Глава 6
Взаимоблокировка

В компьютерных системах множество ресурсов, которые одновременно могут использоваться только одним процессом. Стоит лишь вспомнить принтеры, накопители на магнитной ленте для резервного копирования данных компаний и элементы во внутренних системных таблицах. Если два процесса одновременно выводят информацию на принтер, то получается полная тарабарщина. Если два процесса будут использовать один и тот же элемент таблицы файловой системы, то эта система непременно будет повреждена. Поэтому все операционные системы способны временно предоставлять процессу исключительные права доступа к конкретным ресурсам.

При работе многих приложений процессу нужен исключительный доступ не к одному, а сразу к нескольким ресурсам. Предположим, к примеру, что каждый из двух процессов захотел записать отсканированный документ на Blu-ray-диск. Процесс *A* запрашивает разрешение на использование сканера и получает его. Процесс *B* запрограммирован по-другому: сначала он запрашивает разрешение на использование пишущего привода Blu-ray-дисков и также получает это разрешение. Теперь *A* запрашивает разрешение на использование пишущего привода Blu-ray-дисков, но запрос отклоняется до тех пор, пока это устройство не будет освобождено процессом *B*. К сожалению, вместо того чтобы освободить привод, *B* запрашивает разрешение на использование сканера. И в этот момент оба процесса оказываются заблокированными навсегда. Такая ситуация называется тупиковой ситуацией (тупиком), или **взаимоблокировкой** (deadlock).

Взаимоблокировки могут случаться и между машинами. К примеру, многие офисы оборудованы локальной сетью, к которой подключено множество компьютеров. Довольно часто такие устройства, как сканеры, пишущие приводы Blu-ray-дисков и DVD, принтеры и приводы накопителей на магнитной ленте, подключены к сети в качестве ресурсов общего пользования, доступных любому пользователю на любой машине. Если эти устройства могут быть дистанционно зарезервированы (например, с домашнего компьютера пользователя), то может возникнуть взаимоблокировка, похожая на только что рассмотренную. При более сложных обстоятельствах во взаимоблокировку могут быть вовлечены три, четыре и более устройств и пользователей.

Взаимоблокировки могут возникать и при массе других обстоятельств. К примеру, в системах управления базами данных, во избежание попадания в состояние состязания программе может понадобиться блокировка нескольких используемых ею записей. Если процесс *A* блокирует запись *R*1, а процесс *B* блокирует запись *R*2, а затем каждый процесс пытается заблокировать запись другого процесса, то получается та же взаимоблокировка. Таким образом, взаимоблокировки могут появляться при работе как с аппаратными, так и с программными ресурсами.

В данной главе будут рассмотрены разновидности взаимоблокировок, условия их возникновения, а также некоторые пути предотвращения взаимоблокировок или уклонения от их возникновения. Хотя этот материал касается взаимоблокировок в контексте

операционных систем, они также случаются в системах управления базами данных и во многих других компьютерных областях, поэтому приводимые здесь сведения применимы к широкому спектру многозадачных систем. На тему взаимоблокировок существует множество научных трудов. Библиографии по этой теме дважды публиковались в *Operating Systems Review*, и на них стоит обратить внимание в качестве источников справочной информации (Newton, 1979; Zobel, 1983). Хотя этим библиографиям уже много лет, основная часть работ по взаимоблокировкам была завершена до 1980 года, поэтому они все еще не утратили своей актуальности.

6.1. Ресурсы

Основная часть взаимоблокировок связана с ресурсами, к которым некоторым процессам были предоставлены исключительные права доступа. К их числу относятся устройства, записи данных, файлы и т. д. Чтобы придать рассмотрению взаимоблокировок как можно более универсальный характер, мы будем называть объекты, к которым предоставляется доступ, **ресурсами**. Ресурсами могут быть аппаратные устройства (например, привод Blu-ray-дисков) или какая-то часть информации (например, запись базы данных). Обычно у компьютера может быть множество ресурсов, которые могут быть предоставлены процессу. Некоторые ресурсы могут быть доступны в нескольких идентичных экземплярах, например три привода Blu-ray-дисков. Когда доступны несколько копий ресурса, то для удовлетворения любого запроса на ресурс может быть использован один из них. Короче говоря, под ресурсом понимается все, что должно предоставляться, использоваться и через некоторое время высвобождаться, поскольку в один и тот же момент времени может использоваться только одним процессом.

