактов.

Пример 6. Конструкция

```
TRANSFER 0.6,dst2,dst1
```

задаёт перемещение транзакта к блоку с меткой dst1 с вероятностью $0,6$ и к блоку с меткой dst2 с вероятностью $0,4$.

Блок TEST определяет направление движения транзакта в зависимости от выполнения условия, заданного алгебраическим соотношением:

```
TEST XX A,B,[C]
```

Здесь XX — знак логической операции: L — меньше, G — больше, E — равно, LE — меньше или равно, GE — больше или равно, NE — не равно; A, B — сравниваемые значения; C — метка блока, куда перемещается транзакт в случае невыполнения заданного условия.

Блок GATE разрешает движение транзактам при определённом состоянии оборудования:

```
GATE XXX A,[B]
```

Здесь XXX — логический указатель:

- для одноканального устройства:
  - FV — устройство занято;
  - FNV — устройство не занято;
  - FI — устройство обслуживает прерывание;
  - FNI — устройство не обслуживает прерывание;
- для многоканального устройства
  - SF — устройство заполнено;
  - SNF — устройство не заполнено;
  - SE — устройство пусто;
  - SNE — устройство не пусто;
- для логического ключа:
  - LR — логический переключатель сброшен;
  - LS — логический переключатель установлен;

A — имя или номер устройства; B — метка альтернативного блока. При выполнении условия, записанного в логическом указателе XXX, транзакт переходит в следующий за GATE блок. В противном случае он направляется в блок с меткой, содержащейся в операнде B.

Блок LOGIC используется для изменения значений логических ключей в модели:

```
LOGIC X A
```

Здесь A — имя или номер логического ключа; X — указатель операции с логическим ключом: S — установить (единица), R — сбросить (обнулить), I — инвертировать.

Блок LOOP используется для организации циклов перемещения транзактов:

```
LOOP A,B
```

Здесь A — номер параметра транзакта, используемого в качестве счетчика цикла с указанием формата: «слово» (PF), «полуслово» (PH), «байт» (PB); B — метка блока, являющегося начальным в повторяющейся группе блоков.

Для синхронизации движения двух транзактов из семейства используются два сопряженных блока MATCH.

Пример 7. Конструкция

```
LABEL1 MATCH LABEL2
```

определяет, что транзакт, вошедший в блок с меткой LABEL1, будет ожидать в этом блоке прихода транзакта того же семейства в блок с меткой LABEL2.

IV.1.2.6. Функции и переменные

Арифметические переменные, арифметические переменные с плавающей точкой и булевские переменные определяются с помощью соответствующих операторов:

```
N VARIABLE A
N FVARIABLE A
N BVARIABLE A
```

Здесь N — имя или номер переменной; A — арифметическое или логическое выражение.

Операции отношения:

'G' — больше;
'L' — меньше;
'E' — равно;
'NE' — не равно;
'LE' — меньше или равно;
'GE' — больше или равно.

Арифметические операции:

^ — возведение в степень;
# — умножение;
/ — деление;
\ — деление нацело;
@ — деление по модулю;
+ — сложение;
- — вычитание.

Логические операции:

'NOT' — логическое отрицание;
'AND' — логическое умножение;
'OR' — логическое сложение.

Стандартные функции:

ABS(•) — абсолютное значение;
ATN(•) — арктангенс в радианах;
COS(•) — косинус в радианах;
INT(•) — целая часть;
EXP(•) — экспонента;
LOG(•) — натуральный логарифм;
SIN(•) — синус в радианах;
SQR(•) — квадратный корень;
TAV(•) — тангенс в радианах.

Определение функций выполняется в GPSS с помощью оператора

```
N FUNCTION A,B
```

Здесь N — имя или номер функции; A — аргумент функции; B — указатель типа функции (в частности, D — для дискретной, C — для непрерывной) и числа точек табуляции. За оператором описания функции следует описание множества значений аргумента и функции, которые отделяются друг от друга символом «/».

Пример 8. Конструкция

```
TimeBetweenTrains FVARIABLE (ABS(1.4\Q$Och1-3))
```

определяет вещественную переменную, зависящую от длины очереди Och1.

Пример 9. Конструкция

```
CountAll VARIABLE F$Shlyuz+Q$VerhOtch
```

определяет переменную, значение которой равно числу транзактов на обслуживающем приборе Shlyuz и в очереди VerhOtch.

Пример 10. Конструкция

```
ProcentNaloga FUNCTION Zarplata,D3
6.1,10/9.2,13/12.3,20
```

задаёт дискретную функцию с тремя значениями: 10, 13, 20.

Пример 11. Конструкция

```
ASSIGN 1,FN$ProcentNaloga
```

присваивает первому параметру текущего транзакта вновь вычисленное значение функции ProcentNaloga.

IV.1.3. Моделирование случайных величин

IV.1.3.1. Непрерывные случайные величины

Моделирования случайной величины, имеющей равномерное распределение:

```
UNIFORM(Stream,Min,Max)
```

Здесь `Stream` — номер генератора случайных чисел (от 1 до 7); `Min` — наименьшее возможное значение; `Max` — наибольшее возможное значение.

Пример 12. Два способа задания генерации транзактов через интервалы времени, равномерно распределённые на отрезке $[2; 8]$ и задержки транзактов на время, имеющее равномерное распределение на отрезке $[3; 5]$:

```
GENERATE (UNIFORM(4,2,8))
ADVANCE (UNIFORM(2,3,5))
```

или

```
GENERATE 5,3
ADVANCE 4,1
```

Моделирование случайной величины, имеющей экспоненциальное распределение:

```
EXPONENTIAL(Stream,Locate,Scale)
```

Здесь `Stream` — номер генератора случайных чисел (от 1 до 7); `Locate` — величина сдвига (константа, добавляемая к значению моделируемой величины); `Scale` — параметр формы распределения (математическое ожидание случайной величины при `Locate =0`).

Пример 13. Генерация транзактов через интервалы времени, имеющие экспоненциальное распределение с математическим ожиданием, равным 4, со сдвигом на 2 единицы вправо:

```
40 GENERATE (Exponential(1,2,4))
```

Моделирование случайной величины, имеющей гамма-распределение:

```
GAMMA(Stream,Locate,Scale,Shape)
```

Здесь `Stream` — номер генератора случайных чисел (от 1 до 7); `Locate` — величина сдвига; `Scale` — параметр масштаба функции распределения; `Shape` — параметр, определяющий форму кривой гамма-распределения.

Моделирование случайной величины, подчиняющейся нормальному закону распределения:

```
NORMAL(Stream,Mean,StdDev)
```

Здесь `Stream` — номер генератора случайных чисел (от 1 до 7); `Mean` — математическое ожидание; `StdDev` — среднеквадратическое отклонение.

IV.1.3.2. Дискретные случайные величины

Моделирование дискретной случайной величины, имеющей биномиальный закон распределения:

`BINOMIAL(Stream,TrialCount,Probability)`

Здесь `Stream` — номер генератора случайных чисел (от 1 до 7); `Locate` — величина сдвига; `TrialCount` — число испытаний Бернулли; `Probability` — вероятность успеха в каждом испытании.

Моделирование дискретной случайной величины, имеющей пуассоновский закон распределения

`POISSON(Stream,Mean)`

Здесь `Stream` — номер генератора случайных чисел (от 1 до 7); `Mean` — математическое ожидание.

IV.1.4. Стандартные числовые атрибуты

Системные стандартные числовые атрибуты:

- `RNj` — число, возвращаемое j-м датчиком случайных чисел;
- `C1` — текущее значение относительного модельного времени;
- `AC1` — текущее значение абсолютного модельного времени;
- `TG1` — текущее значение счетчика завершений;
- `XN1` — номер активного транзакта (обрабатываемого в данный момент);
- `M1` — время пребывания в модели активного транзакта.

Стандартные числовые атрибуты транзактов:

- `Pj` — значение j-го параметра текущего транзакта;
- `PFj` — значение j-го параметра текущего транзакта форматом «слово»;
- `PHj` — значение j-го параметра текущего транзакта форматом «полуслово»;
- `PBj` — значение j-го параметра текущего транзакта форматом «байт»;
- `PLj` — значение j-го параметра текущего транзакта форматом «плавающая точка»;
- `PR` — приоритет активного транзакта;
- `MPj` — значение, равное разности абсолютного модельного времени, и значения j-го параметра текущего транзакта;
- `MBj` — флаг синхронизации: 1, если транзакт в блоке j принадлежит тому же семейству, что и текущий транзакт; 0 — в противном случае.

Стандартные числовые атрибуты блоков:

- `Wj` — количество транзактов, находящихся в блоке с номером j в текущий момент модельного времени;

– Nj — общее количество транзактов, поступивших в блок с номером j.

Стандартные числовые атрибуты одноканальных обслуживающих устройств:
– Fj — текущее состояние устройства j: 1, если устройство занято; иначе — 0;
– FRj — коэффициент использования устройства в тысячных долях;
– FCj — общее число транзактов, вошедших в устройство j;
– FTj — среднее время использования устройства одним транзактом.

Стандартные числовые атрибуты многоканальных обслуживающих устройств:
– SAj — среднее содержимое многоканального устройства j (целая часть);
– SMj — максимальное содержимое многоканального устройства j;
– SCj — общее число транзактов, вошедших в многоканальное устройство j;
– STj — среднее время пребывания транзактов в многоканальном устройстве j;
– SEj — признак пустоты многоканального устройства j: 1 — пусто, 0 — в противном случае;
– SFj — признак заполненности многоканального устройства j: 1 — заполнено, 0 — в противном случае.

Стандартные числовые атрибуты логических ключей:
– LSj — состояние логического ключа с номером j: 1 — включен, 0 – выключен.

Стандартные числовые атрибуты переменных и функций:
– Vj — значение арифметической переменной j;
– BVj — значение логической переменной j (1 — истина, 0 — ложь);
– FNj — значение функции j (дробная часть отбрасывается за исключением использования в качестве аргумента другой функцией).

