БИЛЕТ № 22

1) Виды сигналов. Добавить с конспекта

Сигналы изначально принимались, как способы извещения об ошибках. Это средство представляло собой простейшие команды. Однако они ресурсоемкие, так как отправка требует системного вызова, а доставка – прерывания процесса-получателя. Они слабоинформативны и число их ограничено. То есть, они служат для информирования процесса или процессов о наступлении события.

Сигналы бывают синхронными и асинхронными. Поэтому в системе реализован механизм управления сигналами, который состоит из 2-х фаз:

1) Генерация и отправка.
2) Доставка и обработка.

Время между двумя фазами может быть достаточно большим.

Виды сигналов зависят от того, какой процесс генерирует сигнал и кто является получателем.

Причины, вызывающие отправку сигнала:

- особые ситуации (деление на 0);
- терминальные прерывания (нажатия кнопки);
- другие процессы;
- системы управления заданиями;
- сигналы для управления фоновыми и текущими задачами;
- сигналы квоты (если процесс превышает квоту – ему отправляется сигнал);
- сигналы уведомления (о наступлении какого-либо события);
- alarm (связанный со счетчиком таймера).

Доставка и обработка сигналов

Для каждого сигнала предусмотрена обработка по умолчанию. Однако пользователь может предусмотреть по сигналу завершение процесса.

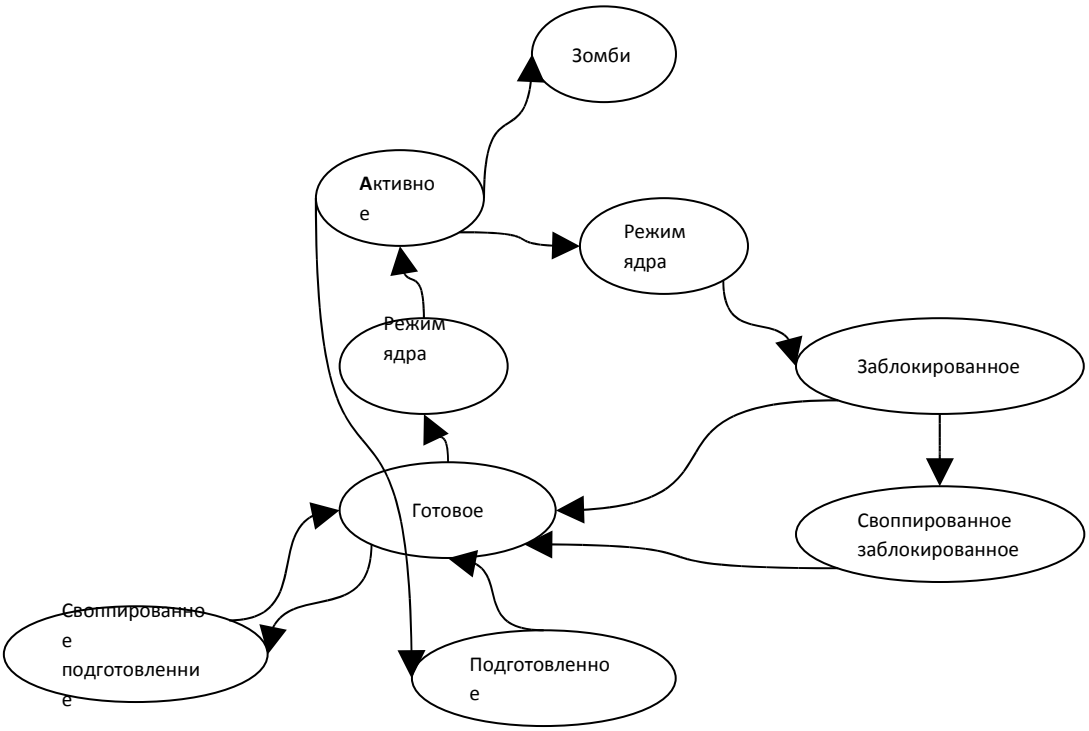
При сигнале процесс может:

- завершить свое выполнение;
- проигнорировать сигнал;
- остановиться и продолжиться потом;

отложить обработку сигнала на время.