6.1.1. Выгружаемые и невыгружаемые ресурсы

Ресурсы бывают двух видов: выгружаемые и невыгружаемые. К **выгружаемым** относятся такие ресурсы, которые могут быть безболезненно отобраны у процесса, который ими обладает. Примером такого ресурса может послужить память. Рассмотрим систему, имеющую 1 Гбайт пользовательской памяти, один принтер и два процесса по 1 Гбайт, каждый из которых хочет что-то вывести на печать. Процесс *A* запрашивает и получает принтер, а затем начинает вычислять значение, предназначенное для вывода на печать. Но до завершения вычисления истекает выделенный ему квант времени, и он выгружается на диск.

Теперь запускается процесс *B*, безуспешно, как оказывается, пытаясь завладеть принтером. Потенциально возникает ситуация взаимоблокировки, поскольку у процесса *A* есть принтер, а у процесса *B* — память и ни один из них не может продолжить свою работу без ресурса, удерживаемого другим процессом.

К счастью, есть возможность отобрать память у процесса *B*, выгрузив этот процесс на диск, и загрузить оттуда процесс *A*. Теперь *A* может возобновить свою работу, выполнить распечатку и высвободить принтер. И никакой взаимоблокировки не возникнет.

А вот **невыгружаемый ресурс** нельзя отобрать у его текущего владельца, не вызвав потенциально сбоя в вычислениях. Если у процесса, который уже приступил к записи на Blu-ray-диск, внезапно отобрать пишущий привод и отдать его другому процессу,

это приведет к порче Blu-ray-диска. Пишущие приводы Blu-ray-дисков нельзя отобрать в произвольный момент.

Выгружаемость ресурса зависит от контекста. На стандартном персональном компьютере память является выгружаемым ресурсом, поскольку страницы всегда могут быть выгружены на диск, чтобы нужный объем свободной памяти был восстановлен. А на смартфоне, не поддерживающем свопинг или страничную организацию памяти, простой выгрузкой взаимоблокировки из-за дефицита памяти избежать не удастся.

Как правило, во взаимоблокировках фигурируют невыгружаемые ресурсы. Обычно потенциальные взаимоблокировки с участием выгружаемых ресурсов могут быть устранены путем перераспределения ресурсов от одного процесса к другому. Поэтому наше внимание будет сконцентрировано на невыгружаемых ресурсах.

В наиболее общем виде при использовании ресурса происходит следующая последовательность событий:

1. Запрос ресурса.
2. Использование ресурса.
3. Высвобождение ресурса.

Если во время запроса ресурс недоступен, запрашивающий процесс вынужден перейти к ожиданию. В некоторых операционных системах при отказе в выделении запрошенного ресурса процесс автоматически блокируется, а когда ресурс становится доступен — возобновляется. В других системах отказ в выделении запрашиваемого ресурса сопровождается кодом ошибки, и принятие решения о том, что следует делать, немного подождать или попытаться снова получить ресурс, возлагается на вызывающий процесс.

Процесс, чей запрос на выделение ресурса был только что отклонен, обычно входит в короткий цикл: запрос ресурса, затем приостановка, — после чего повторяет попытку. Хотя этот процесс не заблокирован, но по всем показателям он является фактически заблокированным, поскольку не может выполнять никакой полезной работы. При дальнейшем рассмотрении вопроса мы будем предполагать, что при отказе в выделении запрошенного ресурса процесс впадает в спячку.

Особенности запроса ресурса существенно зависят от используемой системы. В некоторых системах для запроса предоставляется системный вызов *request*, позволяющий процессам запросить ресурс в явном виде. В других системах единственными ресурсами, о которых знает операционная система, являются специальные файлы, которые в конкретный момент времени могут быть открыты только одним процессом. Они открываются с использованием обычного вызова *open*. Если файл уже используется, вызывающий процесс блокируется до тех пор, пока файл не будет закрыт текущим владельцем.

6.1.2. Получение ресурса

Для некоторых видов ресурсов, таких как записи в базе данных, управление использованием ресурсов зависит от самих пользовательских процессов, а не от системы. Один из способов, позволяющих ввести пользовательское управление ресурсами, заключается в присоединении семафора к каждому из ресурсов.

