**Моделирование программного обеспечения**

Напомним некоторые определения, относящиеся к теории программирования. *Моделью программирования* называют совокупность приемов программирования и структур данных, отвечающих архитектуре используемого компьютера, и предназначенную для выполнения определенного класса алгоритмов. Описание алгоритма является иерархическим, его сложность зависит от степени детализации. На самом верхнем уровне иерархии находится задача в целом, нижний уровень включает примитивные операции - машинные команды. Алгоритм можно представить в виде диаграммы - ***информационного графа,*** который описывает последовательность выполнения операций и взаимную зависимость между различными операциями или блоками

операций. Узлами информационного графа являются операции, а однонаправленными дугами - каналы обмена данными.

Традиционной считается ***последовательная модель программирования.*** В этом случае в любой момент времени э выполняется только одна операция и только над одним элементом данных. Эта модель характеризуется возможностью применения стандартных языков программирования, хорошей переносимостью программ и невысокой производительностью.

В более поздних вычислительных системах получила развитие идея распараллеливания вычислений, которая. воплотилась в архитектуре практически всех современных компьютеров. В связи с этим появились ***модели параллельного программирования***. Их особенностями являются более высокая производительность программ, применение специальных приемов программирования и, как следствие, более высокая трудоемкость программирования и проблемы с переносимостью программ.

Простейшая модель параллельного программировании называется ***моделью задача/канал****.* В этом случае программа состоит из нескольких задач, связанных между собой каналами телекоммуникаций и выполняемых одновременно несколькими процессорами. Каждая задача состоит из последовательного кода и локальной памяти. Канал - это очередь сообщений, передающих данные от одной задачи к другой. Каждая задача может записывать и считывать данные из локальной памяти, посылать и принимать сообщения. Операция обмена сообщениями может быть ***асинхронной****,* она завершается сразу после пересылки сообщения, либо синхронной, которая блокирует выполнение задачи до тех пор, пока не придет подтверждение получения сообщения адресатом.

Другой распространенной моделью параллельного! программирования является ***модель параллелизма данных****.* Она основана на применении одного и того же алгоритма несколькими процессорами к множеству элементов структуры данных (например, к фрагментам одного массива).

Обратимся теперь к описанию алгоритмов с помощью сетей Петри. Эти сети, оперируя терминами условия-события являются удобным инструментом моделирования как структуры информационных графов программ, так и динамики выполнения программных блоков. При этом наличие данных для выполнения некоторых операций моделируется маркировкой соответствующей позиции, а сама операция моделируетсясрабатыванием перехода.

Ниже приводятся некоторые сведения о моделях параллельного программирования и их интерпретации с помощью сетей Петри. Отметим, что данный материал не претендует на изложение теории параллельных вычислений - он лишь демонстрирует моделирующие возможности сетей Петри.

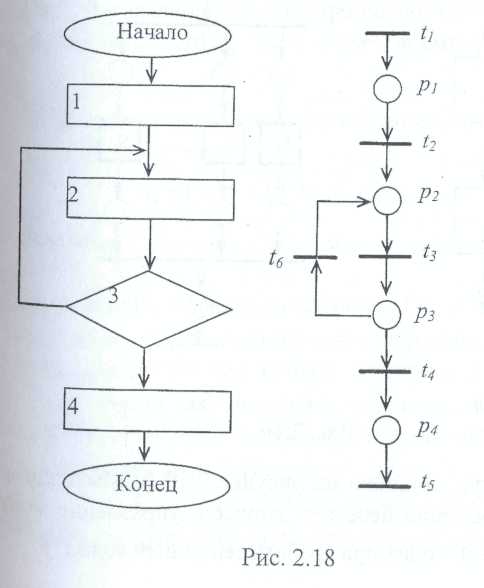
**1. Последовательная модель программирования**