IV.1.5. Анализ результатов моделирования в GPSS

Выходной файл статистики состоит из подразделов, содержащих стандартную статистику об объектах GPSS. Файл статистики начинается с заголовка, который содержит имя модели, дату и время моделирования. Далее следует выходная информация, содержащая основные сегменты вывода:
– о результатах работы модели:
  – START TIME — абсолютное модельное время в момент начала моделирования;
  – END TIME — абсолютное время или момент, когда счетчик завершений принял значение 0;

  - BLOCKS — количество блоков, использованных в текущей модели, к моменту завершения моделирования;
  - FACILITIES — количество одноканальных устройств, использованных в модели к моменту завершения моделирования;
  - STORAGES — количество многоканальных устройств, использованных в текущей модели к моменту завершения моделирования;
- об именах:
  - NAME — содержит имена, используемые в программе модели;
  - VALUE — определяет числовое значение (номер), соответствующее имени, устанавливает начальный номер GPSS равным 10000;
- о блоках текущей модели:
  - LABEL — метка оператора, связанного с блоком GPSS;
  - LOC — номер строки модели, связанной с блоком;
  - BLOCK TYPE — тип блока GPSS;
  - ENTRY COUNT — количество транзактов, вошедших в данный блок после последнего выполнения команд RESET или CLEAR или с начала процедуры моделирования;
  - CURRENT COUNT — количество транзактов, находящихся в данном блоке и ожидающих выполнения некоторых условий;
  - RETRY — количество транзактов, ожидающих выполнения некоторых условий;
- об одноканальных устройствах:
  - FACILITY — номер или имя одноканального устройства;
  - ENTRIES — количество транзактов, вошедших в устройство после последнего выполнения команды RESET или CLEAR или начала работы программы;
  - UTIL. — часть периода моделирования, в течение которого устройство было занято (коэффициент загрузки);
  - AVE. TIME — среднее время занятости устройства одним транзактом в течение процедуры моделирования после последнего выполнения команд CLEAR или RESET; AVAILABLE — состояние готовности устройства в конце периода моделирования;
  - OWNER — номер последнего транзакта, занимавшего устройство (0 означает, что устройство не занималось);
  - PEND — количество транзактов, ожидающих устройство (находящееся в режиме прерывания);
  - INTER — количество транзактов, обработка которых прервана на устройстве в данный момент модельного времени;
  - RETRY — количество транзактов, ожидающих выполнения некоторых условий;
  - DELAY — количество транзактов, ожидающих занятия устройства (включая транзакты, ожидающие освобождение устройства в режиме прерывания);

- об очередях:
  - QUEUE — имя или номер объекта типа «очередь»;
  - MAX — максимальное содержимое объекта типа «очередь» в течение периода моделирования, который начинается с начала работы или с последней команды RESET или CLEAR;
  - CONT — текущее содержимое объекта типа «очередь» в момент завершения моделирования;
  - ENTRIES — общее количество входов в очередь в течение периода моделирования (счетчик входов);
  - ENTRIES(0) — общее количество входов в очередь с нулевым временем ожидания (счетчик «нулевых» входов);
  - AVE.CONT — среднее значение длины очереди;
  - AVE.TIME — среднее время, проведенное транзактом в очереди с учётом всех входов в очередь;
  - AVE.(-0) — среднее время, проведенное транзактом в очереди без учета «нулевых» входов в очередь;
  - RETRY — количество транзактов, ожидающих специальных условий, зависящих от состояния объекта типа «очередь»;
- о многоканальных устройствах:
  - STORAGE — имя или номер многоканального устройства;
  - CAP. — количество каналов, заданное оператором STORAGE;
  - REMAIN — число свободных каналов в конце периода моделирования;
  - MIN — минимальное количество использовавшихся каналов за период моделирования;
  - MAX — максимальное количество использовавшихся каналов за период моделирования;
  - ENTRIES — количество входов в многоканальное устройство за период моделирования;
  - AVL. — состояние готовности многоканального устройства в конце периода моделирования;
  - AVE.C. — среднее число занятых каналов в устройстве за весь период моделирования;
  - UTIL. — часть периода моделирования, в течение которого многоканальное устройство использовалось;
  - RETRY — количество транзактов, ожидающих специальные условия, зависящие от состояния устройства;
  - DELAY — количество транзактов, ожидающих возможность входа в блок ENTER;
- о ячейках памяти:
  - SAVEVALUE — имя или номер ячейки;
  - VALUE — значение ячейки в конце моделирования;
  - RETRY — количество транзактов, ожидающих наступления специальных условий, зависящих от состояния ячейки;
- список будущих событий:

- XN — номер транзакта, ожидающего выполнения некоторого события;
- PRI — приоритет транзакта;
- BDT — время назначенного события, связанного с данным транзактом;
- ASSEM — номер семейства транзактов;
- CURRENT — номер блока, в котором находится транзакт (0, если транзакт еще не вошел ни в один из блоков);
- NEXT — номер блока, в который должен войти транзакт;
- PARAMETER — номер или имя параметра транзакта;
- VALUE — значение параметра.

Лабораторная работа 14. Модели парикмахерской

14.1. Модель обслуживания клиентов одним парикмахером

14.1.1. Постановка задачи

Интервалы прихода клиентов в парикмахерскую с одним креслом распределены равномерно на интервале 18 ± 6 мин. Время стрижки также распределено равномерно на интервале 16 ± 4 мин. Клиенты приходят в парикмахерскую, стригутся в порядке очереди: «первым пришёл — первым обслужился». Необходимо построить GPSS-модель парикмахерской, которая должна обеспечить сбор статистических данных об очереди. Промоделируйте работу парикмахерской в течение 8 часов.

14.1.2. Построение модели

Порядок блоков в модели соответствует порядку фаз, в которых клиент оказывается при движении в реальной системе:
1) клиент приходит;
2) если необходимо, ждет своей очереди;
3) садится в кресло парикмахера;
4) парикмахер обслуживает клиента;
5) клиент уходит из парикмахерской.

Модель будет состоять из двух частей: моделирование обслуживания клиентов в парикмахерской и задание времени моделирования.

Для задания равномерного распределения прихода клиентов используем блок `GENERATE`, для задания равномерного времени обслуживания (задержки в системе) — `ADVANCE`. Для моделирования ожидания клиентов в очереди используем блоки `QUEUE` и `DEPART`, в которых в качестве имени очереди укажем `barberq`. Для моделирования посещения клиентом кресла парикмахера используем блоки `SEIZE` и `RELEASE` с параметром `barber` — имени «устройства обслуживания».

Таким образом, имеем:

```
;barber
GENERATE  18,6
QUEUE     barberq
SEIZE     barber
DEPART    barberq
ADVANCE   16,4
```

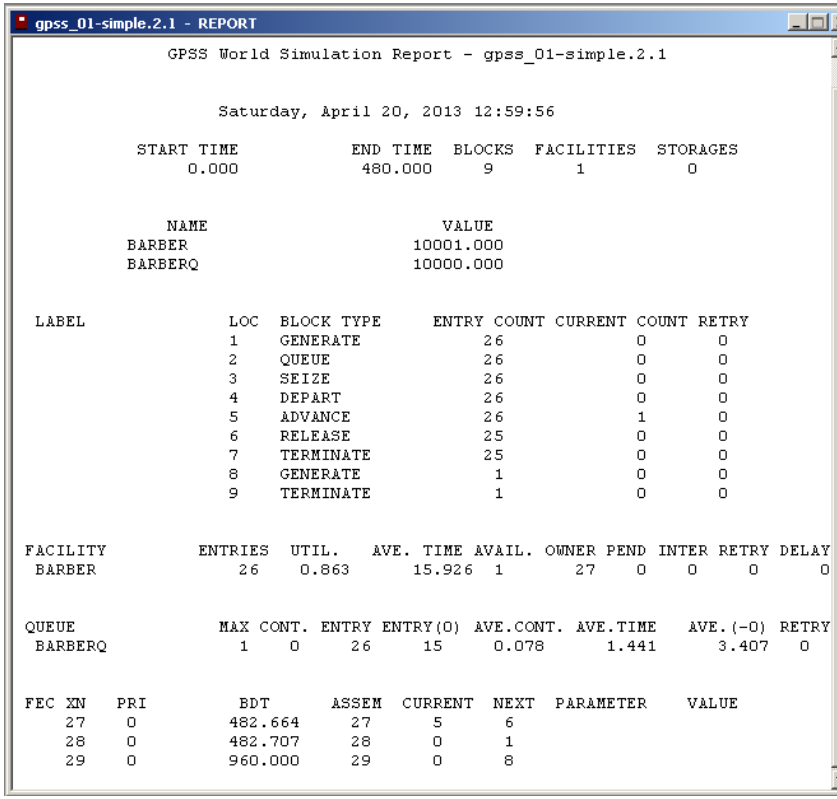
```
RELEASE   barber
TERMINATE 0
```

Требуется, чтобы модельное время было 8 часов. Соответственно, параметр блока `GENERATE` — 480 (8 часов, умноженное на 60 минут). Работа программы начинается с оператора `START` с начальным значением счётчика завершений, равным 1; заканчивается — оператором `TERMINATE` с параметром 1, что задаёт ординарность потока в модели.

Таким образом, имеем:

```
;timer
GENERATE 480
TERMINATE 1
START 1
```

После запуска симуляции получаем отчёт (рис. 14.1).



gpss_01-simple.2.1 - REPORT

```
              GPSS World Simulation Report - gpss_01-simple.2.1


                    Saturday, April 20, 2013 12:59:56

           START TIME            END TIME  BLOCKS  FACILITIES  STORAGES
                0.000             480.000     9        1          0


              NAME                       VALUE
          BARBER                      10001.000
          BARBERQ                     10000.000


 LABEL              LOC  BLOCK TYPE     ENTRY COUNT CURRENT COUNT RETRY
                    1    GENERATE            26             0       0
                    2    QUEUE               26             0       0
                    3    SEIZE               26             0       0
                    4    DEPART              26             0       0
                    5    ADVANCE             26             1       0
                    6    RELEASE             25             0       0
                    7    TERMINATE           25             0       0
                    8    GENERATE             1             0       0
                    9    TERMINATE            1             0       0


FACILITY          ENTRIES  UTIL.   AVE. TIME AVAIL. OWNER PEND INTER RETRY DELAY
 BARBER             26     0.863      15.926  1       27    0     0     0      0


QUEUE              MAX CONT. ENTRY ENTRY(0) AVE.CONT. AVE.TIME   AVE.(-0) RETRY
 BARBERQ             1    0     26      15     0.078      1.441      3.407   0


FEC XN   PRI         BDT      ASSEM  CURRENT  NEXT  PARAMETER    VALUE
    27    0         482.664     27      5      6
    28    0         482.707     28      0      1
    29    0         960.000     29      0      8
```

Рис. 14.1. Отчёт по модели обслуживания клиентов в парикмахерской

Результаты работы модели:

- модельное время в начале моделирования: START TIME=0.0;
- абсолютное время или момент, когда счетчик завершений принял значение 0: END TIME=480.0;
- количество блоков, использованных в текущей модели, к моменту завершения моделирования: BLOCKS=9;
- количество одноканальных устройств, использованных в модели к моменту завершения моделирования: FACILITIES=1;
- количество многоканальных устройств, использованных в текущей модели к моменту завершения моделирования: STORAGES=0.

Имена, используемые в программе модели: barber, barberq.

Далее идёт информация о блоках текущей модели, в частности, ENTRY COUNT — количество транзактов, вошедших в блок с начала процедуры моделирования.

Затем идёт информация об одноканальном устройстве FACILITY (кресло парикмахера), откуда видим, что к парикмахеру попало 27 клиентов (значение поля OWNER=27), но одного клиента парикмахер не успел обслужить (значение поля ENTRIES=26). Полезность его работы составила $0,863$. При этом среднее время занятости парикмахера составило $15,926$ мин.