2) Состояние процессов.

Процесс – любая исполняемая программа.
Процесс – траектория процессора в адресном пространстве машины.
В рамках одного процесса активизируются разные потоки. ОС знает о процессе, а о потоке не знает. Есть проблема управления. Нити – 3-й уровень – делят поток.



128 Дать определение – процесс Это любая выполняемая работа в системе; это динамический объект (внутренняя единица работы) системы, которому она выделяет ресурсы;

129 Условие перехода процесса из подготовительного состояния в готовность. Выделение ресурсов.

130 Условие перехода процесса из состояния готовности в активное. Выделение кванта времени процессора, запуск.

131 Условие перехода процесса из активного состояния в блокированное. Ожидание события, освобождения ресурса, конца операции ввода/вывода.

132 Условие перехода процесса из блокированного состояния в готовое. Наступление события, освобождения ресурса, завершение ввода/вывода.

133 Условие перехода процесса из активного состояния в готовое. Истечение кванта времени. (Выделенное время процессора для выполнение этого процесса завершилось и процессор занят другим процессом).

134 Условие создания процесса. Корректность описания процесса на языке описания процессов и выделение ресурсов

135 Отличие создания процесса при рекурсивном обращении (?) Хранить все локальные переменные в стеке.

14 ПРОЦЕСС – Любая вычислительная задача решаемая системой (динамическим объектом системы, которому выделяются ресурсы, это траектория процессора в адресном пространстве ВС)

По способу создания процессы делятся на:

- Системные процессы. Они создаются при загрузке системы, имеют оверлейную или динамически-последовательную структуру
- Проблемные (пользовательские

333333333333333333333) процессы. Создаются при активизации работы (задачи пользователя) операционной системой. Под активизацией процесса здесь понимается начало выполнения программы, подчиненной этому процессу, т.е. начало процесса. Пользовательский процесс может быть активизирован только другим процессом

По способу существования процессы делятся на:

- Последовательные. Это процессы, которые выполняются друг за другом, т.е. когда закончится выполнение первого процесса, начинается второй и т.д.
- Параллельные. Процессы, которые могут выполняться одновременно. Параллельные процессы могут быть:

o Независимые, т.е. выполняют разные операции и не имеют общих частей.

o Асинхронные ? это процессы, которые должны синхронизироваться и взаимодействовать друг с другом.

ОПЕРАЦИИ НАД ПРОЦЕССАМИ

1. Создание процесса включает в себя:

а) присвоение имени процессу,

б) включение этого имени в список имен процессов, известных системе,

в) определение начального приоритета,

г) формирование блока управления процессом,

д) выделение начальных ресурсов.

2. Уничтожение процесса означает его удаление из системы. Ресурсы, выделенные этому процессу, возвращаются системе, имя в системных списках стирается и блок управления процессом освобождается.

3. Изменение приоритета означает просто модификацию значения приоритета в блоке управления данным процессом. В основном увеличивают приоритет у процессов, когда предполагают, что они будут находиться в состоянии бесконечного откладывания или уменьшают приоритет процессу, надолго захватившему процессор.

4. Запуск (выбор) процесса, осуществляемый планировщиком, который выделяет процессу время процессора.

5. Приостановленный процесс не может продолжить свое выполнение до тех пор, пока его не активизирует любой другой процесс (кратковременное исключение определенных процессов в периоды пиковой нагрузки). В случае длительной приостановки процесса, его ресурсы должны быть освобождены. Решение об освобождении ресурса зависит от его природы. Основная память должна освобождаться немедленно, а вот принтер может и не быть освобожден. Приостановку процесса обычно производят в следующих случаях:

а) если система работает ненадежно и может отказать, то текущие процессы надо приостановить, исправить ошибку и затем активизировать приостановленные процессы;

б) если промежуточные результаты работы процесса вызвали сомнение, то нужно его приостановить, найти ошибку и запустить либо сначала, либо с контрольной точки;

в) если ВС перегружена, что вызвало снижение ее эффективности;

г) если для продолжения нормального выполнения процессу необходимы ресурсы, которые ВС не может ему предоставить.