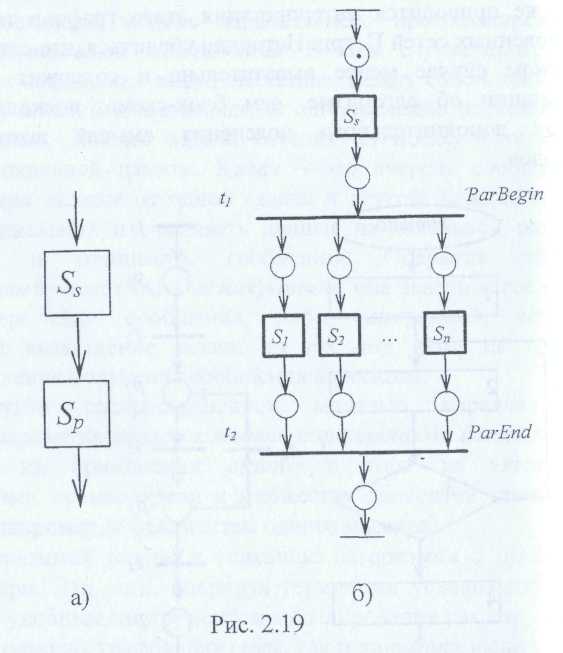
На рисунке 2.18 в качестве примера последовательной модели программирования показан информационный граф (стандартная блок-схема) алгоритма циклических вычислений. Здесь же приводится интерпретация этого графа в терминах обыкновенных сетей Петри. Нетрудно убедиться, что сеть Петри данном случае менее выразительна и содержит меньше информации об алгоритме, чем блок-схема, поскольку она требует дополнительного пояснения смысла позиций и переходов.

**2. Модель параллелизма данных**

Рассмотрим теперь схему программы, соответствующую модели параллелизма данных. Предположим, что в программе имеются блоки двух типов: *Ss* и *Sp* (рис. 2.19 а). Алгоритм,

блока *SK* может выполняться только в последовательном, режиме одним процессором, а блок *Sp* допускает разбивку на n независимых модулей *S1...,Sn,* каждый из которых можетвыполняться на отдельном процессоре.

****



Сеть Петри показана на рисунке 2.19 б. Мы видим, чтоблок *Ss* одновременно передает данные и управление модулям *S,,...,Sn* (это происходит при срабатывании перехода *tt).*

Модули выполняются независимо, однако дальнейшие вычисления могут начаться только после завершения выполнения всех модулей (т.е. должны сложиться условия для срабатывания перехода *t2).* Таким образом, с помощью сети Петри можно описать нетривиальные особенности параллельного выполнения модулей. С помощью стандартных блок-схем это было бы сделать затруднительно.

На примере данной схемы можно вывести известную ***формулу* *Амдала****,* которая характеризует эффективность параллельных алгоритмов.

Пусть блок *Ss* требует для своего выполнения времени t, а блок *Sp* при его вьполнении на одном процессоре -времени *tp.* Тогда общее время решения задачи на одном процессоре составит *t,=ts+ tp.* Если же блок *Sp* выполняется параллельно на *п* одинаковых процессорах, то время его выполнения в идеальном случае (при равномерной загрузке всех процессоров) составит *tp / n,* а общее время решения задачи будет *t2=ts+tp/n.*

Коэффициент ускорения вычислений составит

*ts+tp I*

*k = t,lt2= ~~s~~~~p~~ =* -, (2.19)

*ts+tp/n s + p/n*

*ts  tp*

где *s =* ——; *p = —*—— \_ относительные доли

*ts + tp ts + tp*

последовательной и параллельной частей *(s + p = 1).*

Выражение 2.19 носит название ***формулы Амдала.***

Мы видим, что чем больше величина *р,* тем больший эффект дает эффект распараллеливания вычислений. При малых *p* ускорение вычислений за счет увеличения числа процессоров будет незначительным. Например, при *р = 0,5* и *п = 10* получим *k = 1,82*, т.е. десятикратное увеличение числа процессоров уменьшает время вычислений менее чем в 2 раза.

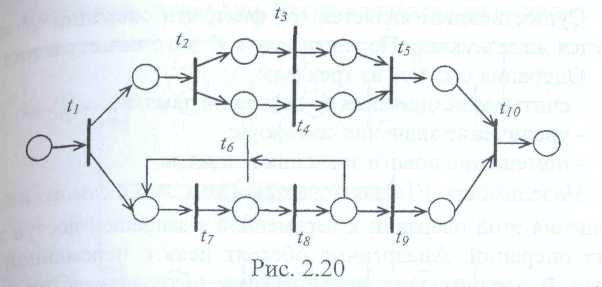
**3. Моделирование некоторых структур параллельного программирования. Семафоры**

Одним из средств создания параллельных программ на базе стандартных языков программирования является введение специальных средств, выделяющих параллельно выполняемые участки программ. В частности, Э. Дейкстрой были предложены скобки *ParBegin* и *ParEnd,* причем первая из них разрешает начало выполнения параллельных операторов, а вторая разрешает переход к дальнейшим вычислениям. На рисунке 2.19 переходу *t,* соответствует скобка *ParBegin,* а переходу *t2 - ParEnd.*

Рассмотрим теперь последовательно-параллельные структуры, которые представляют собой обобщение последовательных структур за счет включения в набор управляющих примитивов специальных операторов или указателей, которые выделяют параллельные ветви программы, исполняемые независимо друг от друга.