Далее информация об очереди:

- QUEUE=barberq — имя объекта типа «очередь»;
- MAX=1 — в очереди находилось не более одного ожидающего клиента;
- CONT=0 — на момент завершения моделирования очередь была пуста;
- ENTRIES=26 — общее число клиентов, прошедших через очередь в течение периода моделирования;
- ENTRIES(0)=15 — число клиентов, попавших к парикмахеру без ожидания в очереди;
- AVE.CONT=$0,078$ клиентов в среднем были в очереди;
- AVE.TIME=1.441 минут в среднем клиенты провели в очереди (с учётом всех входов в очередь);
- AVE.(-0)=$3,407$ минут в среднем клиенты провели в очереди (без учета «нулевых» входов в очередь).

В конце отчёта идёт информация о будущих событиях:

- XN=27 — порядковый номер клиента, ожидающего прихода в парикмахерскую;
- PRI=0 — все клиенты равноправны;
- BDT=$482,664$ — время назначенного события, связанного с данным транзактом;
- ASSEM=27 — номер семейства транзактов;
- CURRENT=5 — номер блока, в котором находится транзакт;
- NEXT=6 — номер блока, в который должен войти транзакт.

14.2. Модель обслуживания двух типов клиентов в парикмахерской

14.2.1. Постановка задачи

В парикмахерскую с одним креслом приходят клиенты двух типов. Клиенты первого типа желают только стричься. Распределение интервалов их прихода — 35 ± 10 мин. Клиенты второго типа желают постричься и побриться. Распределение интервалов их прихода — 60 ± 20 мин. Парикмахер обслуживает клиентов в порядке «первым пришел — первым обслужился». Время, затраченное на стрижку, составляет 18 ± 6 мин, а на бритье — 10 ± 2 мин.

Написать GPSS-модель парикмахерской, обеспечив сбор данных об очереди клиентов.

14.2.2. Построение модели

Необходимо реализовать отличие в обслуживании клиентов, которые только стригутся, и клиентов, которые стригутся и бреются. Такую систему можно промоделировать с помощью двух сегментов. Один из них моделирует обслуживание только стригущихся клиентов, а второй — стригущихся и бреющихся. В каждом из сегментов пара `QUEUE`–`DEPART` должна описывать одну и ту же очередь, а пара блоков `SEIZE`–`RELEASE` должна описывать в каждом из двух сегментов одно и то же устройство и моделировать работу парикмахера.

Сегмент моделирования обслуживания клиентов первого типа:

```
;haircut
GENERATE  35,10
QUEUE     barberq
SEIZE     barber
DEPART    barberq
ADVANCE   18,6
RELEASE   barber
TERMINATE 0
```

Сегмент моделирования обслуживания клиентов второго типа:

```
;haircut and shaving
GENERATE  60,20
QUEUE     barberq
SEIZE     barber
DEPART    barberq
ADVANCE   10,2
ADVANCE   18,6
RELEASE   barber
TERMINATE 0
```

Сегмент моделирования таймера:

```
;timer
GENERATE 480
TERMINATE 1
START 1
```

После запуска симуляции получаем отчёт (рис. 14.2).

gpss_02-two_types.3.1 - REPORT

```
              GPSS World Simulation Report - gpss_02-two_types.3.1


                   Saturday, April 20, 2013 15:02:24

          START TIME            END TIME  BLOCKS  FACILITIES  STORAGES
               0.000             480.000    17        1          0


             NAME                       VALUE
          BARBER                     10001.000
          BARBERQ                    10000.000


 LABEL              LOC  BLOCK TYPE     ENTRY COUNT CURRENT COUNT RETRY
                     1    GENERATE            11              0       0
                     2    QUEUE               11              0       0
                     3    SEIZE               11              0       0
                     4    DEPART              11              0       0
                     5    ADVANCE             11              0       0
                     6    RELEASE             11              0       0
                     7    TERMINATE           11              0       0
                     8    GENERATE             8              0       0
                     9    QUEUE                8              0       0
                    10    SEIZE                8              0       0
                    11    DEPART               8              0       0
                    12    ADVANCE              8              1       0
                    13    ADVANCE              7              0       0
                    14    RELEASE              7              0       0
                    15    TERMINATE            7              0       0
                    16    GENERATE             1              0       0
                    17    TERMINATE            1              0       0


FACILITY          ENTRIES  UTIL.   AVE. TIME AVAIL. OWNER PEND INTER RETRY DELAY
 BARBER             19    0.828      20.909  1       20    0    0     0       0


QUEUE               MAX CONT. ENTRY ENTRY(0) AVE.CONT. AVE.TIME   AVE.(-0) RETRY
 BARBERQ              2    0     19       5     0.373      9.419     12.783   0


FEC XN   PRI         BDT      ASSEM  CURRENT  NEXT  PARAMETER    VALUE
    20    0         484.010     20     12     13
    21    0         486.330     21      0      1
    22    0         501.796     22      0      8
    23    0         960.000     23      0     16
```

Рис. 14.2. Отчёт по модели обслуживания клиентов двух типов в парикмахерской

Задание: проанализируйте полученный отчёт.

14.3. Модель парикмахерской с несколькими парикмахерами

14.3.1. Постановка задачи

Предположим, что клиенты появляются в парикмахерской через каждые 5 ± 5 минут. Время стрижки одного клиента составляет 19 ± 5 мин. Требуется определить характеристики очереди клиентов при условии, что их будут обслуживать четыре парикмахера.

14.3.2. Построение модели

Сегмент моделирования обслуживания клиентов:

```
;barber
kres STORAGE 4
GENERATE  5,5
QUEUE     barberq
ENTER     KRES
DEPART    barberq
ADVANCE   19,5
LEAVE     kres
TERMINATE 0
```

Сегмент моделирования таймера:

```
;timer
GENERATE 480
TERMINATE 1
START 1
```

После запуска симуляции получаем отчёт (рис. 14.3).

14.3.3. Задание

1) Проанализируйте полученный отчёт.
2) Измените модель: требуется учесть в ней возможные отказы клиентов от обслуживания — когда приходящий клиент видит в очереди более двух клиентов, он уходит из парикмахерской, то есть отказывается от обслуживания (используйте блок `TEST` и стандартный числовой атрибут `Qj` текущей длины очереди `j`).
3) Проанализируйте отчёт изменённой модели.

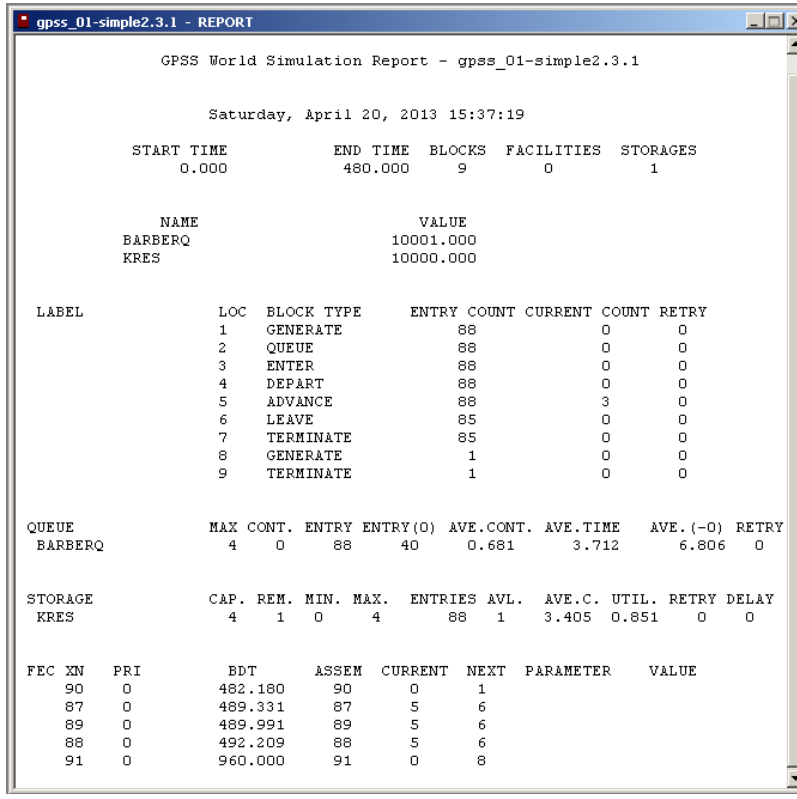


Рис. 14.3. Отчёт по модели обслуживания клиентов в парикмахерской четырьмя парикмахерами

Лабораторная работа 15. Модели обслуживания с приоритетами

15.1. Модель обслуживания механиков на складе

15.1.1. Постановка задачи

На фабрике на складе работает один кладовщик, который выдает запасные части механикам, обслуживающим станки. Время, необходимое для удовлетворения запроса, зависит от типа запасной части. Запросы бывают двух категорий. Для первой категории интервалы времени прихода механиков 420 ± 360 сек., время обслуживания — 300 ± 90 сек. Для второй категории интервалы времени прихода механиков 360 ± 240 сек., время обслуживания — 100 ± 30 сек.

Порядок обслуживания механиков кладовщиком такой: запросы первой категории обслуживаются только в том случае, когда в очереди нет ни одного запроса второй категории. Внутри одной категории дисциплина обслуживания — «первым пришел – первым обслужился». Необходимо создать модель работы кладовой, моделирование выполнять в течение восьмичасового рабочего дня.

15.1.2. Построение модели

Есть два различных типа заявок, поступающих на обслуживание к одному устройству. Различаются распределения интервалов приходов и времени обслуживания для этих типов заявок. Приоритеты запросов задаются путем использования для операнда `E` блока `GENERATE` запросов второй категории большего значения, чем для запросов первой категории.

