Взаимоблокировки процессов

Общая информация

В компьютерных системах множество ресурсов, которые одновременно могут использоваться только одним процессом.

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 ОС способны временно предоставлять процессу исключительные права доступа к конкретным ресурсам

Пример

Например каждый из двух процессов захотел записать отсканированный документ на Blu-ray-диск. Процесс А запрашивает разрешение на использование сканера и получает его. Процесс В запрограммирован по-другому: сначала он запрашивает разрешение на использование пишущего привода Blu-ray-дисков и также получает это разрешение. Теперь А запрашивает разрешение на использование пишущего привода Blu-ray-дисков, но запрос отклоняется до тех пор, пока это устройство не будет освобождено процессом В. К сожалению, вместо того чтобы освободить привод, В запрашивает разрешение на использование сканера. И в этот момент оба процесса оказываются заблокированными навсегда

Определение

Взаимная блокировка — ситуация в многозадачной операционной системе, при которой несколько процессов находятся в состоянии ожидания ресурсов, занятых друг другом, и ни один из них не может продолжать свое выполнение

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.

Ресурсы

Основная часть взаимоблокировок связана с ресурсами, к которым некоторым процессам были предоставлены исключительные права доступа.

Ресурсами могут быть аппаратные устройства (например, привод Blu-ray-дисков) или какая-то часть информации (например, запись базы данных).

Под ресурсом понимается все, что должно предоставляться, использоваться и через некоторое время

высвобождаться

Выгружаемые ресурсы

К выгружаемым относятся такие ресурсы, которые могут быть безболезненно отобраны у процесса, который ими обладает. Примером такого ресурса может послужить память.

Пример

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.

Пример

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

Невыгружаемые ресурсы

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.

Как правило, во взаимоблокировках фигурируют невыгружаемые ресурсы.

Использование ресурсов

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

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

Если ресурс недоступен

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

Если ресурс недоступен

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.

Семафоры для защиты

Использование семафоров для защиты одного ресурса

```
typedef int semaphore;
semaphore resource_1;


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

Семафоры для защиты

Использование семафоров для защиты двух ресурсов

```
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);
}
```

Семафоры для защиты

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.

Семафоры для защиты

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.

Семафоры для защиты

оба процесса запрашивают ресурсы в одном и том же порядке;

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

Семафоры для защиты

оба процесса запрашивают ресурсы в разном порядке; ВОЗМОЖНА ВЗАИМОБЛОКИРОВКА