Ветви имеют общее начало - точку расхождения ветвей и конец - точку схождения ветвей. Совокупность ветвей с общим началом и концом образует сегмент параллельной структуры. При выполнении программы процесс вычислений продвигается до начала первого сегмента, после чего расщепляется на столько копий, сколько ветвей содержит сегмент. Каждый из параллельных процессов вычисления ветвей протекает независимо от других и достигнув конца ветви, останавливается, ожидая, пока все остальные процессы в сегменте не достигнут конца сегмента. В конце сегмента все процессы сливаются в один. Параллельныеветви сегментов могут, в свою очередь, содержать сегменты.

В параллельно-последовательных языках программирования для выделения параллельных ветвей используются специальныепрограммные примитивы, такие как упомянутые скобки параллельности *ParBegin ... ParEnd,* или специальныеоператоры, отмечающие начало и конец параллельного сегмента. Примером могут служить операторы *fork(a,b,c…) и**join(a,b,c,..).* Здесь *a,b,c,* ... - метки, помечающие начало и конец параллельных ветвей.



Так, на рисунке 2.20 переход *t1* имеет смысл *fork(a,b),* переход *tl0-join(a,b).* Аналогично *t2-fork(c,d), t5- join(c,d).*

В рассмотренных структурах отсутствует взаимодействие между параллельными ветвями, например, использование одних и тех же переменных, одних и тех же устройств, переходы из одной ветви в другую и т.д. В реальных случаях такие в взаимодействия могут возникать, и поэтому нужны дополни­тельные программные средства, которые приостанавливали бы процесс исполнения некоторой ветви, пока не прибудут данные тот другой ветви или пока другие ветви не освободят общий ресурс. Отрезок ветви, на котором требуется «захват» общего «ресурса, называется ***критическим интервалом***ветви.

Э. Дейкстра предложил для этих целей механизм ***семафоров****,* включающий переменные специального типа(*semafore)* и две операции *Р* и *V,* аргументами которых могут быть только переменные типа *semafore.*

Область значений семафора - целые неотрицательные числа. Если область значений сужена до двух (0 и 1), то семафор называется бинарным.

II Операция *V* изменяет значения семафора *s* на *s +1.*Действия операции *Р* определяются следующим образом:  
если *s ≠0,* то *Р* замещает элемент *s* на s -*1*;

I если *s* = *0,* то *Р* не изменяет значения *s* и не завершается до тех пор, пока некоторая другая ветвь не изменит *s* с помощью операции *V.*

Существенным является тот факт, что операции Р и V считаются «неделимыми». По отношению к *V* это означает следующее.

Операция состоит из трех фаз:

- считывание значения семафора из памяти;

- увеличение значения семафора;

- помещение нового значения в память.

Неделимость V(s) заключается в том, что с самого начала выполнения этой операции к переменной *s* запрещен доступ всех других операций. Аналогично обстоят дела с переменной *P*. Обычно *Р п*редшествует критическому интервалу ветви, а *V* завершает его.

Ниже рассмотрен пример параллельной программы соскобками *ParBegin ... ParEnd,* выделяющими сегмент с двумя параллельными ветвями *рrос\_1* и *proc\_2,* разделенными запятой и семафором *sem.* Каждая из параллельных ветвей представляет собой бесконечный последовательный цикл, содержащий критический интервал (*crit\_section)* и остальную часть цикла *(remainder\_of\_cycle).*

В этом примере решается задача взаимного исключения критических интервалов двух циклических процессов параллельных ветвей.