Модель можно представить следующим образом:

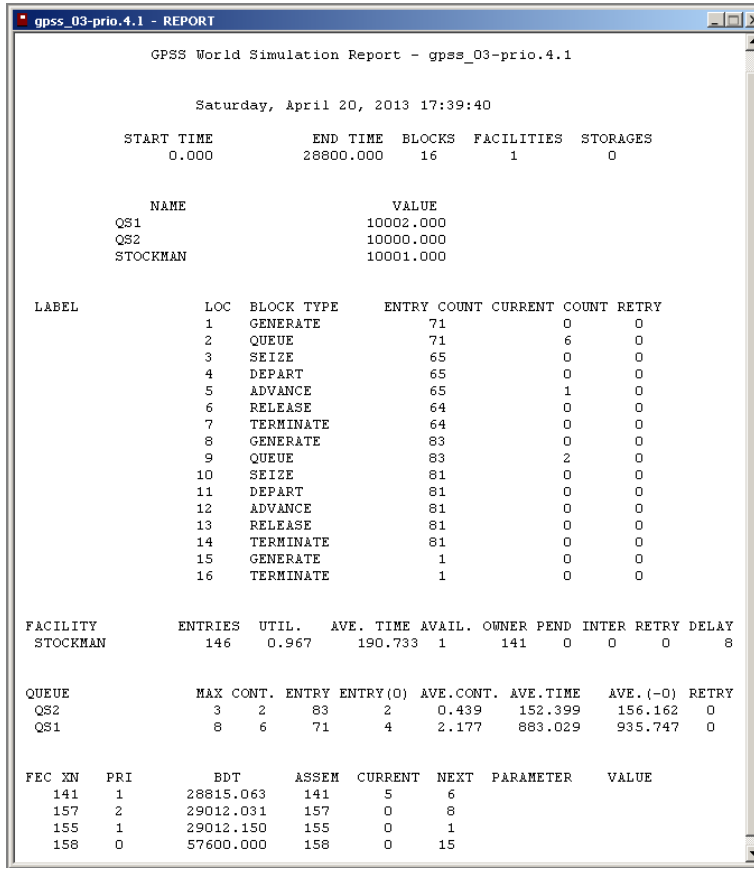
```
; type 1
GENERATE   420,360,,,1
QUEQUE     qs1
SEIZE      stockman
DEPART     qs1
ADVANCE    300,90
RELEASE    stockman
TERMINATE  0
; type 2
GENERATE   360,240,,,2
QUEQUE     qs2
SEIZE      stockman
DEPART     qs2
```

```
ADVANCE    100,30
RELEASE    stockman
TERMINATE  0
```

Сегмент моделирования таймера:

```
;timer
GENERATE 28800
TERMINATE 1
START     1
```

После запуска симуляции получаем отчёт (рис. 15.1).



```
gpss_03-prio.4.1 - REPORT

               GPSS World Simulation Report - gpss_03-prio.4.1


                    Saturday, April 20, 2013 17:39:40

          START TIME           END TIME  BLOCKS  FACILITIES  STORAGES
               0.000          28800.000    16        1          0


             NAME                       VALUE
          QS1                         10002.000
          QS2                         10000.000
          STOCKMAN                    10001.000


 LABEL              LOC  BLOCK TYPE     ENTRY COUNT CURRENT COUNT RETRY
                     1    GENERATE            71             0       0
                     2    QUEUE               71             6       0
                     3    SEIZE               65             0       0
                     4    DEPART              65             0       0
                     5    ADVANCE             65             1       0
                     6    RELEASE             64             0       0
                     7    TERMINATE           64             0       0
                     8    GENERATE            83             0       0
                     9    QUEUE               83             2       0
                    10    SEIZE               81             0       0
                    11    DEPART              81             0       0
                    12    ADVANCE             81             0       0
                    13    RELEASE             81             0       0
                    14    TERMINATE           81             0       0
                    15    GENERATE             1             0       0
                    16    TERMINATE            1             0       0


FACILITY         ENTRIES  UTIL.   AVE. TIME AVAIL. OWNER PEND INTER RETRY DELAY
 STOCKMAN           146    0.967     190.733  1      141    0    0     0      8


QUEUE              MAX CONT. ENTRY ENTRY(0) AVE.CONT. AVE.TIME   AVE.(-0) RETRY
 QS2                 3    2     83       2     0.439    152.399    156.162   0
 QS1                 8    6     71       4     2.177    883.029    935.747   0


FEC XN   PRI         BDT      ASSEM  CURRENT  NEXT  PARAMETER    VALUE
   141    1       28815.063    141      5      6
   157    2       29012.031    157      0      8
   155    1       29012.150    155      0      1
   158    0       57600.000    158      0     15
```

Рис. 15.1. Отчёт по модели обслуживания механиков с приоритетами

Задание: проанализируйте полученный отчёт.

15.2. Модель обслуживания в порту судов двух типов

15.2.1. Постановка задачи

Морские суда двух типов прибывают в порт, где происходит их разгрузка. В порту есть два буксира, обеспечивающих ввод и вывод кораблей из порта. К первому типу судов относятся корабли малого тоннажа, которые требуют использования одного буксира. Корабли второго типа имеют большие размеры, и для их ввода и вывода из порта требуется два буксира. Из-за различия размеров двух типов кораблей необходимы и причалы различного размера. Кроме того, корабли имеют различное время погрузки/разгрузки.

Требуется построить модель системы, в которой можно оценить время ожидания кораблями каждого типа входа в порт. Время ожидания входа в порт включает время ожидания освобождения причала и буксира. Корабль, ожидающий освобождения причала, не обслуживается буксиром до тех пор, пока не будет предоставлен нужный причал. Корабль второго типа не займёт буксир до тех пор, пока ему не будут доступны оба буксира.

Параметры модели:

- для корабля первого типа:
  - интервал прибытия: 130 ± 30 мин;
  - время входа в порт: 30 ± 7 мин;
  - количество доступных причалов: 6;
  - время погрузки/разгрузки: 12 ± 2 час;
  - время выхода из порта: 20 ± 5 мин;
- для корабля второго типа:
  - интервал прибытия: 390 ± 60 мин;
  - время входа в порт: 45 ± 12 мин;
  - количество доступных причалов: 3;
  - время погрузки/разгрузки: 18 ± 4 час;
  - время выхода из порта: 35 ± 10 мин.

15.2.2. Построение модели

```
prch1  STORAGE 6 ; 6 причалов для кораблей 1 типа
prch1  STORAGE 3 ; 3 причала для кораблей 2 типа
buks   STORAGE 2 ; 2 буксира

; ships of type 1
GENERATE 130,30  ; подход к порту
QUEUE    type1
ENTER    prch1   ; получение причала
ENTER    buks    ; получение буксира
```

```
DEPART   type1   ;
ADVANCE  30,7    ; буксирование до причала
LEAVE    buks    ; освобождение буксира
ADVANCE  720,120 ; погрузка / разгрузка
ENTER    buks    ; получение буксира
LEAVE    prch1   ; освобождение причала
ADVANCE  20,5    ; буксирование (отчаливание)
LEAVE    buks    ; освобождение буксира
TERMINATE

; ships of type 2
GENERATE 390,60  ; подход к порту
QUEUE    type2
ENTER    prch2   ; получение причала
ENTER    buks,2  ; получение 2-х буксиров
DEPART   type2   ;
ADVANCE  45,12   ; буксирование до причала
LEAVE    buks,2  ; освобождение буксиров
ADVANCE  1080,240; погрузка / разгрузка
ENTER    buks,2  ; получение 2-х буксиров
LEAVE    prch2   ; освобождение причала
ADVANCE  35,10   ; буксирование (отчаливание)
LEAVE    buks,2  ; освобождение буксира
TERMINATE 0
```

Сегмент моделирования таймера:

```
;timer
GENERATE  48000
TERMINATE 1
START     1
```

Среднее время ожидания кораблями каждого типа входа в порт получаем в конце моделирования из стандартной статистики об очередях: оно равно показателю `AVERAGE TIME` соответствующей очереди. Эти же значения дают стандартные числовые атрибуты `QT$TYPE1` и `QT$TYPE2`.

Задание: получите и проанализируйте отчёт.

Лабораторная работа 16. Задачи оптимизации. Модель двух стратегий обслуживания

16.1. Постановка задачи

На пограничном контрольно-пропускном пункте транспорта имеются 2 пункта пропуска. Интервалы времени между поступлением автомобилей имеют экспоненциальное распределение со средним значением μ. Время прохождения автомобилями пограничного контроля имеет равномерное распределение на интервале $[a, b]$.

Предлагается две стратегии обслуживания прибывающих автомобилей:

1) автомобили образуют две очереди и обслуживаются соответствующими пунктами пропуска;
2) автомобили образуют одну общую очередь и обслуживаются освободившимся пунктом пропуска.

Исходные данные: $\mu = 1,75$ мин, $a = 5$ мин, $b = 10$ мин.

16.2. Построение модели

Целью моделирования является определение:

- характеристик качества обслуживания автомобилей, в частности, средних длин очередей; среднего времени обслуживания автомобиля; среднего времени пребывания автомобиля на пункте пропуска;
- наилучшей стратегии обслуживания автомобилей на пункте пограничного контроля;
- оптимального количества пропускных пунктов.

В качестве критериев, используемых для сравнения стратегий обслуживания автомобилей, выберем:

- коэффициенты загрузки системы;
- максимальные и средние длины очередей;
- средние значения времени ожидания обслуживания.

Для первой стратегии обслуживания, когда прибывающие автомобили образуют две очереди и обслуживаются соответствующими пропускными пунктами, имеем следующую модель:

```
GENERATE (Exponential(1,0,1.75)) ; прибытие автомобилей
TEST LE Q$Other1,Q$Other2,Obsl_2 ; длина оч. 1<= длине оч. 2
TEST E Q$Other1,Q$Other2,Obsl_1  ; длина оч. 1=  длине оч. 2

TRANSFER 0.5,Obsl_1,Obsl_2 ; длины очередей равны,
                           ; выбираем произв. пункт пропуска
```

```
; моделирование работы пункта 1
Obsl_1 QUEUE Other1 ; присоединение к очереди 1
SEIZE punkt1        ; занятие пункта 1
DEPART Other1       ; выход из очереди 1
ADVANCE 3,1         ; обслуживание на пункте 1
RELEASE punkt1      ; освобождение пункта 1
TERMINATE           ; автомобиль покидает систему

; моделирование работы пункта 2
Obsl_2 QUEUE Other2 ; присоединение к очереди 2
SEIZE punkt2        ; занятие пункта 2
DEPART Other2       ; выход из очереди 2
ADVANCE 3,1         ; обслуживание на пункте 2
RELEASE punkt2      ; освобождение пункта 2
TERMINATE           ; автомобиль покидает систему

; задание условия остановки процедуры моделирования
GENERATE 10080 ; генерация фиктивного транзакта,
               ; указывающего на окончание рабочей недели
               ; (7 дней x 24 часа x 60 мин = 10080 мин)

TERMINATE 1    ; остановить моделирование
START 1        ; запуск процедуры моделирования
```

16.3. Задание

- составить модель для второй стратегии обслуживания, когда прибывающие автомобили образуют одну очередь и обслуживаются освободившимся пропускным пунктом;
- свести полученные статистики моделирования в таблицу 16.1.
- по результатам моделирования сделать вывод о наилучшей стратегии обслуживания автомобилей;
- изменив модели, определить оптимальное число пропускных пунктов (от 1 до 4) для каждой стратегии при условии, что:
  - коэффициент загрузки пропускных пунктов принадлежит интервалу $[0,5; 0,9]$;
  - среднее число автомобилей, одновременно находящихся на контрольно-пропускном пункте, не должно превышать 3;
  - среднее время ожидания обслуживания не должно превышать 4 мин.