6. Возобновление (или активизация) процесса ? операция подготовки процесса к повторному запуску выполняемая операционной системой, с той точки, в которой он был приостановлен

ИЕРАРХИЯ ПРОЦЕССОВ Процесс может породить новый процесс. При этом первый, порождающий процесс, называется родительским, а второй, порожденный процесс, ? дочерним. Для создания некоторого процесса необходим только один родительский процесс

3) Экзоядро.

Система с экзоядром

Экзоядро — ядро операционной системы компьютеров, предоставляющее лишь функции для взаимодействия между процессами и безопасного выделения и освобождения ресурсов.

На нижнем уровне ядра работает программа, которая называется экзоядро. В ее задачу входит распределение ресурсов для виртуальных машин, а после этого – проверка их использования (то есть отслеживание попыток машин использовать чужой ресурс). Каждая виртуальная машина на уровне пользователя может работать с собственной операционной системой, с той разницей, что каждая машина ограничена набором ресурсов, которые она запросила и которые ей были предоставлены**. Преимущества схемы экзоядра**: позволяет обойтись без уровня отображения. При других методах работы каждая виртуальная машина считает, что она использует свой собственный диск с нумерацией блоков от 0 до максимума. Поэтому монитор виртуальной машины должен

поддерживать таблицы преобразования адресов на диске (и всех других ресурсов). Необходимость преобразования отпадает при при наличии экзоядра, которому нужно только хранить запись о том, какой виртуальной машине выделен данный ресурс. Такой подход имеет еще одно преимущество: он отделяет многозадачность (в экзоядре) от операционной системы пользователя (в пространстве пользователя) с меньшими затратами, так как для этого ему необходимо всего лишь не допускать вмешательства одной виртуальной машины в работу другой.

4) Последовательность загрузки процесса в виртуальную память.

Виртуальная память используется для работы с программами, размер которых физически больше, чем размер ОП. Адресное пространство ограничено размерами адресного поля (4 Гб для 32-х разрядного поля).

Совокупность страниц, находящихся в кэш-памяти называют рабочей областью.

Принцип пространственной и временной локальности заключается в том, что с большой вероятностью можно предположить, что следующая команда будет n+1, если сейчас выполняется команда n. Если мы загрузили участок программы в память, то какой-то промежуток времени он будет выполняться (временная локальность). Если мы загрузили определенное число страниц, которые выполнены по свойству временной и пространственной локальности, то они образуют рабочую зону.

Существуют алгоритмы формирования рабочей зоны. Проблема: стратегия локальности и глобальности размещения локальная. Если администратор выделил нам определенную область ОП, – то это приведет к удалению страниц при страничных прерываниях (замена страниц того же процесса).

Глобальная – замещение страниц происходит во всем адресном пространстве (нет места для определенного пользователя – процесса).

При загрузке процесса часть его виртуальных страниц помещается в оперативную память, а остальные - на диск. Смежные виртуальные страницы не обязательно располагаются в смежных физических страницах. При загрузке операционная система создает для каждого процесса информационную структуру - таблицу страниц, в которой устанавливается соответствие между номерами виртуальных и физических страниц, загруженных в оперативную память, или делается отметка о том, что виртуальная страница выгружена на диск. Кроме того, в таблице страниц содержится управляющая информация, такая как признак модификации страницы, признак запрещения выгрузки страницы, признак обращения к странице и другие данные, формируемые и используемые механизмом виртуальной памяти.