*Var*

*1,2 : label;*

*sem : semafore;*

*begin*

*sem:=1;*

*parbegin*

*proc\_l:*

*begin 1 : P(sem);*

*crit\_sect\_l;*

*V(sem);*

*remainder\_of\_cycle\_l; goto 1 end,*

*proc\_2:*

*begin 2 : P(sem);*

*crit\_sect\_2;*

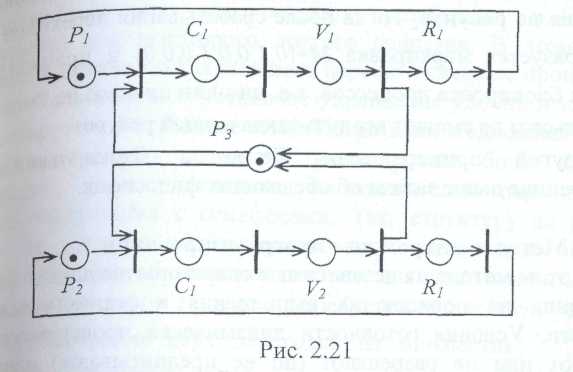
*V(sem);*

*remainder\_of\_cycle\_2; goto 2 end,*

*parend*

*end.*

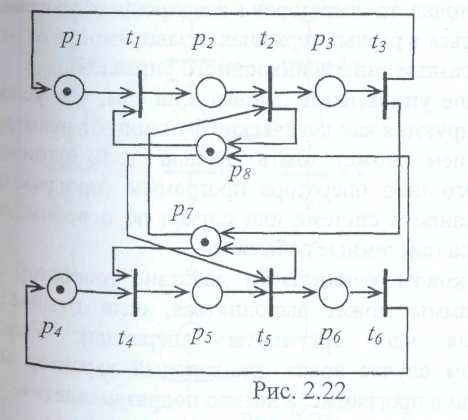
Соответствующая сеть Петри приведена на рисунке 2.21.



Здесь переходы *Р;* соответствуют выполнению операции *P,* Сi - критическому интервалу ветви, *Vi* - операциям *V, R; -*остальной части цикла (*i* *=1,2).* Позиция *Р3* соответствует

семафору *sem.*

При выполнении параллельно-последовательных программ возможны ситуации, называемые ***взаимной блокировкой***(рис. 2.22).



Пусть М0= [1,0,0,1,0,0,1,1] – начальная маркировка, показанная на рисунке. Тогда после срабатывания переходов t1 и *t4* образуется маркировка *М=[0,1,0,0,1,0,0,0]* и произойдет взаимная блокировка процессов, т.е. ни один процесс не сможет выполняться и не сможет вернуть захваченный ресурс.

Другой пример сети Петри с блокировкой, рассмотренная выше задача об обедающих философах.

**4. Метод асинхронного программирования**

В этом методе на независимые операторы накладываются ограничения на порядок их выполнения в форме условий готовности. Условия готовности динамически проверяются и разрешают или не разрешают (но не предписывают) начать выполнение операторов, с которыми данные условия связаны. Условия готовности берут на себя всю организацию управления.

Из сказанного следует, что асинхронный принцип организации вычислений является как бы дополнительным к параллельно-последовательному и оба принципа эквивалентны в том смысле, что последовательно-параллельная программа содержит средства синхронизации параллельных ветвей.

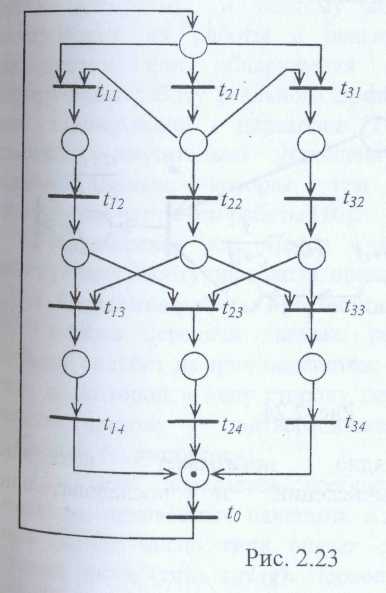
Условия готовности операторов в асинхронных программах могут формироваться в разных терминах, в зависимости от этой можно выделить разные типы асинхронного управления.

**Событийное управление** основано на том, что условия готовности формируются как логические функции от некоторых событий. Событием может быть начало или окончание выполнения какого-либо оператора программы (программные, события), прерывание в системе или сигнал об освобождении некоторого ресурса (системные события).

**При потоковом управлении** действие (оператор или операция) программы может вьшолняться, если готовы всенеобходимые для него аргументы (операнды). Условие готовности в этом случае носит стандартный характер и невыписывается явно в программе, а неявно подразумевается.

**В обратном потоковом управлении** действие может выполняться, если его результат необходим в качестве аргумента для некоторого другого действия. В этом случае. второе действие как бы вызывает первые в качестве процедуры.