Таблица 16.1

Сравнение стратегий

Показатель	стратегия 1			стратегия 2
	пункт 1	пункт 2	в целом	
Поступило автомобилей				
Обслужено автомобилей				
Коэффициент загрузки				
Максимальная длина очереди				
Средняя длина очереди				
Среднее время ожидания				

Лабораторная работа 17. Задания для самостоятельной работы

17.1. Моделирование работы вычислительного центра

На вычислительном центре в обработку принимаются три класса заданий A, B и C. Исходя из наличия оперативной памяти ЭВМ задания классов A и B могут решаться одновременно, а задания класса C монополизируют ЭВМ. Задания класса A поступают через 20 ± 5 мин, класса B — через 20 ± 10 мин, класса C — через 28 ± 5 мин и требуют для выполнения: класс A — 20 ± 5 мин, класс B — 21 ± 3 мин, класс C — 28 ± 5 мин. Задачи класса C загружаются в ЭВМ, если она полностью свободна. Задачи классов A и B могут дозагружаться к решающей задаче.

Смоделировать работу ЭВМ за 80 ч. Определить её загрузку.

17.2. Модель работы аэропорта

Самолёты прибывают для посадки в район аэропорта каждые 10 ± 5 мин. Если взлетно-посадочная полоса свободна, прибывший самолёт получает разрешение на посадку. Если полоса занята, самолет выполняет полет по кругу и возвращается в аэропорт каждые 5 мин. Если после пятого круга самолет не получает разрешения на посадку, он отправляется на запасной аэродром.

В аэропорту через каждые 10 ± 2 мин к взлетно-посадочной полосе выруливают готовые к взлёту самолёты и получают разрешение на взлёт, если полоса свободна. Для взлета и посадки самолёты занимают полосу ровно на 2 мин. Если при свободной полосе одновременно один самолёт прибывает для посадки, а другой — для взлёта, то полоса предоставляется взлетающей машине.

Требуется:

- выполнить моделирование работы аэропорта в течение суток;
- подсчитать количество самолётов, которые взлетели, сели и были направлены на запасной аэродром;
- определить коэффициент загрузки взлетно-посадочной полосы.

17.3. Моделирование работы морского порта

Морские суда прибывают в порт каждые $[a \pm \delta]$ часов. В порту имеется N причалов. Каждый корабль по длине занимает M причалов

и находится в порту $[b \pm \varepsilon]$ часов.

Требуется построить GPSS-модель для анализа работы морского порта, определить оптимальное количество причалов для эффективной работы порта.

Исходные данные:

1) $a = 20$ ч, $\delta = 5$ ч, $b = 10$ ч, $\varepsilon = 3$ ч, $N = 10$, $M = 3$;
2) $a = 30$ ч, $\delta = 10$ ч, $b = 8$ ч, $\varepsilon = 4$ ч, $N = 6$, $M = 2$.

Литература

1. Алтаев А. А. Имитационное моделирование на языке GPSS. — Улан-Удэ : Изд-во ВСГТУ, 2001. — 122 с.
2. Галкин А. М., Кучерявый Е. А., Молчанов Д. А. Пакет моделирования NS-2: учеб. пособие. — СПб : СПбГУТ, 2007. — URL: `http://seti.sut.ru/admin61/editor_files/file_upload/metod_ns2.pdf`.
3. Заборовский В. С. Моделирование и анализ сетей связи с коммутацией пакетов. Network Simulator (Сетевой симулятор ns2). — СПб : Изд-во СПбГТУ, 2001. — 108 с.
4. Заборовский В. С., Мулюха В. А., Подгурский Ю. Е. Моделирование и анализ компьютерных сетей: телематический подход. — СПб : Изд-во СПбГПУ, 2010. — 93 с.
5. Зайцев Д. А., Шмелева Т. Р. Моделирование телекоммуникационных систем в CPN Tools. — Одесса : Одесская национальная академия связи им. А.С. Попова, 2008.
6. Лобач В. И., Кирлица В. П., Малюгин В. И., Сталевская С. Н. Имитационное и статистическое моделирование. — Мн. : БГУ, 2004. — 189 с.
7. Котов В. Е. Сети Петри. — Москва : Наука, 1984. — С. 160.
8. Лоу А. М., Кельтон В. Д. Имитационное моделирование. — 3-е изд. — 2004. — С. 848. — ISBN: 5-94723-981-7, 966-552-118-7, 0070592926.
9. Марков А. А. Моделирование информационно-вычислительных процессов. — Москва : МГТУ им. Н. Э. Баумана, 1999. — С. 360. — ISBN: 5-7038-1334-4.
10. Морозов В. К., Рогачев Г. Н. Моделирование информационных и динамических систем. — Москва : Академия, 2011. — С. 384. — ISBN: 978-5-7695-4221-3.
11. Питерсон Д. Теория сетей Петри и моделирование систем. — Москва : Мир, 1984. — С. 264.
12. Советов Б. Я., Яковлев С. А. Моделирование систем. Практикум: учеб. пособие для вузов. — 2-е, перераб. и доп. изд. — М. : Высш. шк., 2003. — 295 с.
13. Томашевский В. Н., Жданова Е. Г. Имитационное моделирование в среде GPSS. — Бестселлер, 2003. — С. 416. — ISBN: 5-98158-004-6.
14. Шевченко Д. Н., Кравченя И. Н. Имитационное моделирование на GPSS : учеб.-метод. пособие для студентов технических специальностей. — Гомель : БелГУТ, 2007. — 97 с.
15. Шелухин О. И., Тенякшев А. М., Осин А. В. Моделирование информационных систем. — 2005. — С. 368. — ISBN: 5-93108-072-4.
16. Шеннон Р. Имитационное моделирование систем - искусство и наука. — 1978. — С. 418.

17. Barakat C. Exercises on "ns-2". — 2003.
18. Cpn tool. — 2014. — URL: http://cpntools.org.
19. Issariyakul T., Hossain E. Introduction to Network Simulator NS2. — Springer, 2009. — URL: https://sites.google.com/site/naeemkhademi/Home/ns-2-document-repository/IntroductiontoNetworkSimulatorNS20387717595.pdf?attredirects=0&d=1.
20. Pieda P., Ethridge J., Baines M., Shallwani F. A Network Simulator Differentiated Services Implementation. — Open IP, Nortel Networks, 2000. — URL: https://sites.google.com/site/naeemkhademi/Home/ns-2-document-repository/DSnortel.pdf?attredirects=0&d=1.

Учебно-методический комплекс дисциплины

«Моделирование информационных процессов»

Рекомендуется для направления подготовки
010300.62 «Фундаментальная информатика
и информационные технологии»

Квалификация (степень) выпускника: бакалавр

Программа дисциплины

Цели и задачи дисциплины

Цели

Целью дисциплины является изучение:
- фундаментальных основ теории моделирования информационных систем и протекающих в них процессов;
- методик разработки компьютерных моделей;
- методов и средств осуществления имитационного моделирования и обработки результатов вычислительных экспериментов;
- формирование представления о работе с современными инструментальными системами моделирования.

Задачи

В результате изучения курса решаются следующие задачи:
- освоение теоретических основ математического и компьютерного моделирования информационно-вычислительных систем;
- приобретение навыков использования основных классов моделей и методов моделирования, принципов построения моделей информационных процессов, методов формализации, алгоритмизации и реализации моделей с помощью современных компьютерных средств;
- приобретение навыков проведения вычислительных экспериментов с использованием техники имитационного моделирования, планирование проведения экспериментов и обработка их результатов;
- построение моделей систем различного класса с использованием инструментальных средств типа xcos, GPSS и др.

Место дисциплины в структуре основной образовательной программы

Цикл, к которому относится дисциплина: вариативная часть математического и естественнонаучного цикла Б2.

Требования к входным знаниям, умениям и компетенциям студента: требуется пройти обучение по дисциплинам: «Теория вероятностей и математическая статистика», «Дифференциальные и разностные уравнения», «Компьютерные сети».

Студент должен:

знать:

– и применять в исследовательской и прикладной деятельности современный математический аппарат, фундаментальные концепции и системные методологии, международные и профессиональные стандарты в области информационных технологий, способность использовать современные инструментальные и вычислительные средства (в соответствии с профилем подготовки) (ПК-4);
– концепции и абстракции, использовать на практике базовые математические дисциплины, включая: «Математический анализ I», «Математический анализ II», «Кратные интегралы и ряды», «Алгебра и геометрия», «Дискретная математика», «Теория функций комплексной переменной», «Математическая логика и теория алгоритмов», «Теория автоматов и формальных языков», «Дифференциальные и разностные уравнения», «Теория вероятностей и математическая статистика», «Вычислительные методы» и др. (ПК-15);

уметь:

– использовать основные законы естественнонаучных дисциплин в профессиональной деятельности, применять методы математического анализа и моделирования, теоретического и экспериментального исследования (ОК-10);

владеть:

– базовыми математическими знаниями и информационными технологиями, эффективно применять их для решения научно-технических задач и прикладных задач, связанных с развитием и использованием инф. технологий (ПК 8).

Дисциплины, для которых данная дисциплина является предшествующей: «Программные системы и математическое моделирование», «Компьютерный практикум по телекоммуникациям», «Компьютерный практикум по оптике наноструктур», курсовая работа, выпускная квалификационная работа.

Требования к результатам освоения дисциплины

Процесс изучения дисциплины направлен на формирование следующих компетенций: ОК: 10, ПК: 2, 3, 4, 8, 9, 22, 26, 27, 28, 29.

В результате изучения дисциплины студент должен:

знать:

– теоретические и методические основы, понимать функциональные возможности, области применения компонентно-базированного программирования (ПК-22);

– теоретические основы и общие принципы использования следующих профессиональных областей: разработка информационных систем, моделирование и анализ программного обеспечения, разработка и принципы сетевых технологий (П-26);

уметь:

– использовать основные законы естественнонаучных дисциплин в профессиональной деятельности, применять методы математического анализа и моделирования, теоретического и экспериментального исследования (ОК-10);
– профессионально решать задачи производственной и технологической деятельности с учетом современных достижений науки и техники, включая: создание информационных ресурсов глобальных сетей, образовательного контента (ПК-2);
– разрабатывать и реализовывать процессы жизненного цикла информационных систем, а также методы и механизмы оценки и анализа функционирования средств и систем информационных технологий (ПК-3);
– применять в исследовательской и прикладной деятельности фундаментальные концепции и системные методологии, международные и профессиональные стандарты в области сетевых технологий (ПК-4);
– осуществлять на практике современные методологии управления жизненным циклом и качеством систем, программных средств и сервисов информационных технологий (ПК-9);
– квалифицированно применять в профессиональной деятельности современные электронные библиотеки и коллекции, сетевые технологии, библиотеки и пакеты программ, современные профессиональные стандарты информационных технологий (П-27);
– решать задачи производственной и технологической деятельности на высоком профессиональном уровне, включая: разработку математических, информационных и имитационных моделей по тематике выполняемых опытно-конструкторских работ и проектов (П-28);
– разрабатывать, оценивать и реализовывать процессы жизненного цикла информационных систем, программного обеспечения, сервисов систем информационных технологий, а также реализовывать методы и механизмы оценки и анализа функционирования средств и систем информационных технологий (П-29);

владеть:

– базовыми сетевыми технологиями, эффективно применять их для решения научно-технических задач и прикладных задач, связанных с развитием и использованием информационных и сетевых технологий (ПК-8).