Все эти семафоры получают исходное значение, равное 1. С таким же успехом могут использоваться и мьютексы. Перечисленные ранее три этапа затем воплощаются в применение к семафору вызова *down* для получения ресурса, использование ресурса и, в завершение, применение вызова *up* при высвобождении ресурса. Эти этапы показаны в листинге 6.1, *а*.

Листинг 6.1. Использование семафоров для защиты: *а* — одного ресурса; *б* — двух ресурсов

```
typedef int semaphore;
semaphore resource_1;

void process_A(void) {
    down(&resource_1);
    use_resource_1( );
    up(&resource_1);
}
```

а

```
typedef int semaphore;
semaphore resource_1;
semaphore resource_2;

void process_A(void) {
    down(&resource_1);
    down(&resource_2);
    use_both_resources( );
    up(&resource_2);
    up(&resource_1);
}
```

б

Иногда процессы нуждаются в двух и более ресурсах. Их можно получать последовательно, как показано в листинге 6.1, *б*. Если требуется больше двух ресурсов, их запрашивают непосредственно один за другим.

Пока все идет хорошо. Пока речь идет только об одном процессе, все работает нормально. Конечно, когда используется только один процесс, нет нужды в формальном получении ресурсов, поскольку нет соперничества за обладание ими.

Теперь рассмотрим ситуацию с двумя процессами — *A* и *B* — и двумя ресурсами. В листинге 6.2 показаны два сценария: *а* — оба процесса запрашивают ресурсы в одном и том же порядке; *б* — запрашивают ресурсы в разном порядке. Разница может показаться несущественной, но это не так.

Листинг 6.2. Код: *а* — не вызывающий взаимоблокировки; *б* — в котором кроется потенциальная возможность взаимоблокировки

```
typedef int semaphore;
    semaphore resource_1;
    semaphore resource_2;

    void process_A(void) {
        down(&resource_1);
        down(&resource_2);
        use_both_resources( );
        up(&resource_2);
        up(&resource_1);
    }

    void process_B(void) {
        down(&resource_1);
        down(&resource_2);
```

```
    semaphore resource_1;
    semaphore resource_2;

    void process_A(void) {
        down(&resource_1);
        down(&resource_2);
        use_both_resources( );
        up(&resource_2);
        up(&resource_1);
    }

    void process_B(void){
        down(&resource_2);
        down(&resource_1);
```

```
    use_both_resources( );
    up(&resource_2);
    up(&resource_1);
}
```

а

```
    use_both_resources( );
    up(&resource_1);
    up(&resource_2);
}
```

б

В листинге 6.2, *а* один из процессов запрашивает первый ресурс раньше, чем это делает второй процесс. Затем этот же процесс успешно получает второй ресурс и выполняет свою работу. Если второй процесс попытается получить ресурс 1 до его высвобождения, то он будет просто заблокирован до тех пор, пока ресурс не станет доступен.

В листинге 6.2, *б* показана другая ситуация. Может случиться, что один из процессов получит оба ресурса и надежно заблокирует другой процесс до тех пор, пока не сделает свою работу. Но может случиться и так, что процесс *A* получит ресурс 1, а процесс *B* получит ресурс 2. Каждый из них теперь будет заблокирован при попытке получения второго ресурса. Ни один из процессов не возобновит свою работу. Плохо то, что возникнет ситуация взаимоблокировки.

Здесь мы видим, что происходит из-за небольшой разницы в стиле программирования: в зависимости от того, какой из ресурсов будет получен первым, программа либо работает, либо дает трудноопределимый сбой. Поскольку взаимоблокировки могут возникать столь просто, для борьбы с ними были проведены обширные исследования. В этой главе подробно рассматриваются взаимоблокировки и средства борьбы с ними.

6.2. Введение во взаимоблокировки

Взаимоблокировкам можно дать следующее формальное определение.

Взаимоблокировка в группе процессов возникает в том случае, если каждый процесс из этой группы ожидает события, наступление которого зависит исключительно от другого процесса из этой же группы.