```
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);
    }
```

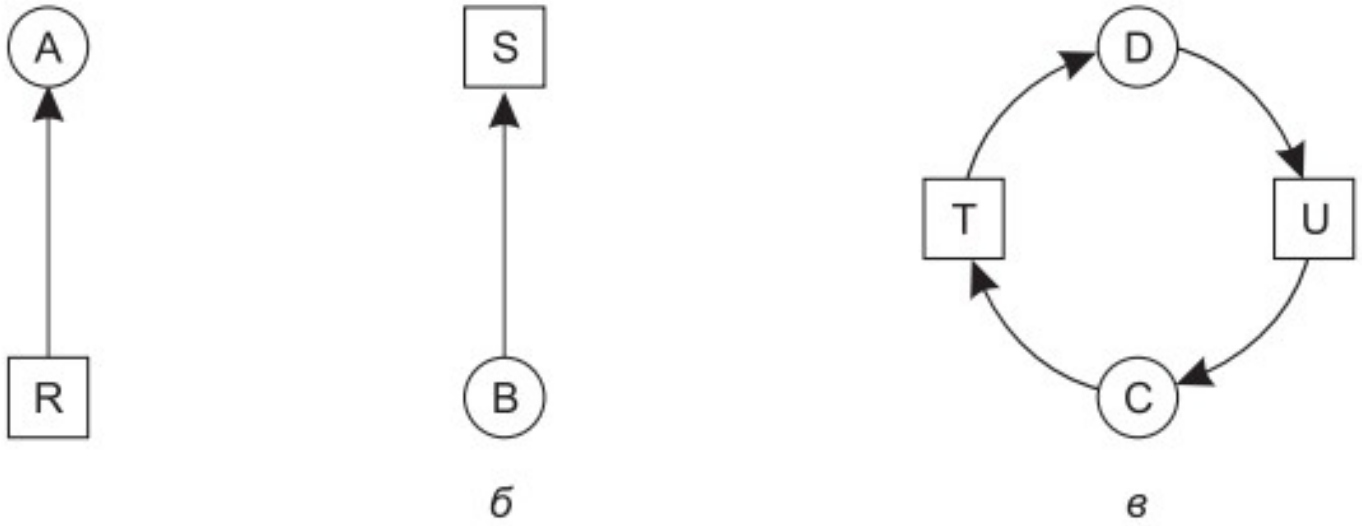
```
void process_B(void){
    down(&resource_2);
    down(&resource_1);
    use_both_resources( );
        up(&resource_1);
        up(&resource_2);
    }
```

Условия возникновения взаимоблокировок

1. Условие взаимного исключения
2. Условие удержания и ожидания
3. Условие невыгружаемости
4. Условие циклического ожидания

Для возникновения ресурсной взаимоблокировки должны соблюдаться все четыре
условия

Моделирование взаимоблокировок



б *в*

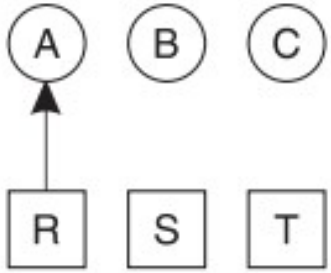
Рис. 6.1. Графы распределения ресурсов: *а* — ресурс занят; *б* — запрос ресурса; *в* — взаимоблокировка

Ситуация

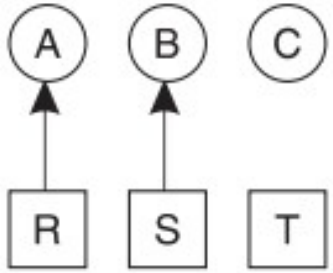
Представим, что есть три процесса, A, B и C, и три ресурса, R, S и T. Последовательности действий по запросу и высвобождению ресурсов, осуществляемые этими тремя процессами, показаны на след слайде. Операционная система может в любое время запустить любой незаблокированный процесс, то есть она может 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.

Ситуация

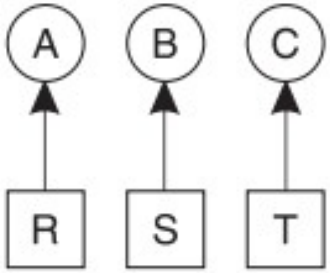
Такой порядок не приводит к взаимоблокировкам (поскольку отсутствует борьба за овладение ресурсами), но в нем нет и никакой параллельной работы. Кроме действий по запросу и высвобождению ресурсов процессы занимаются еще и вычислениями, и вводом-выводом данных. Когда процессы запускаются последовательно, отсутствует возможность использования центрального процессора одним процессом, в то время когда другой процесс ожидает завершения операции ввода-вывода



д

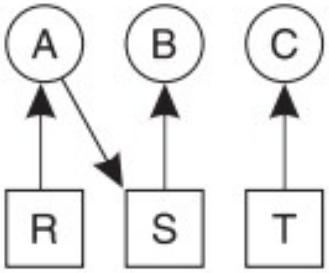


е

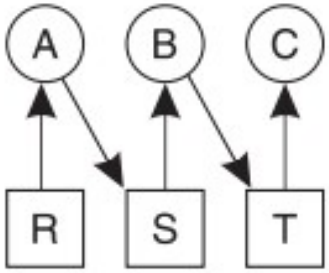


ж

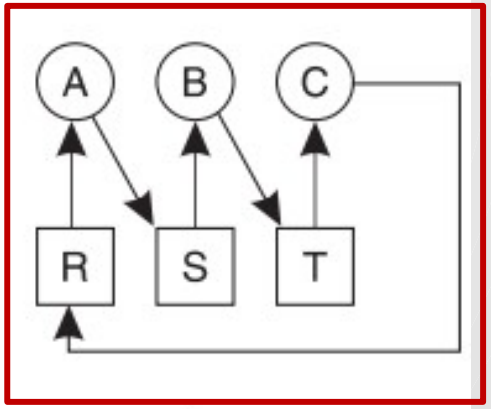
ВЗАИМОБЛОКИРОВКА



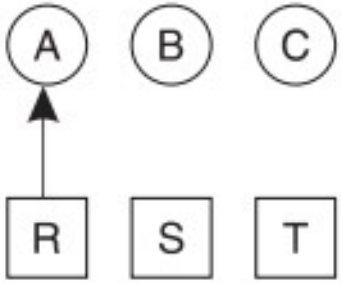
з



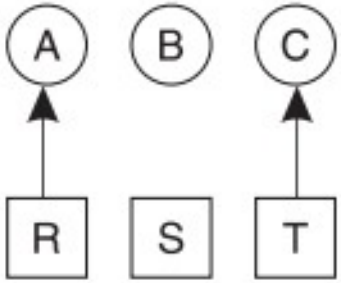
и



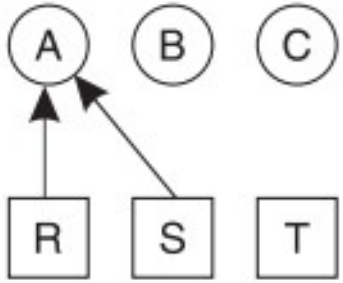
к



м

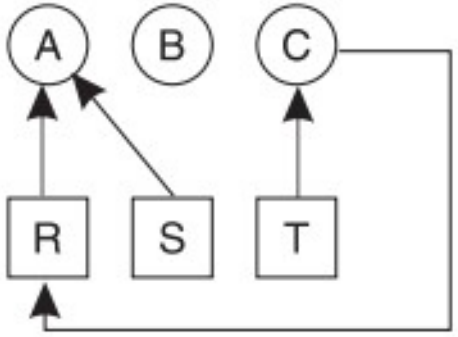


н