Событийное и потоковое управление удобно и наглядно моделируется в терминах сетей Петри и их модификаций. При этом сети Петри позволяют моделировать более сложные ситуации, чем механизм последовательно-параллельного программирования с семафорами. Так, структуру на рисунке 2.23 не удается запрограммировать с помощью обычных семафоров- Это объясняется тем, что для выполнения каждого из трех операторов *t13, t23, t33* требуется захват двух ресурсов, т.е. срабатывание двух переходов из множества *{l12,t22,t32}.* Семафор управляет распределением только одного ресурса, поскольку неделимая операция *Р* применяется только к одному семафору.



Для того, чтобы аппарат сетей Петри можно было применять не только для описания семантики событийного управления, но и непосредственно использовать в языке параллельного программирования, разработаны аналитические методы представления сетей, которые позволяют задавать структуру асинхронного управления программы в видеуправляющих выражений [9].

Рассмотрим пример потокового асинхронного уцравления. При таком управлении исполнение оператора может начаться, если для него готовы входные данные (аргументы).

Сети Петри при этом моделируют с помощью позиций наличие аргументов, а с помощью переходов - выполнен! операций.

*а + Ь*

Например, вычисления выражения моделируется

*с-а*

схемой, показанной на рисунке 2.24. Символ 1/*х* означает вычисление обратной величины.

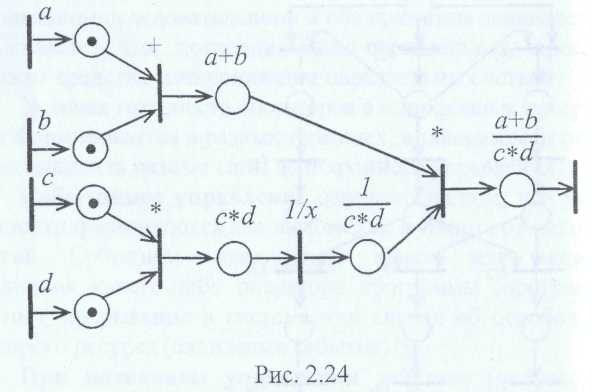


Рисунок наглядно показывает возможность распараллеливания вычислений и последовательностьвыполняемых операций.

**3 Моделирование протоколов передачи данных**

' Протоколами называют наборы правил (алгоритмы) для связи между абонентами вычислительной или телекоммуникационной системы. Протоколы управляют форматом, временными интервалами, последовательностью работы и контролем ошибок при передаче сообщений. В соответствии с принятой моделью связи открытых систем OSI (Open System Interconnection)

существует семь уровней взаимодействия между абонентами системы *-* от физического (нижний уровень) до прикладного (верхний уровень). На каждом из уровней должен действовать свой протокол. На практике обычно используется всего три уровня протоколов:

- низший аппаратный (протоколы Ethernet, Token Ring, FDDI);

- средний уровень (протоколы NetBIOS, IPX/SPX, TCP/IP);.

- высший уровень (протоколы SMB, NCP).

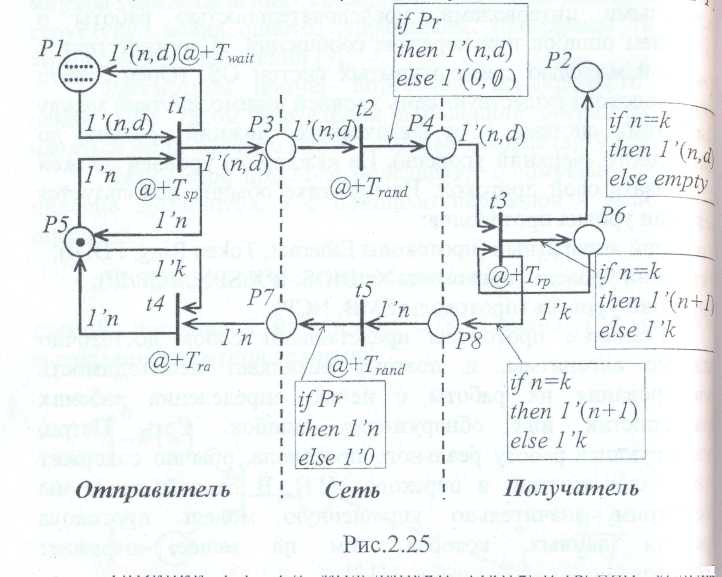
Реальные протоколы представляют собой достаточно сложные алгоритмы, и поэтому возникает необходимость моделирования их работы с целью определения рабочих характеристик или обнаружения ошибок. Сеть Петри, моделирующая работу реального протокола, обычно содержит сотни узлов-позиций и переходов [11]. В данной книге мы рассмотрим значительно упрощенную модель протокола передачи данных, которая, тем не менее, отражает существенные черты его работы [10].