Объем дисциплины и виды учебной работы

Общая трудоемкость дисциплины составляет 2 зачётные единицы.

№	Вид учебной работы	Всего часов	Семестры
			6
1	**Аудиторные занятия (всего)**	**54**	**54**
	В том числе:		
1.1	Лекции		
1.2	Прочие занятия	54	54
	В том числе:		
1.2.1	*Практические занятия (ПЗ)*	2	2
1.2.2	*Семинары (С)*	2	2
1.2.3	*Лабораторные работы (ЛР)*	50	50
1.2.4	**Из них в интерактивной форме (ИФ):**	50	50
2	**Самостоятельная работа студентов (ак. часов)**	**18**	**18**
	В том числе:		
2.1	Курсовой проект (работа)	-	-
2.2	Расчетно-графические работы	-	-
2.3	Реферат	-	-
2.4	Подготовка и прохождение промежуточной аттестации	18	18
2.5	*Другие виды самостоятельной работы:*		
2.5.1	Самостоятельная проработка дополнительных материалов по дисциплине	-	-
2.5.2	Выполнение домашних заданий	-	-
3	**Общая трудоемкость (ак.часов)**	**72**	**72**
4	**Общая трудоемкость (зачетных единиц)**	**2**	**2**

Содержание дисциплины

Содержание разделов дисциплины

1. Основные понятия теории моделирования информационных систем.
   1) Моделирование как метод научного познания, роль и место вычислительного эксперимента в исследовательской деятельности. Классификация моделей: понятия математической и компьютерной модели, имитационное моделирование. Моделирование непрерывных, дискретных и гибридных систем. Принципы системного подхода в моделировании. Стадии разработки моделей. Понятия компонентного и объектно ориентированного моделирования.
   2) Обзор современных программных инструментальных средств моделирования систем.
2. Имитационное моделирование в NS-2.
   1) Тактические планы проведения имитационного моделирования: задание начальных условий и параметров и оценка их влияния на достижение установившегося результата. Вопросы обеспечения точности и достоверности результатов имитационного моделирования. Постановки задач обработки результатов имитационного моделирования.
   2) Основы работы в NS-2. Общее описание, список некоторых команд NS-2. Файл трассировки. NAM. Основы работы в Xgraph. Основы работы в Gnuplot. AWK.
   3) Выполнение лабораторных работ.
3. Компонентное моделирование. Scilab, подсистема xcos.
   1) Понятие динамической и событийно-управляемой системы, гибридные системы. Принципы компонентного компьютерного моделирования. Иерархические системы. Блоки и связи между ними. Ориентированные и неориентированные блоки и связи. Неявные взаимодействия компонентов.
   2) Реализация компонентного моделирования в подсистеме xcos математического пакета Scilab. Основные библиотечные блоки. Последовательность построения и отладки xcos-моделей. Средства анализа результатов моделирования.
   3) Выполнение лабораторных работ.
4. Сетевые модели и синхронизация событий. Сети Петри.
   1) Сети Петри. Основные понятия и определения. Применение сетей Петри к моделированию программного обеспечения. Задачи синхронизации. Задачи анализа сетей Петри. Методы анализа сетей Петри. Матричное представление сети Петри.
   2) Основы работы в CPN Tools.
   3) Выполнение лабораторных работ.

5. Моделирование систем массового обслуживания и функциональных процессов.
   1) Дискретно-событийный подход к моделированию. Проблемно-ориентированный язык и программная среда GPSS/PC. Общие принципы моделирования информационных и вычислительных процессов в GPSS/PC. Базовые сведения о системе: объекты, переменные и выражения, функции. Модель системы: модельное время и статистика. Внутренняя организация: списки и общая внутренняя последовательность событий. Элементы языка моделирования GPSS/PC. Среда моделирования GPSS/PC: операторы, команды управления, интерактивное взаимодействие.
   2) Выполнение лабораторных работ.

Разделы дисциплины и междисциплинарные связи с обеспечиваемыми (последующими) дисциплинами

№ п/п	Наименование обеспечиваемых (последующих) дисциплин	№ № разделов данной дисциплины, необходимых для изучения обеспечиваемых (последующих) дисциплин				
		1	2	3	4	5
1.	Программные системы и математическое моделирование	+	+	+	+	+
2.	Компьютерный практикум по телекоммуникациям	+	+	+	+	+
3.	Компьютерный практикум по оптике наноструктур	+	+	+	+	+
4.	Курсовая работа	+	+	+	+	+
5.	Выпускная квалификационная работа	+	+	+	+	+

Разделы дисциплин и виды занятий

№ п/п	Наименование раздела дисциплины	Лекц.	Практ. зан. и лаб. раб.			СРС	Всего час.
			ПЗ/С	ЛР	Из них в ИФ		
1.	Основные понятия теории моделирования информационных систем		2			3	**5**
2.	Имитационное моделирование в NS-2			16	16	3	**19**
3.	Компонентное моделирование. Scilab, xcos			12	12	3	**15**
4.	Сетевые модели и синхронизация событий. Сети Петри.			12	12	3	**15**
5.	Моделирование СМО и функциональных процессов			10	10	3	**13**
6.	Итоговый контроль знаний		2			3	**5**
	Итого:	**0**	**4**	**50**	**50**	**18**	**72**

Описание интерактивных занятий

№ п/п	№ р/д	Тема ИФ	Вид занятия	Труд. (час.)
1	2	Л.1. Простые модели компьютерной сети	Лаб. раб., вып. малой гр. (2-3 чел.)	4
		Л.2 Исследование протокола TCP и алгоритма управления очередью RED.	Лаб. раб., вып. малой гр. (2-3 чел.)	3

		Л.3. Моделирование стохастических процессов. Реализация модели на NS-2	Лаб. раб., вып. малой гр. (2-3 чел.)	3
		Л.4. Задание для самостоятельного выполнения:	Лаб. раб., вып. малой гр. (2-3 чел.). Творческое задание	6
2	3	Л.5. Модель эпидемии (SIR)	Лаб. раб., вып. малой гр. (2-3 чел.)	3
		Л.6. Модель «хищник–жертва»	Лаб. раб., вып. малой гр. (2-3 чел.)	3
		Л.7. Модель M |M |1|∞. Реализация в xcos	Лаб. раб., вып. малой гр. (2-3 чел.)	3
		Л.8. Модель TCP/AQM	Лаб. раб., вып. малой гр. (2-3 чел.)	3
3	4	Л. 9. Модель «накорми студентов»	Лаб. раб., вып. малой гр. (2-3 чел.)	1
		Л. 10. Задача об обедающих мудрецах	Лаб. раб., вып. малой гр. (2-3 чел.)	2
		Л. 11. Модель СМО M |M |1	Лаб. раб., вып. малой гр. (2-3 чел.)	3
		Л. 12. Пример моделирования простого протокола передачи данных	Лаб. раб., вып. малой гр. (2-3 чел.)	3
		Л.13. Задание для самостоятельного выполнения	Лаб. раб., вып. малой гр. (2-3 чел.). Творческое задание	3
4	5	Л.14. Модели парикмахерской	Лаб. раб., вып. малой гр. (2-3 чел.)	1
		Л.15. Модели обслуживания с приоритетами	Лаб. раб., вып. малой гр. (2-3 чел.)	2
		Л.16. Задачи оптимизации. Модель двух стратегий обслуживания	Лаб. раб., вып. малой гр. (2-3 чел.)	3
		Л.17. Задания для самостоятельной работы	Лаб. раб., вып. малой гр. (2-3 чел.). Творческое задание	4

Практические занятия (семинары)

№ п/п	№ р/д	Тема практического занятия (семинара)	Труд. (час.)
1	1	Базовые понятия теории МИП	2
2	6	Итоговый контроль знаний	2

Лабораторный практикум

№ п/п	№ р/д	Наименование лабораторных работ	Труд. (час.)
1	2	Л.1. Простые модели компьютерной сети	4
		Л.2 Исследование протокола TCP и алгоритма управления очередью RED.	3
		Л.3. Моделирование стохастических процессов. Реализация модели на NS-2	3
		Л.4. Задание для самостоятельного выполнения	6
2	3	Л.5. Модель эпидемии (SIR)	3
		Л.6. Модель «Хищник–жертва»	3
		Л.7. Модель M |M |1|∞. Реализация в xcos	3
		Л.8. Модель TCP/AQM	3
3	4	Л. 9. Модель «Накорми студентов»	1
		Л. 10. Задача об обедающих мудрецах	2
		Л. 11. Модель СМО M |M |1	3
		Л. 12. Пример моделирования простого протокола передачи данных	3
		Л.13. Задание для самостоятельного выполнения	3
4	5	Л.14. Модели парикмахерской	1
		Л.15. Модели обслуживания с приоритетами	2
		Л.16. Задачи оптимизации. Модель двух стратегий обслуживания	3
		Л.17. Задания для самостоятельной работы	4

Примерная тематика курсовых проектов (работ)

Курсовые работы не предусмотрены.

Учебно-методическое и информационное обеспечение дисциплины

а) **Основная литература**

1. Алексеев Е.Р., Чеснокова О.В., Рудченко Е.А. Scilab: Решение инженерных и математических задач, 2008. — `http://books.altlinux.ru/altlibrary/scilab`.
2. Грекул В.И., Денищенко Г.Н., Коровкина Н.Л. Проектирование информационных систем. — Интернет-университет инфор-

мационных технологий — ИНТУИТ.ру, 2008. — 308 с. — http://www.intuit.ru/department/se/devis/.
3. Губарь Ю.В. Введение в математическое моделирование. — ИНТУИТ.ру, 2007. — http://www.intuit.ru/department/calculate/intromathmodel/.

б) **Дополнительная литература**

1. Советов Б.Я., Яковлев С.А. Моделирование систем : учебник для ВУЗов. — М.: Высшая школа, 1999. — 319 с.
2. Бусленко Н.П. Моделирование сложных систем. — М.: Наука, 1978. — 399 с.
3. Питерсон Дж. Теория сетей Петри и моделирование систем. — М.: Мир, 1984. — 264 с.
4. Бычков С.П., Храмов А.А. Разработка моделей в системе моделирования GPSS : учебное пособие. – М.: МИФИ, 1997. — 32 с.
5. Кравченко П.П., Хусаинов Н.Ш. Имитационное моделирование вычислительных систем средствами GPSS/PC. — Таганрог: ТРТУ, 2000. — 116 с.
6. Бенькович Е.С., Колесов Ю.Б., Сениченков Ю.Б. Практическое моделирование динамических систем – СПб.: БХВ-Петербург, 2002. — 464 с.
7. Кулябов Д.С., Королькова А.В. Архитектура и принципы построения современных сетей и систем телекоммуникаций. — М.: РУДН, 2008.
8. Боев В. Концептуальное проектирование систем в AnyLogic и GPSS World. — ИНТУИТ.ру, 2013. URL: http://www.intuit.ru/studies/courses/4818/1066/info9.
9. Грекул В. Теория информационных систем. — ИНТУИТ.ру, 2009. URL: http://www.intuit.ru/studies/courses/507/363/info.
10. Кирсанов А. Теория информационных технологий и систем. — ИНТУИТ.ру, 2009. URL: http://www.intuit.ru/studies/courses/1158/315/info

в) **Программное обеспечение:** ОС Linux, ОС Windows, scilab, xcos, ns-2, GPSS/PC, cpntools.

