5.1 Взаимоблокировка процессов

Взаимоблокировка процессов может происходить, когда несколько процессов борются за один ресурс.

Ресурсы бывают выгружаемые и невыгружаемые, аппаратные и программные.

Выгружаемый ресурс - этот ресурс безболезненно можно забрать у процесса (например: память).

Невыгружаемый ресурс - этот ресурс нельзя забрать у процесса без потери данных (например: принтер).

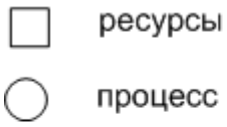
Проблема взаимоблокировок процессов возникает при борьбе за невыгружаемый ресурсы.

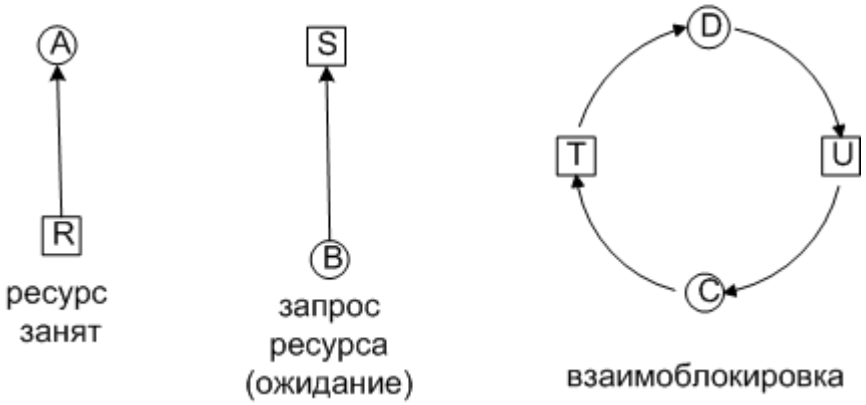
Условия необходимые для взаимоблокировки:

1. Условие взаимного исключения - в какой-то момент времени, ресурс занят только одним процессом или свободен.
2. Условие удержания и ожидания - процесс, удерживающий ресурс может запрашивать новые ресурсы.
3. Условие отсутствия принудительной выгрузки ресурса.
4. Условие циклического ожидания - должна существовать круговая последовательность из процессов, каждый, из которого ждет доступа к ресурсу, удерживаемому следующим членом последовательности.

5.2 Моделирование взаимоблокировок

Моделирование тупиков с помощью графов.





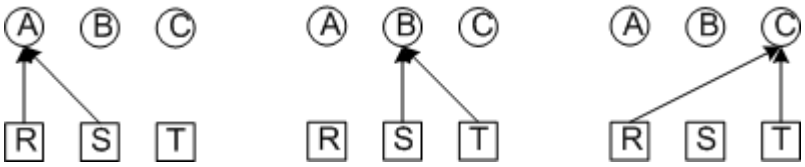
Условные обозначения

На такой модели очень хорошо проверить возникает ли взаимоблокировка. Если есть цикл, значит, есть и взаимоблокировка.

Рассмотрим простой пример:

три процесса A, B, C

три ресурса R, S, T

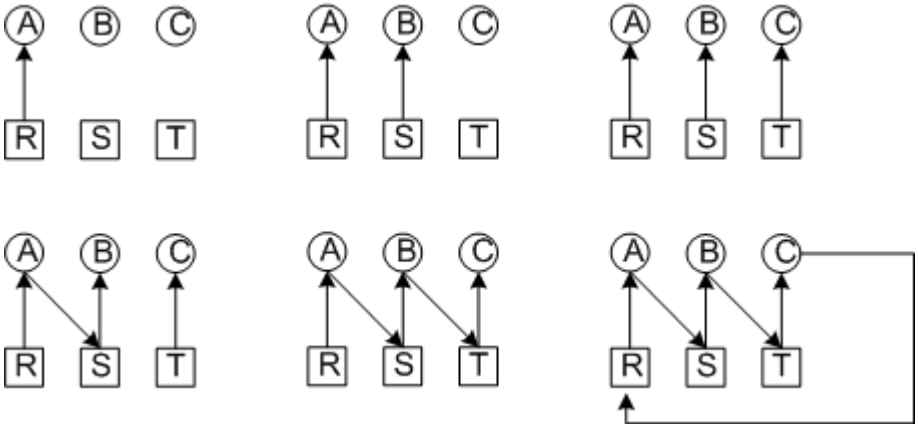


Последовательное выполнение процессов, взаимоблокировка не возникает

Рассмотрим циклический алгоритм:

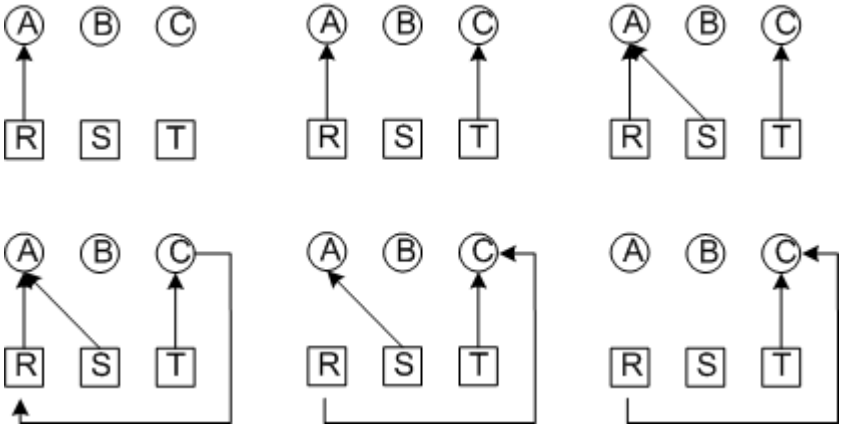
три процесса A, B, C

три ресурса R, S, T



Возникает взаимоблокировка

Рассмотрим тот же самый случай, но допустим, что система, зная о предстоящей взаимоблокировке, заблокирует процесс **B.**



Взаимоблокировка не возникает.

5.3 Методы борьбы с взаимоблокировками

Четыре стратегии избегания взаимоблокировок:

1. Пренебрежением проблемой в целом (вдруг пронесет).
2. Обнаружение и устранение (взаимоблокировка происходит, но оперативно ликвидируется).
3. Динамическое избежание тупиков.
4. Предотвращение четырех условий, необходимых для взаимоблокировок.

5.3.1 Пренебрежением проблемой в целом (страусовый алгоритм)

Если вероятность взаимоблокировки очень мала, то ею легче пренебречь, т.к. код исключения может очень усложнить ОС и привести к большим ошибкам. Также многие взаимоблокировки тяжело обнаружить.

Этот алгоритм используется как в UNIX, так и в Windows.

Поэтому (и не только) на серверах часто устанавливают автоматическую перезагрузку (раз в сутки, как правило ночью), если возникнет взаимоблокировка, то после перезагрузки ее не будет.

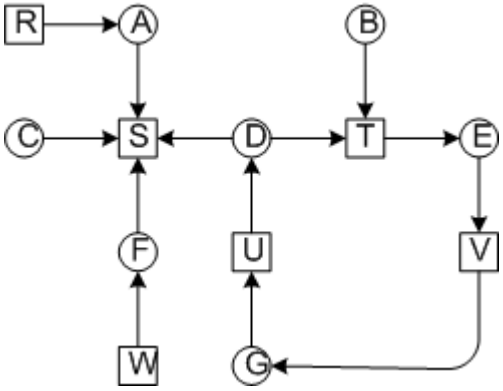
5.3.2 Обнаружение и устранение взаимоблокировок

Система не пытается предотвратить взаимоблокировку, а пытается обнаружить ее и устранить.

Обнаружение взаимоблокировки при наличии одного ресурса каждого типа

Под одним ресурсом каждого типа, подразумевается один принтер, один сканер и один плоттер и т.д.

Рассмотрим систему из 7-ми процессов и 6-ти ресурсов.



Обнаружение взаимоблокировки при наличии одного ресурса каждого типа

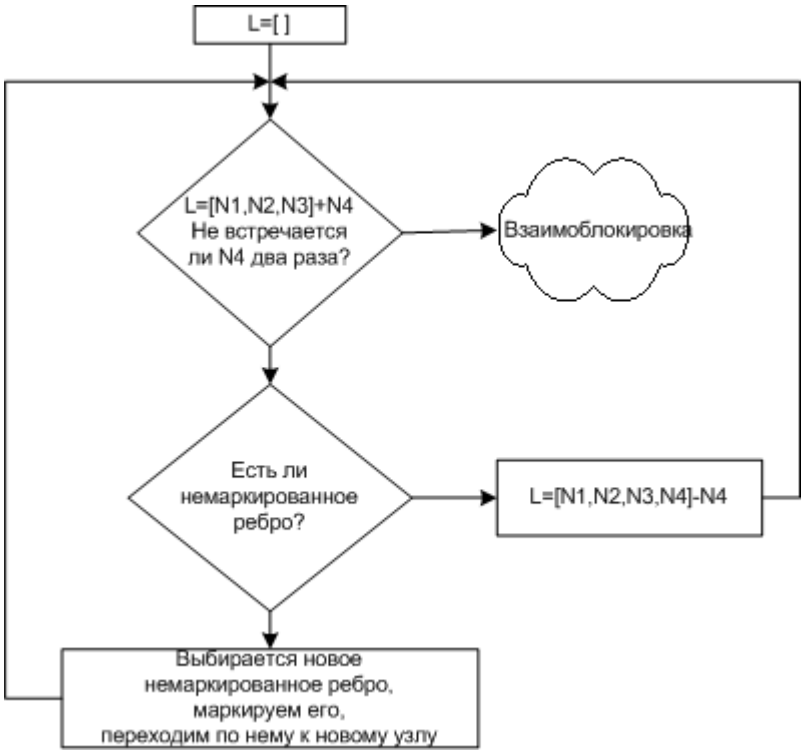
Визуально хорошо видна взаимоблокировка, но нам нужно чтобы ОС сама определяла взаимоблокировку.