При активизации очередного процесса в специальный регистр процессора загружается адрес таблицы его страниц. При каждом обращении к памяти происходит чтение из таблицы страниц информации о виртуальной странице, к которой произошло обращение. Если данная виртуальная страница находится в оперативной памяти, то выполняется преобразование виртуального адреса в физический. Если же нужная виртуальная страница в данный момент выгружена на диск, то происходит так называемое страничное прерывание. Выполняющийся процесс переводится в состояние ожидания, и активизируется другой процесс из очереди готовых. Параллельно программа обработки страничного прерывания находит на диске требуемую виртуальную страницу и пытается загрузить ее в оперативную память. Если в памяти имеется свободная физическая страница, то загрузка выполняется немедленно, если же свободных страниц нет, то в соответствии с выбранным критерием решается вопрос, какую страницу следует выгрузить из оперативной памяти.

После того, как выбрана страница, которая должна покинуть оперативную память, анализируется ее признак модификации из таблицы страниц. Если выгружаемая страница с момента загрузки была модифицирована, то ее новая версия должна быть переписана на диск. Если нет, то она может быть просто уничтожена, то есть соответствующая физическая страница объявляется свободной.

При страничной организации виртуальное адресное пространство процесса делится механически на равные части. Это не позволяет дифференцировать способы доступа к разным частям программы - сегментам, что часто бывает очень полезным. Например, можно запретить обращаться с операциями записи и чтения в кодовый сегмент программы, а для сегмента данных разрешить только чтение. Кроме того, разбиение программы на смысловые части делает принципиально возможным разделение одного сегмента несколькими процессами. Например, если два процесса используют одну и ту же математическую подпрограмму, то в оперативную память может быть загружена только одна копия этой подпрограммы. Сегментное распределение памяти реализует эти возможности.

Виртуальное адресное пространство процесса делится на сегменты, размер которых определяется программистом с учетом смыслового значения содержащейся в них информации. Отдельный сегмент может представлять собой подпрограмму, массив данных и тому подобное. Иногда сегментация программы выполняется по умолчанию компилятором.

При загрузке процесса часть сегментов помещается в оперативную память, а другая их часть размещается в дисковой памяти. Сегменты одной программы могут занимать в оперативной памяти несмежные участки. Во время загрузки система создает таблицу сегментов процесса, в которой для каждого сегмента указывается его начальный физический адрес в оперативной памяти, размер, правила доступа, признак модификации, признак обращения к данному сегменту за последний интервал времени и некоторая другая информация. Если виртуальные адресные пространства нескольких процессов включают один и тот же сегмент, то в таблицах сегментов этих процессов делаются ссылки на один и тот же участок оперативной памяти, в который данный сегмент загружается в единственном экземпляре.

5) Избежание тупиков.

Тупик — это:

1. ситуация, из которой система не может выйти;
2. ситуация, когда процесс ждет события, которое никогда не произойдет.
   Примером из жизни может служить транспортная пробка на перекрестке.

Второй пример: круговая цепь ожидания (рис. 2.16).

Здесь ресурс 2 выделен процессу 1, а ресурс 1 — процессу 2. В какой-то момент процесс 1 затребовал дополнительно ресурс 1, а процесс 2 — ресурс 2. Естественно, ни один из этих ресурсов не может быть выделен затребовавшему его процессу. Получаем тупиковую ситуацию, при которой ни один из двух процессов не может закончить свое выполнение, пока не будет освобожден требуемый ресурс, чего в данном случае никогда не сможет произойти. Выход заключается в удалении одного из процессов попавших в тупиковую ситуацию из системы.

Тупик ***неизбежен***, если присутствует четыре условия его возникновения:

1. Условие *взаимного исключения*. Процесс обладает монопольным правом владения ресурсами во время всего существования процесса.
2. Условие *ожидания*. Процесс, обладая монопольным правом, пытается захватить новые ресурсы.
3. Условие *перераспределения*. Ресурс нельзя отнять до полного завершения процесса.
4. Условие: *круговая цепь ожидания*. Каждый из процессов цепочки требует дополнительный ресурс, который уже выделен процессу данной цепочки.