г) **Базы данных, информационно-справочные и поисковые системы:**

1. Official ns-2 website — http://www.isi.edu/nsnam/ns/
2. Официальный сайт SciLab — http://www.scilab.org/
3. Официальный сайт Modelica — https://www.modelica.org/
4. URL Официальный сайт OpenModelica — http://www.openmodelica.org/

Материально-техническое обеспечение дисциплины

Москва, ул. Орджоникидзе, д.3, корп. 5. Дисплейные классы ДК3, ДК4, ДК6, ДК7, Intel Core i3-550 3.2 GHz – 60 шт.

Методические рекомендации по организации изучения дисциплины

Учебным планом на изучение дисциплины отводится один семестр. Промежуточный контроль знаний предусматривает: выполнение и защиту лабораторных работ. В качестве итогового контроля знаний предусмотрен зачет в форме тестирования в конце семестра.

Примерный перечень тем итоговых семестровых испытаний:

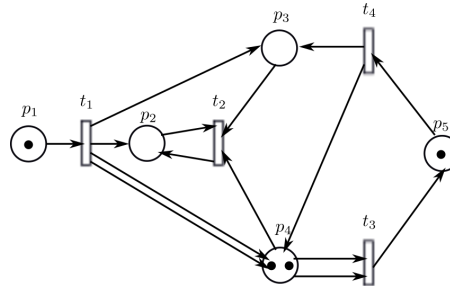
1. Классификация моделей: понятия математической и компьютерной модели, имитационное моделирование.
2. Моделирование непрерывных, дискретных и гибридных систем.
3. Принципы системного подхода в моделировании.
4. Принципы компонентного компьютерного моделирования.
5. Виды имитационного моделирования. Области применения имитационного моделирования с примерами моделируемых процессов.
6. Методы визуализации результатов эксперимента в NS-2.
7. Применение сетей Петри к моделированию программного обеспечения.
8. Сети Петри. Основные понятия и определения.
9. Сети Петри. Задачи синхронизации.
10. Сети Петри. Задачи анализа сетей Петри.
11. Основные блоки GPSS.
12. Подготовка данных к моделированию в GPSS.
13. Структура файла отчета GPSS.

Фонды оценочных средств

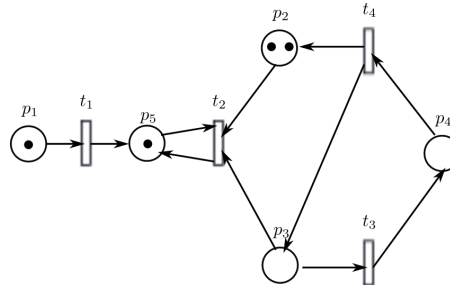
Формулировки тестовых заданий по темам

1) Основные понятия теории моделирования информационных систем:
   1. Непрерывная модель (дать определение).
   2. Детерминированная модель (дать определение).
   3. Динамическая модель (дать определение).
   4. Имитационная модель (дать определение)
2) Имитационное моделирование в NS-2:
   1. В NS-2 агент TCP как создаётся?
   2. В NS-2 агенты как соединяются?
   3. В NS-2 строка `$ns attach-agent $n0 $udp0` (что делает?).
   4. В NS-2 строка `$ns duplex-link $n0 $n1` (что делает?).
   5. В NS-2 строка `$ns run` (что делает?).
   6. В NS-2 строка `set f [open out.tr w]` (что делает?).
   7. В NS-2 строка `set n0 [$ns node]` (что делает?).
   8. В NS-2 строка `set ns [new Simulator]` (что делает?).
   9. Как сделать комментарий в Tcl?
3) Компонентное моделирование. Scilab, подсистема xcos:
   1. Как сделать комментарий в SciLab?
   2. С какого знака в SciLab начинаются стандартные скалярные переменные, например, мнимая единица `i`?
   3. Что произойдет при вызове в SciLab `printf(''e=%.7f'', % e)`?
   4. В SciLab формуле синтаксиса `[t]=sin(x)` параметр x — обязателен?
   5. Задайте в SciLab пользовательскую переменную j для хранения результата деления 1/3.
   6. Как вывести на экран (в SciLab) значение числа 1/9 с тремя знаками после запятой?
   7. Каков результат следующих действий в SciLab:
      `a = 10; b = 6; a+b; c = ans - 1`;
4) Сетевые модели и синхронизация событий. Сети Петри:
   1. Сети Петри (назначение).
   2. В сети Петри что называется примитивным событием?
   3. Сети Петри (выбор правильного утверждения — по определениям).
   4. Какого цвета становится индикатор в CPNTools, если действие было успешно завершено?
   5. Как в CPNTools визуально определить, разрешен переход или нет?

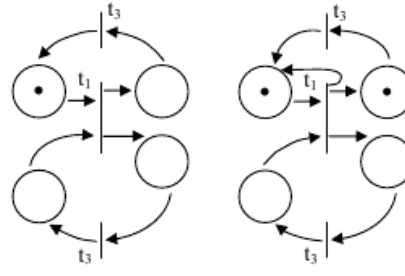
6. Что означает ярко-красная подсветка элементов в CPNTools?
7. Из следующего множества атрибутов CPNTools выберите только те, которые относятся к позициям.
8. Для приведённой на рисунке сети Петри укажите разрешённые переходы

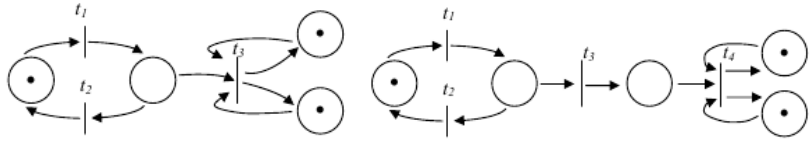


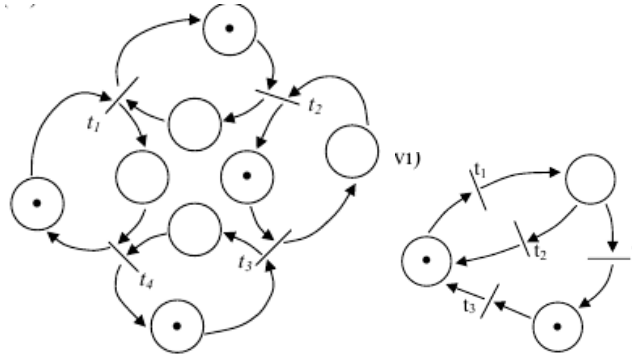
9. Какая маркировка соответствует приведённой на рисунке сети Петри?



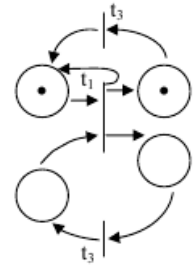
10. Какие из приведенных ниже сетей Петри имеют достижимыми каждое состояние?



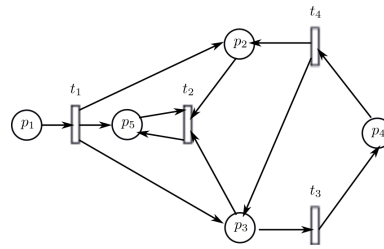




11. Какие из приведенных сетей Петри (см. 10) не являются безопасными?
12. Какие из приведенных сетей Петри (см. 10) не являются консервативными?
13. Какие из приведенных сетей Петри (см. 10) не являются ограниченными?
14. Какие из приведенных сетей Петри (см. 10) являются безопасными?
15. Охарактеризуйте приведенную на рисунке сеть Петри.



16. Для приведённой на рисунке сети Петри `C = (P, T, I, O)`, `P = {p1, p2, p3, p4, p5}`, `T={t1 , t2, t3, t4}` укажите элементы расширенной функции `I(p2)`.



17. Для приведённой на рисунке сети Петри (см. 16) `C = (P, T, I, O)`, `P = {p1, p2, p3, p4, p5}`, `T = {t1, t2, t3, t4}` укажите входные позиции перехода `t2`.
18. Для приведённой на рисунке сети Петри (см. 16) `C = (P, T, I, O)`, `P = {p1, p2, p3, p4, p5}`, `T = {t1, t2, t3, t4}` укажите выходные позиции перехода `t2`.

5) Моделирование систем массового обслуживания и функциональных процессов:
   1. В GPSS блок ADVANCE (что делает?).
   2. В GPSS блок ASSIGN (что делает?).
   3. В GPSS блок DEPART (что делает?).
   4. В GPSS блок GENERATE (что делает?).
   5. В GPSS блок LEAVE (что делает?).
   6. В GPSS блок PREEMPT (что делает?).
   7. В GPSS блок PRIORITY (что делает?).
   8. В GPSS блок QUEUE (что делает?).
   9. В GPSS блок RELEASE (что делает?).
   10. В GPSS блок RETURN (что делает?).
   11. В GPSS блок SEIZE (что делает?).
   12. В GPSS блок SPLIT (что делает?).
   13. В GPSS блок TERMINATE (что делает?).
   14. В GPSS блок TRANSFER (что делает?).
   15. В GPSS в выходном файле статистики AVE.CONT: (что выводится?).
   16. В GPSS в выходном файле статистики AVE.TIME: (что выводится?).
   17. В GPSS в выходном файле статистики ENTRIES: (что выводится?).
   18. В GPSS в выходном файле статистики FACILITIES: (что выводится?).
   19. В GPSS в выходном файле статистики QUEUE: (что выводит-

ся?).
20. В GPSS в выходном файле статистики START TIME: (что выводится?).
21. В GPSS в выходном файле статистики STORAGES: (что выводится?).
22. В GPSS функция GAMMA() используется для чего?
23. В GPSS функция POISSON() используется для чего?

Список вопросов и заданий для контроля знаний

1. Укажите и поясните три составляющие технологии.
2. Укажите цель, предмет и средства информационной технологии.
3. Перечислите и поясните составляющие методологии технологического процесса.
4. Перечислите и поясните составляющие системного подхода к понятию «технология».
5. Перечислите и поясните типы технологий управления процессом производства.
6. Поясните структуру базовой информационной технологии.
7. Дайте определение информационного процесса.
8. Дайте определение сигнала, укажите типы сигналов.
9. Какие типы сигналов применяются для управления, а какие для накопления информации?
10. Дайте определения моделирования и модели. Сравните разные определения.
11. Причины, по которым необходимо использовать имитационное моделирование.
12. Виды имитационного моделирования.
13. Области применения имитационного моделирования с примерами моделируемых процессов.
14. Классификация моделей.
15. Опишите методы визуализации результатов эксперимента в NS-2. Графический интерпретатор nam.
16. Построить имитационную модель математического маятника в xcos, используя стандартные блоки.
17. Построить имитационную модель математического маятника в xcos, используя язык modelica.
18. Построить график изменения амплитуды колебаний математического маятника в xcos, используя стандартные блоки.
19. Построить фазовый портрет математического маятника в xcos, используя стандартные блоки.
20. В GPSS смоделировать систему типа $M|M|N|K$.
21. В GPSS смоделировать систему типа $M|M|N|K$ с ограниченным временем ожидания.