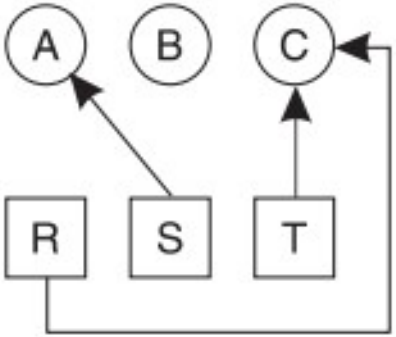


о

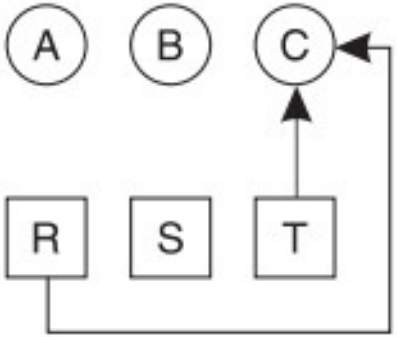
ВЗАИМОБЛОКИРОВКА была предусмотрена



п



р



с

Пример 2

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

Пример 2

Возникает следующий вопрос: «Находится ли эта система в состоянии взаимоблокировки, и если находится, то какие процессы вовлечены в это состояние?»

Чтобы ответить на этот вопрос, можно построить граф ресурсов, показанный на след. слайде

Пример 2

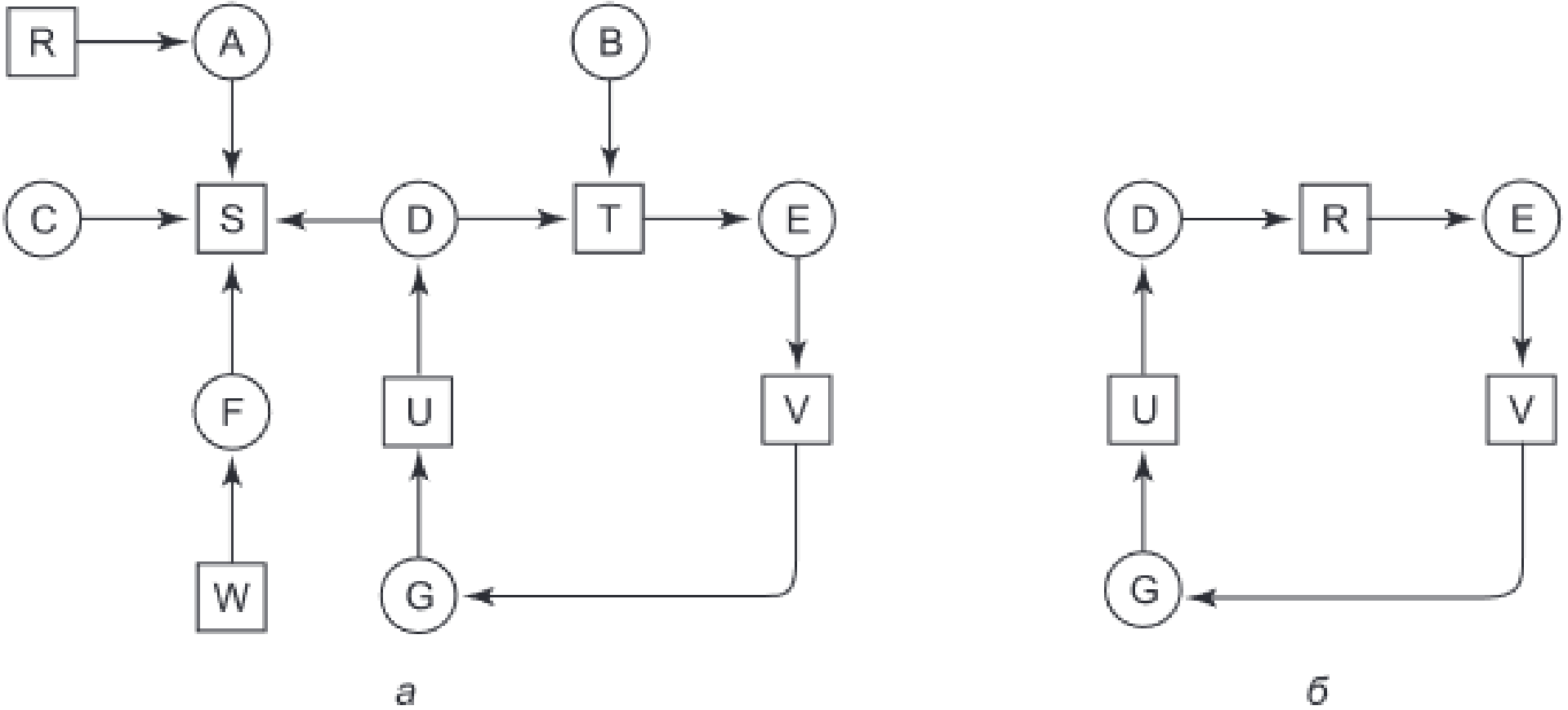


Рис. 6.3. Граф ресурсов (*а*); извлеченный из него цикл (*б*)

Пример 2

Этот граф содержит один цикл, который можно обнаружить визуально.

Этот цикл показан на рис. б. Из цикла видно, что процессы D, E и G вовлечены во взаимоблокировку. Процессы A, C и F не находятся в состоянии взаимоблокировки, поскольку ресурс S может быть выделен любому из них, который затем закончит свою работу и вернет ресурс. Затем оставшиеся два процесса смогут взять его по очереди и также завершить свою работу.

Алгоритм поиска взаимоблокировок

Известно множество алгоритмов для обнаружения циклов в направленных графах. Далее будет приведен простой алгоритм, проверяющий граф и прекращающий свою работу либо при обнаружении цикла, либо при обнаружении отсутствия циклов. В нем используется одна динамическая структура данных, L, представляющая собой список узлов, а также список ребер. В процессе работы алгоритма ребра будут помечаться для обозначения того, что они уже были проверены, чтобы предотвратить повторные проверки.

Алгоритм поиска взаимоблокировок

1. Для каждого узла N, имеющегося в графе, выполняются следующие пять шагов, использующих узел N в качестве начального.

2. Инициализируется (очищается) список L, а со всех ребер снимаются пометки.

3. Текущий узел добавляется к концу списка L, и проводится проверка, не появится ли этот узел в списке L дважды. Если это произойдет, значит, граф содержит цикл (отображенный в списке L) и алгоритм прекращает работу.