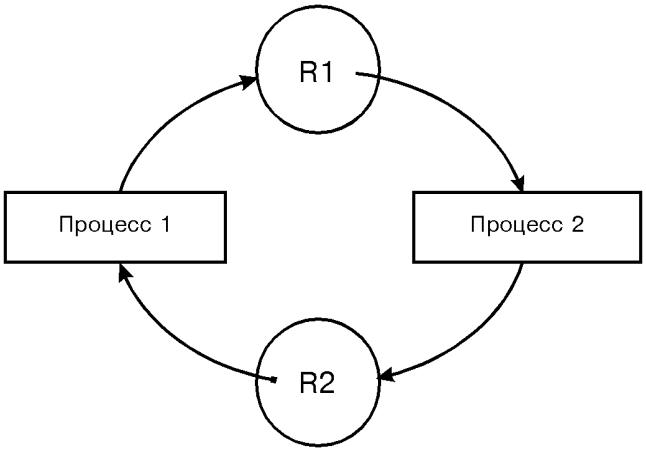


Рис. 2.16

- *Тупики в системе спулинга*

При вводе заданий используется режим **спулинга** — режим буферизации для выравнивания скоростей при вводе и считывании информации из буфера. Система, использующая режим спулинга, сильно подвержена тупикам.

В таких системах буфер может быть полностью заполнен до того, как данные из него станут поступать на печатающее устройство, а его размера недостаточно для буферизации всех данных вывода. При этом возникает тупиковая ситуация, т.к. размер буфера недостаточен для помещения всех поступивших данных, а принтер не сможет начать вывод. В данном случае единственным выходом является рестарт системы, но при этом теряется вся информация.

В таких системах хорошим выходом было бы неполное заполнение буфера (на 75-80%) и передача после этого управления печатающему устройству. По освобождении буфера можно его снова неполностью заполнить, затем снова перейти к распечатке и т.д.

Во многих современных ОС принтер начинает распечатку до заполнения буфера (по мере заполнения). При этом часть буфера (распечатанная) освобождается и туда можно заносить новую информацию. В таких системах вероятность возникновения тупиковой ситуации значительно меньше.

2.7.13. Способы борьбы с тупиками

Последствия тупиков равносильны ситуации бесконечного откладывания. Тупики и ситуация бесконечного откладывания — это ситуации равносильные "потери процесса".

Можно выделить 4 стратегии борьбы с тупиками:

1. **Предотвращение тупика**. Требуется создать условия, исключающие возможность возникновения тупиковых ситуаций. Однако этот метод часто приводит к нерациональному использованию ресурсов.
2. **Обход тупика**. Этот метод предусматривает менее жесткие ограничения, чем первый и обеспечивает лучшее использование ресурсов. Метод учитывает возможность возникновения тупиков, и при увеличении вероятности тупиковой ситуации принимаются меры по обходу тупика.
3. **Обнаружение тупиков**. Требуется установить сам факт возникновения тупиковой ситуации, причем точно определить те процессы и ресурсы, которые в нее включены.
4. **Восстановление после тупиков**. Необходимо устранить тупиковую ситуацию, чтобы система смогла продолжить работу, а процессы, попавшие в тупиковую ситуацию, смогли завершиться и освободить занимаемые ими ресурсы. Восстановление — это серьезная проблема, так что в большинстве систем оно осуществляется путем полного выведения из решения одного или более процессов, попавших в тупиковую ситуацию. Затем выведенные процессы обычно перезапускаются с самого начала, так что вся или почти вся проделанная процессами работа теряется.