Поскольку все процессы находятся в состоянии ожидания, ни один из них не станет причиной какого-либо события, которое могло бы возобновить работу другого процесса, принадлежащего к этой группе, и ожидание всех процессов становится бесконечным. В этой модели предполагается, что у процессов есть только один поток, а прерывания, способные возобновить работу заблокированного процесса, отсутствуют. Условие отсутствия прерываний необходимо, чтобы не позволить заблокированному по иным причинам процессу возобновить свою работу, скажем, по аварийному сигналу, после чего вызвать событие, освобождающее другие имеющиеся в группе процессы.

В большинстве случаев событием, наступления которого ожидает каждый процесс, является высвобождение какого-либо ресурса, которым на данный момент владеет другой участник группы. Иными словами, каждый процесс из группы, попавшей в ситуацию взаимоблокировки, ожидает ресурса, которым обладает другой процесс из этой же группы. Ни один из процессов не может работать, ни один из них не может высвободить какой-либо ресурс, и ни один из них не может возобновить свою работу. Количество процессов и количество и вид удерживаемых и запрашиваемых ресурсов не имеет значения. Этот результат сохраняется для любого типа ресурсов, включая аппаратные и программные ресурсы. Этот вид взаимоблокировки называется **ресурсной взаимоблокировкой**. Наверное, это самый распространенный, но далеко не единствен-

ный вид. Сначала мы рассмотрим ресурсную взаимоблокировку, а в конце этой главы вернемся к краткому обзору других видов взаимоблокировки.

6.2.1. Условия возникновения ресурсных взаимоблокировок

Коффман (Coffman et al., 1971) показал, что для возникновения ресурсных взаимоблокировок должны выполняться четыре условия:

1. Условие взаимного исключения. Каждый ресурс либо выделен в данный момент только одному процессу, либо доступен.
2. Условие удержания и ожидания. Процессы, удерживающие в данный момент ранее выделенные им ресурсы, могут запрашивать новые ресурсы.
3. Условие невыгружаемости. Ранее выделенные ресурсы не могут быть принудительно отобраны у процесса. Они должны быть явным образом высвобождены тем процессом, который их удерживает.
4. Условие циклического ожидания. Должна существовать кольцевая последовательность из двух и более процессов, каждый из которых ожидает высвобождения ресурса, удерживаемого следующим членом последовательности.

Для возникновения ресурсной взаимоблокировки должны соблюдаться все четыре условия. Если одно из них не соблюдается, ресурсная взаимоблокировка невозможна.

Следует заметить, что каждое условие относится к той политике, которой придерживается или не придерживается система. Может ли данный ресурс быть выделен одновременно более чем одному процессу? Может ли процесс удерживать ресурс и запрашивать другой ресурс? Может ли ресурс быть отобран? Может ли получиться циклическое ожидание? Чуть позже мы увидим, как взаимоблокировке может противостоять попытка устранения некоторых из этих условий.

6.2.2. Моделирование взаимоблокировок

Холт (Holt, 1972) показал, как эти четыре условия могут быть смоделированы с использованием направленных графов. У графов имеется два вида узлов: процессы, показанные окружностями, и ресурсы, показанные квадратами. Направленное ребро, которое следует от узла ресурса (квадрата) к узлу процесса (окружности), означает, что этот ресурс был ранее запрошен, получен и на данный момент удерживается этим процессом. На рис. 6.1, *а* ресурс *R* в данное время выделен процессу *A*.

Направленное ребро, идущее от процесса к ресурсу, означает, что процесс в данное время заблокирован в ожидании высвобождения этого ресурса. На рис. 6.1, *б* процесс *B* ожидает высвобождения ресурса *S*. На рис. 6.1, *в* мы наблюдаем взаимоблокировку: процесс *C* ожидает высвобождения ресурса *T*, который в данный момент удерживается процессом *D*. Процесс *D* не собирается высвобождать ресурс *T*, поскольку он ожидает высвобождения ресурса *U*, удерживаемого процессом *C*. Оба процесса находятся в состоянии вечного ожидания. Циклическая структура графа означает, что мы имеем дело с взаимоблокировкой, включившей процессы и ресурсы в цикл (предполагается, что в системе есть только один ресурс каждого типа). В данном примере получился следующий цикл: *C*–*T*–*D*–*U*–*C*.