Алгоритм поиска взаимоблокировок

4. Для заданного узла определяется, нет ли каких-нибудь отходящих от него непомеченных ребер. Если такие ребра есть, осуществляется переход к шагу 5, если их нет, осуществляется переход к шагу 6.

5. Произвольно выбирается и помечается непомеченное отходящее от узла ребро. Затем по нему осуществляется переход к новому текущему узлу, и алгоритм возвращается к шагу 3.

Алгоритм поиска взаимоблокировок

6. Если этот узел является первоначальным узлом, значит, граф не содержит никаких циклов, и алгоритм завершает свою работу. В противном случае алгоритм зашел в тупик. Этот узел удаляется, и алгоритм возвращается к предыдущему узлу, то есть к тому узлу, который был текущим перед только что удаленным узлом данный узел делается текущим и осуществляется переход к шагу 3

Алгоритм поиска взаимоблокировок

Этот алгоритм берет поочередно каждый узел в качестве корневого, в надежде, что из этого получится дерево, и выполняет в дереве поиск в глубину. Если в процессе обхода алгоритм возвращается к уже встречавшемуся узлу, значит, он нашел цикл. Если алгоритм обходит все ребра из какого-нибудь заданного узла, то он возвращается к предыдущему узлу. Если он возвращается к корневому узлу и не может идти дальше, то подграф, доступный из текущего узла, не содержит циклов. Если данное свойство сохраняется для всех узлов, значит, полный граф не содержитциклов, а система не находится в состоянии взаимоблокировки.

Пример поиска

Начинаем с узла R и инициализируем L как пустой список. Затем добавляем узел

R в список, переходим к единственно возможному узлу A и добавляем его также

к списку L, получая L = [R, A]. Из узла A следуем к узлу S, получая L = [R, A, S]. Узел S не имеет отходящих от него ребер, следовательно, это тупик, который заставляет нас вернуться к узлу A. Так как у узла A также нет немаркированных отходящих от него ребер, мы возвращаемся к узлу R, завершая, таким образом, его исследование.

Пример поиска