2.7.13.1. Предотвращение тупиков

Возникновение тупика невозможно, если нарушается одно из четырех необходимых условий возникновения тупиковых ситуаций. Первое условие при этом можно и не нарушать, т.е. считаем что процесс может обладать монопольным правом владения ресурсами. Нарушая остальные три условия получаем следующие три способа предотвращения тупиков:

I способ. Процесс запрашивает и получает все ресурсы сразу. Если процесс не имеет возможности получить все требуемые ресурсы сразу, т.е. некоторые из них на данный момент заняты, то он должен ожидать освобождения требуемых ресурсов. При этом он не должен удерживать за собой какие-либо ресурсы. Нарушается условие "ожидания дополнительных ресурсов" (2-е) и тупиковая ситуация невозможна. При этом способе имеем простой процессов, ожидающих выделения ресурсов, и работу вхолостую значительной части ресурсов. Можно прибегнуть к разделению программы на несколько программных шагов, работающих независимо друг от друга. При этом выделение ресурсов можно осуществлять для каждого шага программы, однако увеличиваются расходы на этапе проектирования прикладных программ.

Этот способ имеет два существенных недостатка:

1). Снижается эффективность работы системы;

2). Усиливается вероятность бесконечного откладывания для всех процессов.

II способ. Если процесс, удерживающий за собой определенные ресурсы, затребовал дополнительные и получил отказ в их получении, он должен освободить все ранее полученные ресурсы. При необходимости процесс должен будет запросить их снова вместе с дополнительными. Здесь нарушается условие "перераспределения" (3-е). При этом способе каждый процесс может несколько раз запрашивать, получать и отдавать ресурсы системе. При этом система будет выполнять массу лишней работы — происходит деградация системы с точки зрения полезной работы. Недостатки этого способа:

1. Если процесс в течение некоторого времени удерживал ресурсы, а затем освободил их, он может потерять имевшуюся информацию.
2. Здесь также возможно бесконечное откладывание процессов — процесс запрашивает ресурсы, получает их, однако не может завершиться, т.к. требуются дополнительные ресурсы; далее он, не завершившись, отдает имеющиеся у него ресурсы системе; затем он снова запрашивает ресурсы и т.д.

Имеем бесконечную цепочку откладывания завершения процесса.

III способ. Стратегия линейности. Все ресурсы выстроены в порядке присвоенных приоритетов и захват новых ресурсов может быть выполнен только в порядке возрастания приоритетов. При этом нарушается условие "круговая цепь ожидания" (4-е).

Трудности и недостатки данного способа:

1. Поскольку запрос ресурсов осуществляется в порядке возрастания приоритетов, а они назначаются при установке машины, то в случае введения новых типов ресурсов может возникнуть необходимость переработки существующих программ и систем;
2. Приоритеты ресурсов должны отражать нормальный порядок, в котором большинство заданий используют ресурсы. Однако, если задание требует ресурсы не в этом порядке, то оно будет захватывать и удерживать некоторые ресурсы задолго до того, как появится необходимость их использования — теряется эффективность.
3. Способ не предоставляет удобства пользователям.

2.7.13.2. Обход тупиков

Алгоритм "банкира".

При использовании алгоритма "банкира" подразумевается, что :

1) системе заранее известно количество имеющихся ресурсов;
2) система знает, сколько ресурсов может максимально потребоваться процессу;
3) число пользователей (процессов), обращающихся к ресурсам, известно и постоянно;
4) система знает, сколько ресурсов занято в данный момент времени этими процессами (в общем и каждым из них);
5) процесс может потребовать ресурсов не больше, чем имеется в системе.

В алгоритме "банкира" введено понятие надежного состояния. Текущее состояние вычислительной машины называется **надежным**, если ОС может обеспечить всем текущим пользователям (процессам) завершение их заданий в течение конечного времени. Иначе состояние считается **ненадежным**. Надежное состояние — это состояние, при котором общая ситуация с ресурсами такова, что все пользователи имеют возможность со временем завершить свою работу. Ненадежное состояние может со временем привести к тупику.

Система удовлетворяет только те запросы, при которых ее состояние остается надежным, т.е. при запросе ресурса система определяет сможет ли она завершить хотя бы один процесс, после этого выделения.