Раскрашенная сеть Петри с временным механизмом, моделирующая работу протокола, приведена на рисунке 2.25.

**1. Описание работы протокола**

Система передачи данных, реализуемая с помощью протокола, состоит из трех подсистем: отправитель, получатель и сеть, по которой в одну сторону передается сообщение, а в обратную сторону - подтверждение о приеме сообщения получателем («квитанция»).

По сети передается сообщение, разбитое на *N* фрагментов, называемых пакетами. Каждый пакет состоит из номера (целое число типа *integer* с временной меткой) и текстовой части (типа *string).* Первоначально все *N* пакетов находятся в позиции *Р1.* Целью работы системы является передача всех пакетов получателю в позицию *Р2* с сохранением последовательности их номеров. Кроме того, отправитель должен получить «квитанции» о передаче всех пакетов, и эти«квитанции» с пометкой о времени передачи помещаются в *P1*.



Позиции *РЗ*, *Р4* моделируют, соответственно, условия начала и конца передачи пакетов, а позиции *Р8, Р* *7* моделируют условия начала и конца передачи «квитанций».

Позиции *Р5, Рб* являются счетчиками, т.е. хранят номера отправленных и принятых пакетов.

Переход *t,* моделирует начало передачи очередного пакета, переходы *t2* и *t5 -* прохождение, соответственно, пакета и «квитанции» через сеть, переход *t3 -* прием и анализ очередного пакета, переход *t4* - прием и анализ очередной «квитанции».

**2< Формальное описание CPN, моделирующей протокол.**

Ниже приведено формальное описание сети на рисунке 2.25 в сокращенном виде.

Множество цветов:

*type color INT - integer timed*;

*type color DATA* = *string* ;

*type color PACKET* = *produrt INT x DATA timed* ;

Var n, k: соlor *INT \ Var d : color DATA*;

• Множества позиций *Р = {Р1,...,Р8)* и переходов *T* = *(tl,...,t5)* показаны непосредственно на рисунке.

• Цветовая функция имеет вид

*color PACKET if {P1,P2,P3,P4}*

*C(p)=*

*color INT otherwise*

*•*Выражения на дугах *Е(а)* показаны непосредственно на рисунке. Обратим внимание на дуги *t2 to P4* и *t5 to P7.* Выражения на этих дугах зависят от датчика случайного кода *Рr,* который с определенной вероятностью вырабатывает значение *true* или *false.* Прохождение сигнала по этим дугам рассмотрено ниже.

• Функции на переходах *G(t),* определяющие временные задержки при срабатывании переходов, (т.е. передаче сообщений), показаны на рисунке.

•Функция инициализации имеет вид (рис. 2.25) *m0{Pl)= 1’(1," раск1’’);1’(2," pack2");...;1’(N," packN") -* иными словами, в начальный момент времени *(*Θ*= 0 ,τ = 0)* в позиции *P1* находится *N* подлежащих передаче пакетов, структура которых определена типом *color PACKET;*

*то{Р5)=то(Рб) = 1’1 -* начальный номер передаваемого пакета.

*то (Р2) = т0 (Р3)= т0 (Р4) = т0 (Р7) = т0 {Р8) = Ø* .

**3. Временной механизм работы CPN**

Важную роль в работе рассматриваемой модели игрант временной механизм.

Все переходы осуществляют временную задержку: *t*1- на время *Tsp, t3 -* на время *Trp, t4* - на время *Тт,* задержка срабатывания переходов *t2* и *t5 Trond* является случайной величиной, которую генерирует датчик случайных чисел, На дуге *tl to PI* введена задержка на время *Twait.*

Перемещаемые в сети фишки имеют временные метки и при их прохождении через переходы значения «прикрепленного» времени возрастают, что моделируетзадержку передачи как сообщений, так и «квитанций».

Помимо временной задержки, в рассматриваемой СРN моделируется потеря сообщения как при передаче пакета (на дуге *t2 to P4*), так и при передаче «квитанции» (на дуге *t5 to Р7).* Моделирование этих событий состоит в том, что с некоторой вероятностью, генерируемой специальным датчиком Рг, вместо очередного пакета вида 1’*(n,d)* по дуге *t2 to P4* передается «пустой» пакет *1'(0,Ø),* а по дуге *t5 to P7* вместо сообщения 1’*п -* «пустое» сообщение 1’0*.*