Календарный план

Виды и содержание учебных занятий				
Неделя	**Практические занятия / семинар**	**Число часов**	**Лабораторные занятия**	**Число часов**
1	Базовые понятия теории МИП.	**2**	Шаблон сценария для NS-2	**1**
Основы работы в NS-2				
2			Л.1. Простые модели компьютерной сети	**3**
3			Л.2 Исследование протокола TCP и алгоритма управления очередью RED	**3**
4			Л.3. Моделирование стохастических процессов. Реализация модели на NS-2	**3**
5-6			Л.4. Задание для самостоятельного выполнения:	**6**
Компонентное моделирование. Scilab, подсистема xcos				
7			Л.5. Модель эпидемии (SIR)	**3**
8			Л.6. Модель «хищник–жертва»	**3**
9			Л.7. Модель M |M |1|∞. Реализация в xcos	**3**
10			Л.8. Модель TCP/AQM	**3**
Сети Петри. Моделирование в CPN Tools				
11			Л. 9. Модель «накорми студентов»	**1**
			Л. 10. Задача об обедающих мудрецах	**2**
12			Л. 11. Модель СМО M |M |1	**3**
13			Л. 12. Пример моделирования простого протокола передачи данных	**3**
14			Л.13. Задание для самостоятельного выполнения	**3**

Имитационное моделирование. Моделирование в GPSS				
15			Л.14. Модели парикмахерской Л.15. Модели обслуживания с приоритетами	**3**
16			Л.16. Задачи оптимизации. Модель двух стратегий обслуживания	**3**
17			Л.17. Задания для самостоятельной работы	**3**
18	Итоговый контроль знаний	**2**		**1**
Итого:		**4**		**50**

Балльно-рейтинговая система

Рейтинговая система оценки знаний студентов

Раздел	Тема	Формы контроля уровня освоения ООП		Баллы темы	Баллы раздела
		ЛР	Итог. контроль (тест)		
1	**2**	**3**	**4**	**5**	**6**
Основные понятия теории моделирования информационных систем	Моделирование как метод научного познания, роль и место вычислительного эксперимента в исследовательской деятельности. Классификация моделей: понятия математической и компьютерной модели, имитационное моделирование. Моделирование непрерывных, дискретных и гибридных систем. Принципы системного подхода в моделировании. Стадии разработки моделей. Понятия компонентного и объектно ориентированного моделирования	-	2	**2**	**2**
Основы работы в NS-2.	*Л.1. Простые модели компьютерной сети:*		3	**6**	**23**
	Л.1.1. Шаблон сценария для NS-2	1			
	Л.1.2. Простой пример описания топологии сети, состоящей из двух узлов и одного соединения	1			
	Л.1.3. Пример с усложнённой топологией сети	2			

1	2	3	4	5	6
	Л.1.4. Пример с кольцевой топологией сети	2			
	Л.2 Исследование протокола TCP и алгоритма управления очередью RED. Пример с дисциплиной RED	2		**2**	
	Л.3. Моделирование стохастических процессов. Реализация модели на NS-2	2		**2**	
	Л.4. Задание для самостоятельного выполнения:				
	Л.4.1.Для приведённой схемы разработать имитационную модель в пакете NS-2.	2		**10**	
	Л.4.2.Посчитать (с выводом значения на экран) число пакетов, поступивших в очередь с первого узла.	2			
	Л.4.3.Посчитать (с выводом значения на экран) среднюю длину очереди и среднее время пребывания в очереди пакетов, поступивших с первого узла.	2			
	Л.4.4.Построить график изменения длины очереди и средней длины очереди на узле 2N + 1.	2			
	Л.4.5.Построить график изменения пропускной способности звена 2N +1 и 2N + 2.	2			
Компонентное моделирование. Scilab, подсистема xcos.	Упражнение. Построить с помощью xcos фигуры Лиссажу с различными значениями параметров.	1	2	**1**	**22**

1	2	3	4	5	6
	Л.5. Модель эпидемии (SIR):			**7**	
	Л.5.1. Реализация модели в xcos	2			
	Л.5.2. Реализация модели с помощью блока Modelica в xcos	2			
	Л.5.3. Задание для самостоятельного выполнения	3			
	Л.6. Модель «хищник–жертва»:			**4**	
	Л.6.1. Реализация модели в xcos	2			
	Л.6.2. Реализация модели с помощью блока Modelica в xcos	2			
	Л.7. Модель M |M |1|∞. Реализация модели в xcos	2		**2**	
	Л.8. Модель TCP/AQM:			**6**	
	Л.8.1. Реализация модели в xcos с различными параметрами.	2			
	Л.8.2. Задание для самостоятельного выполнения: реализовать модель с использованием языка Modelica, построить график динамики изменения размера TCP окна W (t) и размера очереди Q(t) и фазовый портрет (W, Q).	4			
Сети Петри. Моделирование в CPN Tools	Л. 9. Модель «накорми студентов»	1	6	**1**	**26**
	Л. 10. Задача об обедающих мудрецах	1		**1**	
	Л. 11. Модель системы массового обслуживания M |M |1	4		**4**	

1	2	3	4	5	6
	Л. 12. Пример моделирования простого протокола передачи данных	6		**6**	
	Л. 13. Задание для самостоятельного выполнения	8		**8**	
Имитационное моделирование. Моделирование в GPSS.	*Л.14. Модели парикмахерской:*		7	**4**	**27**
	Л.14.1.Модель обслуживания клиентов одним парикмахером	1			
	Л.14.2. Модель обслуживания двух типов клиентов в парикмахерской	1			
	Л.14.3. Модель парикмахерской с несколькими парикмахерами	2			
	Л.15. Модели обслуживания с приоритетами			**4**	
	Л.15.1. Модель обслуживания механиков на складе	2			
	Л.15.2. Модель обслуживания в порту судов двух типов	2			
	Л.16. Задачи оптимизации. Модель двух стратегий обслуживания	3		**3**	
	Л.17. Задания для самостоятельной работы			**9**	
	Л.17.1. Моделирование работы вычислительного центра	3			
	Л.17.2. Модель работы аэропорта	3			
	Л.17.3. Моделирование работы морского порта	3			
	Итого:	**80**	**20**	**80**	**100**

Таблица соответствия баллов и оценок

Баллы БРС	Традиционные оценки РФ	Оценки ECTS
95 - 100	5	A
86 - 94		B
69 - 85	4	C
61 - 68	3	D
51 - 60		E
31 - 50	2	FX
0 - 30		F
	Зачет	Passed

Правила применения БРС

1. Раздел (тема) учебной дисциплины считаются освоенными, если студент набрал более 50 % от возможного числа баллов по этому разделу (теме).
2. Студент не может быть аттестован по дисциплине, если он не освоил все темы и разделы дисциплины, указанные в сводной оценочной таблице дисциплины.
3. По решению преподавателя и с согласия студентов, не освоивших отдельные разделы (темы) изучаемой дисциплины, в течение учебного семестра могут быть повторно проведены мероприятия текущего контроля успеваемости или выданы дополнительные учебные задания по этим темам или разделам. При этом студентам за данную работу засчитывается минимально возможный положительный балл (51 % от максимального балла).
4. При выполнении студентом дополнительных учебных заданий или повторного прохождения мероприятий текущего контроля полученные им баллы засчитываются за конкретные темы. Итоговая сумма баллов не может превышать максимального количества баллов, установленного по данным темам (в соответствии с приказом Ректора № 564 от 20.06.2013). По решению преподавателя предыдущие баллы, полученные студентом по учебным заданиям, могут быть аннулированы.
5. График проведения мероприятий текущего контроля успеваемости формируется в соответствии с календарным планом курса. Студенты обязаны сдавать все задания в сроки, установленные преподавателем.
6. Время, которое отводится студенту на выполнение мероприятий текущего контроля успеваемости, устанавливается преподавате-

лем. По завершении отведённого времени студент должен сдать работу преподавателю, вне зависимости от того, завершена она или нет.

7. Использование источников (в том числе конспектов лекций и лабораторных работ) во время выполнения контрольных мероприятий возможно только с разрешения преподавателя.
8. Отсрочка в прохождении мероприятий текущего контроля успеваемости считается уважительной только в случае болезни студента, что подтверждается наличием у него медицинской справки, заверенной круглой печатью в поликлинике № 25, предоставляемой преподавателю не позднее двух недель после выздоровления. В этом случае выполнение контрольных мероприятий осуществляется после выздоровления студента в срок, назначенный преподавателем. В противном случае, отсутствие студента на контрольном мероприятии признается не уважительным.
9. Студент допускается к итоговому контролю знаний с любым количеством баллов, набранных в семестре, но при условии, что у студента имеется теоретическая возможность получить за весь курс не менее 31 балла.
10. Итоговый контроль знаний оценивается из 20 баллов независимо от числа баллов за семестр.
11. Если в итоге за семестр студент получил менее 31 балла, то ему выставляется оценка F и студент должен повторить эту дисциплину в установленном порядке. Если же в итоге студент получил 31-50 баллов, т. е. FX, то студенту разрешается добор необходимого (до 51) количества баллов путем повторного одноразового выполнения предусмотренных контрольных мероприятий, при этом по усмотрению преподавателя аннулируются соответствующие предыдущие результаты. Ликвидация задолженностей проводится в период с 07.02 по 28.02 (с 07.09 по 28.09) по согласованию с деканатом.

Сведения об авторах

Королькова Анна Владиславовна — кандидат физико-математических наук, доцент-исследователь кафедры прикладной информатики и теории вероятностей РУДН.

Кулябов Дмитрий Сергеевич — кандидат физико-математических наук, доцент, доцент-исследователь кафедры прикладной информатики и теории вероятностей РУДН.

Учебное издание

Анна Владиславовна Королькова
Дмитрий Сергеевич Кулябов

Моделирование информационных процессов

Тематический план изданий учебной и научной литературы
2014 г., № 16

Редактор *И.Л. Панкратова*
Технический редактор *Н. А. Ясько*
Компьютерная вёрстка *А. В. Королькова*, *Д. С. Кулябов*
Дизайн обложки *М. В. Рогова*

Подписано в печать 20.03.2014 г. Формат 60×84/16. Печать офсетная.
Усл. печ. л. 11,16. Тираж 100 экз. Заказ № 249.

Российский университет дружбы народов
115419, ГСП-1, г. Москва, ул. Орджоникидзе, д. 3

Типография РУДН
115419, ГСП-1, г. Москва, ул. Орджоникидзе, д. 3, тел. 952-04-41