Пример решения задачи выделения ресурсов по алгоритму "банкира".

Имеем 6 ресурсов и 6 процессов (рис .2.17.). В таблице показано состояние занятости ресурсов на данный момент (1 свободный ресурс).

Процесс	Выделенное число ресурсов	Требуемое число ресурсов
A	2	3
B	1	3
C	0	2
D	1	2
E	1	4
F	0	5

Рис. 2.17

Эта таблица отражает надежное состояние, т.к. существует такая последовательность выделений — освобождений ресурсов, при которой все процессы за конечное время будут завершены.

Недостатки алгоритма банкира:

1) Если количество ресурсов большое, то становится достаточно сложно вычислить надежное состояние.
2) Достаточно сложно спланировать, какое количество ресурсов может понадобиться системе.
3) Число работающих пользователей (процессов) остается постоянным.
4) Пользователи должны заранее указывать максимальную потребность в ресурсах, однако, потребность может определяться динамически по мере выполнения процесса.
5) Пользователь всегда заказывает ресурсов больше, чем ему может потребоваться.
6) Алгоритм требует, чтобы процессы возвращали выделенные им ресурсы в течение некоторого конечного времени. Однако, для систем реального времени требуются более конкретные гарантии. Т.е. в системе должно быть задано максимальное время захвата всех ресурсов процессом (через это время ресурсы должны быть освобождены).

Графический метод обхода тупика (рис. 2.18).

Имеются 2 процесса Р1 и Р2 и каждому из них для выполнения требуются по 2 ресурса: Д — диск , П — процессор. Если Р1 захватил процессор , а Р2 — диск, то возникает тупиковая ситуация. Однако, если Р1 захватил П и Д и завершил свое выполнение, то Р2 захватывает эти же ресурсы и заканчивает выполнение — беступиковое взаимодействие.

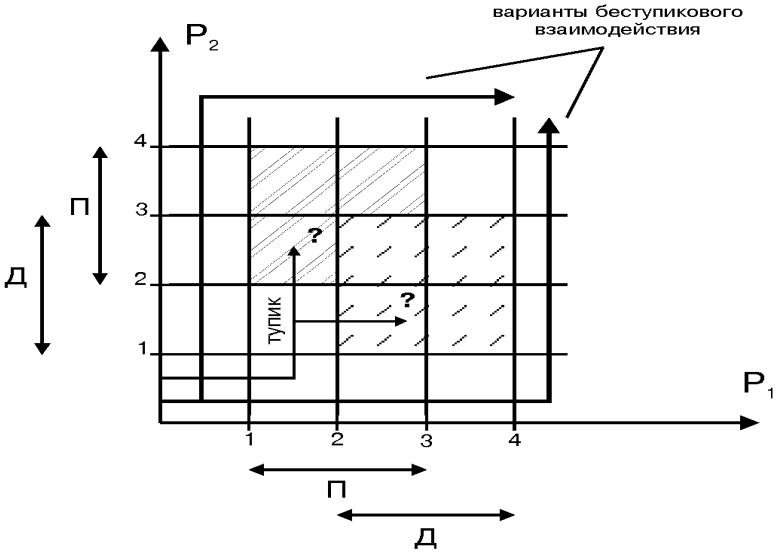


Рис. 2.18

2.7.13.3. Обнаружение тупиков

Обнаружение тупика — это установление факта того, что возникла тупиковая ситуация и определение процессов и ресурсов, вовлеченных в тупиковую ситуацию. Для обнаружения тупиковой ситуации используют операцию **редукции графа**. Эта операция выполняется системой, когда есть подозрение, что какие-то процессы находятся в тупике. При этом определяется, какие процессы могут успешно завершиться (по графу) и освободить ресурсы. Если запросы ресурсов для некоторого процесса могут быть удовлетворены, то граф можно редуцировать на этот процесс. При этом процесс удаляется из графа, включая все дуги от него к требуемым ресурсам и от выделенных ресурсов к нему, т.е. эти ресурсы могут считаться свободными и выделяться другим процессам. Если граф можно редуцировать на все процессы, значит, тупиковой ситуации нет, а если этого сделать нельзя, то все "нередуцируемые" процессы образуют группу процессов, находящихся в тупиковой ситуации.