**4. Описание работы CPN**

Проследим передачу одного пакета в системе. Пусть часы глобального времени в момент начала передачи показываю τ = τ\*. В позиции *Р5* находится фишка 1’*п,* в позиции *Р1*фишки 1’*(n,d),...,1’(N,d),* а в позиции *Р2 –* фишки 1’*(l,d),...,1’(n-l,d),* где *п* - номер очередного передаваемого пакета, *n = l,...,N.*

При срабатывании *tl* из *Р1* извлекается фишка 1’*(n,d)* с задержкой *Tsp* передается в *РЗ.* Кроме того, фишка 1’n, задержкой *Tsp* по дуге *tl to Р5* возвращается в *Р5,* а фишка 1’*(n,d)* по дуге *tl to PI* с временной меткой τ\*+Tsp+T wait направляется в *P'l.*

Следующим срабатывает переход *t2* и со случайной задержкой *Trand* передает в *Р4* либо фишку *l'(n,d),* либо пустую1’(0,Ø) - взависимости от сигнала датчика *Рr. Переход t3* анализирует пришедшую в *Р4* информацию. Если пришел очередной ожидаемый пакет, т.е. если хранимая в *Р6* фишка *1’ к* имеет тот же номер, что и фишка *l'(n,d) {n = к* ), то после задержки *Trp* происходят следующие действия:

. принятый пакет *l'(n,d)* передается получателю в *Р2* с временной меткой τ *+ Tsp + Tmnd* + *Trp* ;

-содержимое счетчика в *Р6* увеличивается на 1;

- в *Р8* отправляется фишка *1'(п +* /) с временной меткой τ\* + Тsp + Trond +*- Тrp*

Если же в *Р4* пришло «пустое» сообщение (т.е. *п≠к),* то в *Р2* ничего не передается, содержимое *Рб* не изменяется и в позицию *Р8* отправляется фишка с прежним номером 1’*п* и указанной выше временной меткой.

Передача «квитанции» по обратному каналу происходит аналогичным образом. Фишка *1'(п + ])* или 1’*п* с накопленной временной задержкой проходит через переход *t5,* при этом временная метка увеличивается на случайную величину *Trmd.*

Затем на дуге *t5 to P7* происходит проверка потери «квиитанции». При срабатывании датчика *Рr* в позицию *Р7 в*место фишек 1’*{п + 1)* или 1’*п* приходит фишка 1’0*.* Переход t4 анализирует пришедшую фишку. При совпадении номера пришедшей фишки с номером отправленной переход *t4* срабывает и отправляет фишку 1’*{п + 1)* в *Р5,* в противном случае срабатывания не происходит и в *Р5* остается прежний номер пакета *п .* В первом случае может сработать переход *tl* и начнется передача пакета 1’*(п +1, d)* в описанном выше порядке. Во втором случае система оказывается заблокированной до тех пор, пока предыдущий пакет по дуге *tl to PI* не вернется в *Р1.*

После этого начнется повторная передача пакета1'(*n,d*) с начальной временной отметкой τ\* + *Т + Twnil.*

Описанная модель протокола передачи данных позволяет исследовать характеристики его работы. Варьируя временазадержки *Tsp Trp Tra Twait,* функции распределения величины *Tmnd* и датчика *Рr* , можно путем имитационного моделирования (многократного «прогона» пакетов через CPN) подобрать параметры протокола, обеспечивающие его эффективную работу.

**2.3.4. Об исследовании сетей Петри с помощью ЭВМ**

**| 1. Специализированные программные средства.**

К настоящему времени разработано большое количество программных систем автоматизированного анализа сетей Петри, реализующих различные аспекты их использования: анализ общих сетей Петри и имитация их функционирования,ориентация на различные расширения этих сетей, использование средств машинной графики для ввода и редактирования сетей и др. Методы анализа общих и частных свойств сетей Петри - безопасность, ограниченность, консервативность, живость переходов, достижимость, наличие тупиков и циклов - также успешно реализуются на ЭВМ. Некоторые из этих программных систем являются автономными системами анализа сетей Петри (как, например, самая развитая и полная интерактивная система OVIDE [25]), в других сеть Петри используется в качестве имитационной модели, позволяющейпровести верификацию и определить динамические характеристики реального моделируемого объекта (коммуникационных протоколов, систем с управлением потоком данных, логических схем, параллельных систем и др.).

Промышленные программы, предназначенные дляанализа реальных вычислительных систем с помощью сетей Петри, обычно имеют дело с десятками и сотнями тысячпозиций и переходов и требуют соответствующих вычислительных ресурсов. Так, для моделирования процессора большой ЭВМ CDC 6600 потребовалась сеть Петри, содержащая около 500тысяч позиций и переходов [8].Более скромная система, реализуемая на персональном компьютере, описана в статье [19J.