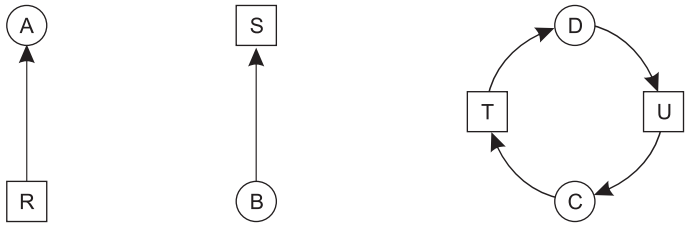

Рис. 6.1. Графы распределения ресурсов: *а* — ресурс занят; *б* — запрос ресурса; *в* — взаимоблокировка

Рассмотрим пример того, как могут быть использованы графы ресурсов. Представим, что есть три процесса, *A*, *B* и *C*, и три ресурса, *R*, *S* и *T*. Последовательности действий по запросу и высвобождению ресурсов, осуществляемые этими тремя процессами, показаны на рис. 6.2, *а–в*. Операционная система может в любое время запустить любой незаблокированный процесс, то есть она может принять решение запустить процесс *A* и дождаться, пока он не завершит всю свою работу, затем запустить процесс *B* и довести его работу до завершения и, наконец, запустить процесс *C*.

Такой порядок не приводит к взаимоблокировкам (поскольку отсутствует борьба за овладение ресурсами), но в нем нет и никакой параллельной работы. Кроме действий по запросу и высвобождению ресурсов процессы занимаются еще и вычислениями, и вводом-выводом данных. Когда процессы запускаются последовательно, отсутствует возможность использования центрального процессора одним процессом, в то время когда другой процесс ожидает завершения операции ввода-вывода. Поэтому строго последовательное выполнение процессов может быть не оптимальным решением. В то же время, если ни один из процессов не выполняет никаких операций ввода-вывода, алгоритм, при котором кратчайшее задание выполняется первым, более привлекателен, чем циклический алгоритм, поэтому при определенных условиях последовательный запуск всех процессов может быть наилучшим решением.

Теперь предположим, что процессы занимаются как вводом-выводом, так и вычислениями, поэтому наиболее разумным алгоритмом планирования их работы является циклический алгоритм. Запросы ресурсов могут происходить в том порядке, который показан на рис. 6.2, *г*. Если эти шесть запросов выполняются именно в этом порядке, им соответствую шесть результирующих ресурсных графов, показанных на рис. 6.2, *д–к*. После выдачи запроса 4 процесс *A*, как показано на рис. 6.2, *з*, блокируется в ожидании ресурса *S*. На следующих двух этапах процессы *B* и *C* также блокируются, что в итоге приводит к зацикливанию и возникновению взаимоблокировки, показанной на рис. 6.2, *к*.

Но как уже ранее упоминалось, от операционной системы не требовалось запускать процессы в каком-то определенном порядке. В частности, если удовлетворение конкретного запроса может привести к взаимоблокировке, операционная система может просто приостановить процесс, не удовлетворяя его запрос (то есть просто не планируя работу процесса) до тех пор, пока это не будет безопасно. На рис. 6.2, если операционная система знает о грядущей взаимоблокировке, она может приостановить процесс *B*, вместо того чтобы выделить ему ресурс *S*. Запуская только процессы *A* и *C*, мы получим такую последовательность действий по запросу и высвобождению ресурсов, которая