Начнем снова с узла B. Из узла B проследуем по отходящим ребрам до тех пор, пока не доберемся до узла D; к этому моменту список будет иметь следующий вид: L = [B, T, E, V, G, U, D]. Теперь нужно сделать произвольный выбор. Если выбрать узел S, мы попадаем в тупик и возвращаемся к узлу D. Во второй раз выбираем узел T и обновляем список L до вида [B, T, E, V, G, U, D, T], где обнаруживаем цикл и останавливаем работу алгоритма.

Выход из взаимоблокировки

1. Восстановление за счет приоритетного овладения ресурсом

Иногда можно временно отобрать ресурс у его текущего владельца и передать его другому процессу. В большинстве случаев для этого может понадобиться вмешательство оператора

Выход из взаимоблокировки

2. Восстановление путем отката

Если разработчики системы и операторы вычислительной машины знают о том, что есть вероятность возникновения взаимоблокировки, они могут организовать периодическое создание процессами контрольных точек.

Это означает, что состояние процесса записывается в файл, что позволит осуществить его последующий перезапуск.

Выход из взаимоблокировки

3. Восстановление путем уничтожения процессов

Самым грубым, но и самым простым способом прервать взаимоблокировку является уничтожение одного или нескольких процессов. Можно уничтожить процесс, находящийся в цикле взаимоблокировки. Если повезет, то другие процессы смогут продолжить свою работу. Если это не поможет, то все можно повторить, пока цикл не будет разорван

Предотвращение взаимоблокировки

1. Атака условия взаимного исключения
2. Атака условия удержания и ожидания
3. Атака условия невыгружаемости
4. Атака условия циклического ожидания

Борьба

Чаще всего для борьбы с взаимными блокировками используются четыре стратегии:

1. Игнорирование проблемы
2. Обнаружение и восстановление
3. Динамическое уклонение от них за счет тщательного распределения ресурсов.
4. Предотвращение за счет структурного подавления одного из четырех условий, необходимых для их возникновения.