Пример редуцируемого графа , содержащего 5 процессов и 4 типа ресурсов.

Примечание к рис.2.19.: дуга от процесса к ресурсу (кругу) — требование ресурса этого типа; тип ресурса - кружок, а сами ресурсы - квадратики внутри круга; P — обозначение процесса; R — типа ресурса; дуга от ресурса (квадрата) к процессу — ресурс выделен процессу.

На рис. 2.19 показан граф беступикового взаимодействия.

- *Восстановление после тупиков.*

Систему, оказавшуюся в тупике, необходимо вывести из него, нарушив одно или несколько необходимых условий его существования. При этом несколько процессов потеряют (полностью или частично) проделанную работу. Однако, это дешевле, чем оставлять систему в тупиковой ситуации. Возникают следующие сложности восстановления системы:

1) В первый момент может быть неочевидно, что система попала в тупиковую ситуацию.
2) В большинстве систем нет достаточно эффективных средств, позволяющих приостановить процесс на неопределенно долгое время, вывести его из системы и возобновить впоследствии. Некоторые процессы (например, процессы реального времени) должны работать непрерывно и не допускают приостановки и последующего восстановления.
3) Если в системе существуют эффективные средства приостановки/возобновления, то их использование требует значительных затрат машинного времени и внимания высококвалифицированного оператора.

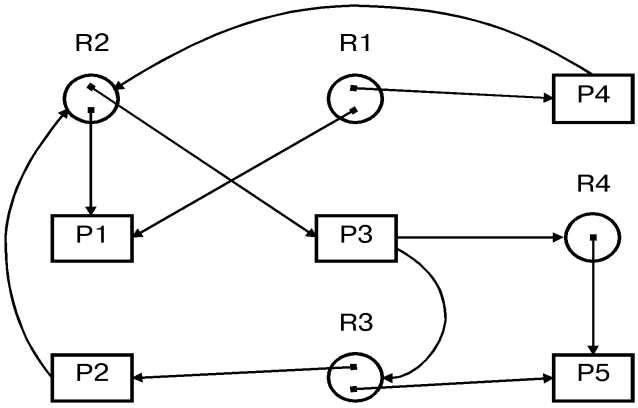


Рис. 2.19

1) Восстановление после тупика небольших масштабов требует и небольшого количества работы; крупный тупик может потребовать довольно большого объема работ по его исключению из системы.

В современных системах восстановление обычно выполняют путем принудительного вывода одного или нескольких процессов из системы, чтобы их ресурсы могли быть использованы. Тогда эти процессы теряются вместе с информацией в них, однако, оставшиеся процессы получают возможность завершить свою работу. Т.е. в этом случае несколько процессов просто убиваются.

При удалении процессов могут возникнуть проблемы с приоритетностью (удаляются обычно наименее приоритетные процессы):

1) Процессы могут не иметь конкретных приоритетов.
2) Значения приоритетов могут оказаться неправильными или могут быть нарушены (например, для бесконечно откладывающегося процесса увеличивается приоритет).
3) Трудность в оптимальном определении, какие из процессов вывести из системы.

Самый эффективный способ восстановления после тупиков — **механизм приостановки/возобновления**. Он позволяет кратковременно переводить процессы в состояние ожидания, а затем активизировать ожидающие процессы, причем без потери информации.

Часто при тупиковых ситуациях требуется перезапуск (рестарт) системы. При этом существует **перезапуск сначала** (вся полученная до рестарта информация теряется) или **с контрольной точки** (теряется только информация после контрольной точки). Система не определяет контрольных точек — их должен предусмотреть программист (для конкретного процесса).