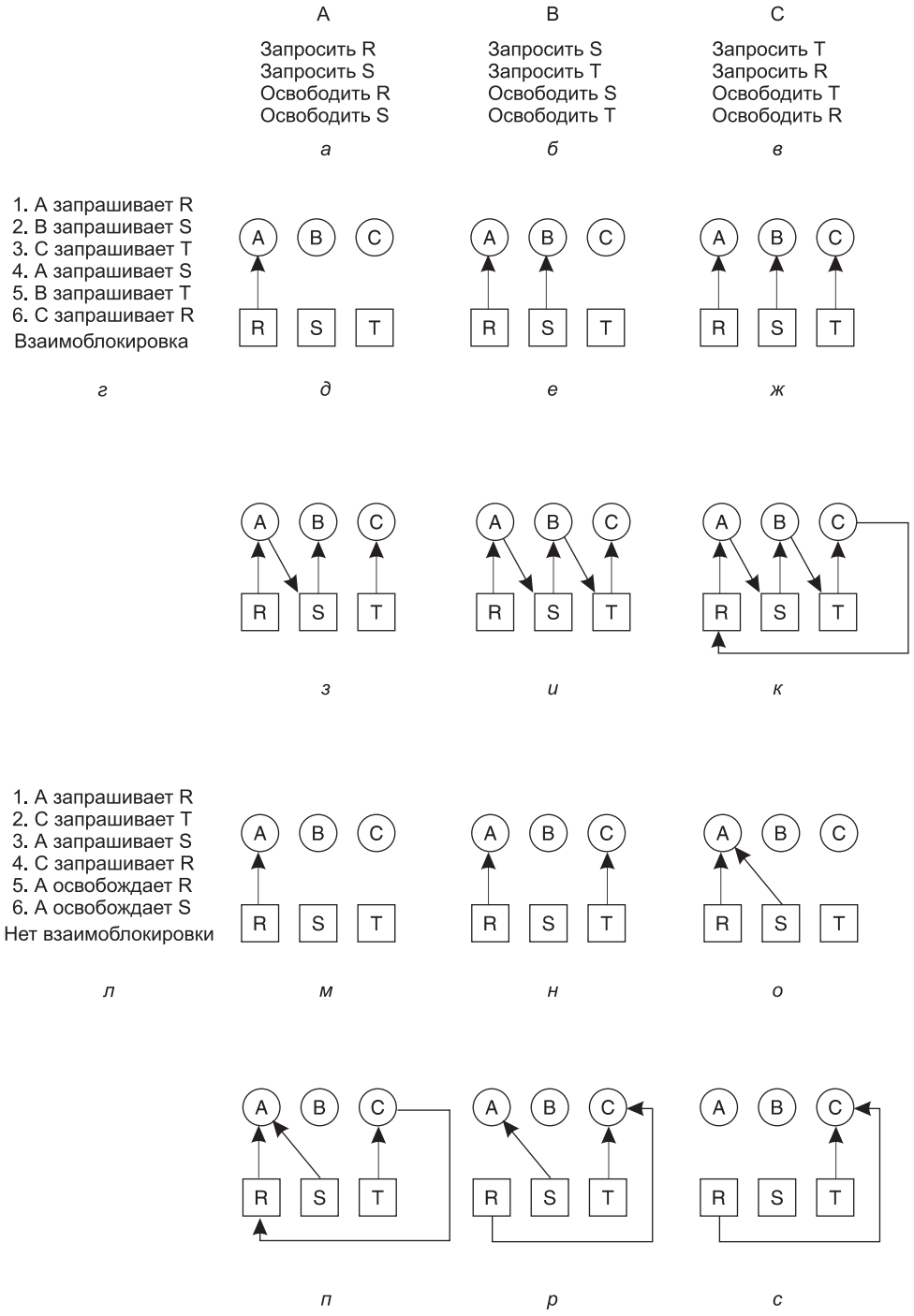

Рис. 6.2. Пример возникновения и предупреждения взаимоблокировки

показана на рис. 6.2, *л*, а не ту, которая показана на рис. 6.2, *г*. Эта последовательность отображается ресурсным графом, показанным на рис. 6.2, *м*–*с*, и не приводит к взаимоблокировке.

По окончании этапа, показанного на рис. 6.2, *с*, процессу *B* может быть выделен ресурс *S*, поскольку процесс *A* завершил свою работу, а процесс *C* имеет все необходимое. Даже если *B* заблокируется при запросе ресурса *T*, взаимоблокировки не возникнет. Процесс *B* просто будет ждать, пока процесс *C* не завершит свою работу.

Далее в этой главе будет подробно рассмотрен алгоритм принятия решений по распределению ресурсов, которые не приводят к взаимоблокировкам. А сейчас нужно понять, что ресурсные графы являются инструментом, позволяющим понять, приводит ли данная последовательность запросов/возвратов ресурсов к взаимоблокировке. Мы всего лишь шаг за шагом осуществляем запросы и возвраты ресурсов и после каждого шага проверяем граф на наличие каких-либо циклов. Если образуется цикл, значит, возникла взаимоблокировка, а если нет, значит, нет и взаимоблокировки. Хотя здесь рассматривался граф ресурсов, составленный для случая использования по одному ресурсу каждого типа, ресурсные графы могут быть применены также для обработки нескольких ресурсов одного и того же типа (Holt, 1972).