Как уже упоминалось в п. 2.2, для анализа систем с помощью CPN создан специальный язык моделирования CPN ML[10]. Также приобрели популярность программные средства, разработанные под руководством проф. К. Йенсена в Ааруском университете (Дания).

Отметим, в частности, программу CPN Tools (http;vww.daimi.au.dk/CPnets/CPN2000). Она распространяется по лицензиям, которые могут свободно получить университеты и академические организации. Программа работает в операционной среде Windows, обладает удобным графическим интерфейсом и может использоваться для моделирования различных систем. Краткая инструкция по работе с этой системой приведена в приложении А.

В рамках данного ознакомительного курса, естественно, нет необходимости обращаться к сложным и дорогостоящим системам. В то же время программирование работы небольших по объему сетей Петри и их расширений не представляет сложности для студентов - третьекурсников направлений 552800 и 654600. Поэтому' составление таких программ предусмотрено в лабораторном практикуме и в качестве упражнений для самостоятельной работы.

**2. Использование параллельных и кластерных вычислительных систем.** Как мы видели выше, исследование сетей Петри реальных размерностей представляет собой сложную задачу компьютерного моделирования. Поэтому понятен большой интерес к оценке возможностей, которые открываются при использовании для этих целей кластерных вычислительных систем и алгоритмов параллельных вычислений. Этот интерес оправдывается еще и тем, что кластерные вычислительные системы становятся доступными не только ведущим научным учреждениям, но широкому кругу исследователей, в том числе студентам.

Ниже рассмотрен один из возможных подходов к распараллеливанию процесса построения дерева маркировки обыкновенной сети Петри [26].

В отличие от задач, в которых имеется весь исходный материал для вычислений и трудоемкость которых зависит от размерности и трудоемких математических операций, в нашем случае основные временные затраты приходятся на порождение дерева маркировок, которое растет динамически. Предсказать скорость роста и размеры дерева, либо провести качественный анализ можно лишь в отдельных случаях.

Проблему можно попытаться решить, если распараллеливать вычисления по мере роста дерева, т.е.обрабатывать каждую новую ветку на отдельном потоке. Таким образом, получится, что каждый узел кластера будет обрабатывать свое собственное поддерево.

Для определения некоторых параметров, например поиска циклов, необходимо иметь все дерево маркировок. Поиск циклов необходимо производить после получения новой маркировки. Поэтому нужно либо передавать все дерево на каждый узел кластера (при этом растет нагрузка на сеть, т.к. нужно часто передавать большие объемы данных), либо хранить все дерево маркировок на одном узле. При этом повышаются требования к производительности этого узла, но, с другой стороны, массированные операции парного сравнения маркировок могут быть также выполнены в параллельном режиме. В этом же процессе может производиться порождение словаря свободного языка сети Петри.

Если ставить перед собой только задачу прямой достижимости (или другие задачи, не требующие операций над всем деревом маркировок одновременно), то тогда проблема хранения этого дерева вообще не возникает. В противном случае эту проблему можно разрешить, также распределив и эти вычисления. Например, поиск циклов выполнять не сразу по всему дереву, а только в том поддереве, которое доступно в данный момент, а в дальнейшем производить вторичный поиск в оставшихся ветках.

0Писанные проблемы являются, возможно, наиболее значимыми, и при их эффективном решении нахождение остальных характеристик сети Петри будет базироваться на полученных данных.

На рисунке 2.26 приведена схема, иллюстрирующая возможный алгоритм построения дерева маркировок и соответствующего словаря обыкновенной сети Петри. Каждый шагэтого алгоритма распадается на два этапа. Первый этап – это срабатывание всех переходов, которые могут сработать на

данном шаге, и получение новых маркировок. Второй этап - упорядочение списка маркировок, определение повторяющихся маркировок, циклов, тупиковых маркировок. Третий этап - перераспределение поддеревьев между вычислительными потоками для нового шага.

Для ограниченных сетей Петри общее возможное количество маркировок конечно (хоть и может быть весьма значительным [9]), в частности, для безопасных сетей оно равно 2 ", где *п* - число позиций в сети. Таким образом, в начале работы алгоритма количество новых маркировок будет расти, а затем, по мере заполнения списка маркировок, это количество начнет сокращаться. При отсутствии на очередном шаге алгоритма новых маркировок вычислительный процесс останавливается.