Чаще всего для борьбы с взаимными блокировками используются четыре стратегии:

1. Игнорирование проблемы. Может быть, если вы проигнорируете ее, она проигнорирует вас.
2. Обнаружение и восстановление. Дайте взаимоблокировкам проявить себя, обнаружьте их и выполните необходимые действия.
3. Динамическое уклонение от них за счет тщательного распределения ресурсов.
4. Предотвращение за счет структурного подавления одного из четырех условий, необходимых для их возникновения.

В следующих четырех разделах будут рассмотрены по порядку каждый из этих методов.

6.3. Страусиный алгоритм

Самым простым подходом к решению проблемы является «страусиный алгоритм»: спрячьте голову в песок и сделайте вид, что проблема отсутствует[1]. Люди реагируют на эту стратегию по-разному. Математики считают ее неприемлемой и говорят, что взаимоблокировки следует предотвращать любой ценой. Инженеры спрашивают, как часто ожидается возникновение проблемы, как часто система дает сбой по другим причинам и насколько серьезны последствия взаимоблокировки. Если взаимоблокировка возникает в среднем один раз в пять лет, а система раз в неделю сбоит из-за технических отказов и дефектов операционной системы, большинство инженеров не захотят платить за избавление от взаимоблокировок существенным снижением производительности или удобства использования.

---

[1] Вообще-то эта байка не соответствует действительности. Страус может развивать скорость 60 км/ч, а силы его удара ногой достаточно, чтобы убить любого льва, решившего устроить себе шикарный обед из страуса, и львы об этом знают. — *Примеч. ред.*

Чтобы усилить контраст между этими двумя позициями, рассмотрим операционную систему, которая блокирует вызывающий процесс, когда системный вызов *open*, относящийся к такому физическому устройству, как привод Blu-ray-диска или принтер, не может быть выполнен из-за занятости устройства. Обычно именно драйвер устройства решает, какое действие и при каких обстоятельствах предпринять. Две вполне очевидные возможности — это блокировка или возвращение кода ошибки. Если одному процессу удастся «открыть» привод Blu-ray-дисков, а другому посчастливится «открыть» принтер, а затем каждый процесс попытается «открыть» еще и другой ресурс и его попытка будет заблокирована, возникнет взаимоблокировка. Лишь немногие современные системы в состоянии обнаружить подобную ситуацию.

6.4. Обнаружение взаимоблокировок и восстановление работоспособности

Вторая по счету — технология обнаружения и восстановления. При ее использовании система не пытается предотвращать взаимоблокировки. Она позволяет им произойти, пытается обнаружить момент их возникновения, а затем предпринимает некоторые действия по восстановлению работоспособности. В этом разделе будут рассмотрены некоторые способы обнаружения взаимоблокировок и некоторые методы восстановления работоспособности, которые можно реализовать.

6.4.1. Обнаружение взаимоблокировки при использовании одного ресурса каждого типа

Начнем с самого простого случая, когда используется только один ресурс каждого типа. У системы может быть один сканер, один пишущий привод Blu-ray-дисков, один плоттер и один накопитель на магнитной ленте, но только по одному экземпляру каждого класса ресурсов. Иными словами, мы временно исключаем системы, имеющие два принтера. Они будут рассмотрены позже с использованием другого метода.

Для такой системы можно построить ресурсный граф (рис. 6.1). Если этот граф содержит один и более циклов, значит, мы имеем дело с взаимоблокировкой. Любой процесс, являющийся частью цикла, заблокирован намертво. Если циклов нет, значит, система не находится в состоянии взаимоблокировки.

В качестве примера более сложной системы по сравнению с ранее рассмотренными, возьмем систему с семью процессами от *A* до *G* и шестью ресурсами от *R* до *W*. Каждый ресурс находится в состоянии текущей занятости, и на каждый ресурс в данный момент поступил запрос:

1. Процесс *A* удерживает *R* и хочет получить *S*.
2. Процесс *B* не удерживает никаких ресурсов, но хочет получить *T*.
3. Процесс *C* не удерживает никаких ресурсов, но хочет получить *S*.
4. Процесс *D* удерживает *U* и хочет получить *S* и *T*.
5. Процесс *E* удерживает *T* и хочет получить *V*.
6. Процесс *F* удерживает *W* и хочет получить *S*.
7. Процесс *G* удерживает *V* и хочет получить *U*.