Для этого нужен алгоритм.

Рассмотрим один из алгоритмов.

Для каждого узла N в графе выполняется пять шагов.

1. Задаются начальные условия: L-пустой список, все ребра не маркированы.
2. Текущий узел добавляем вконец списка L и проверяем количество появления узла в списке. Если он встречается два раза, значит цикл и взаимоблокировка.
3. Для заданного узла смотрим, выходит ли из него хотя бы одно немаркированное ребро. Если да, то переходим к шагу 4, если нет, то переходим к шагу 5.
4. Выбираем новое немаркированное исходящее ребро и маркируем его. И переходим по нему к новому узлу и возвращаемся к шагу 3.
5. Зашли в тупик. Удаляем последний узел из списка и возвращаемся к предыдущему узлу. Возвращаемся к шагу 3. Если это первоначальный узел, значит, циклов нет, и алгоритм завершается.



Алгоритм обнаружения взаимоблокировок

Для нашего случая тупик обнаруживается в списке L=[B,T,E,V,G,U,D,T]

Обнаружение взаимоблокировки при наличии нескольких ресурсов каждого типа

Рассмотрим систему.

m - число классов ресурсов (например: принтеры это один класс)

n - количество процессов

P(n) - процессы

E - вектор существующих ресурсов

E(i) - количество ресурсов класса i

A - вектор доступных (свободных) ресурсов

A(i) - количество доступных ресурсов класса i

C - матрица текущего распределения (какому процессу, какие ресурсы принадлежат)

R - матрица запросов (какой процесс, какой ресурс запросил)

Существующие ресурсы

$$(E_1 \quad E_2 \quad E_3 \quad \dots \quad E_m)$$

Доступные ресурсы

$$(A_1 \quad A_2 \quad A_3 \quad \dots \quad A_m)$$

Матрица текущего распределения

$$\begin{bmatrix} C_{11} & C_{12} & \dots & C_{1m} \\ C_{21} & C_{22} & \dots & C_{2m} \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ C_{n1} & C_{n2} & \dots & C_{nm} \end{bmatrix}$$

Строка **n** предоставлена процессу **n**

Матрица запроса

$$\begin{bmatrix} R_{11} & R_{12} & \dots & R_{1m} \\ R_{21} & R_{22} & \dots & R_{2m} \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ R_{n1} & R_{n2} & \dots & R_{nm} \end{bmatrix}$$

Строка **n** нужна процессу **n**

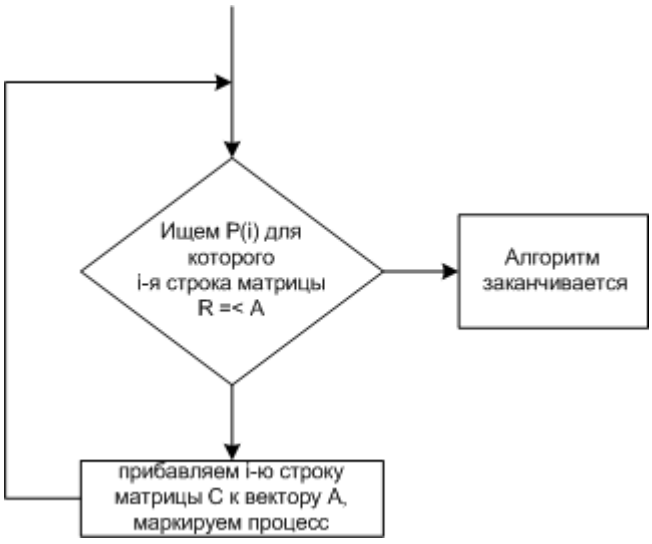
C(ij) - количество экземпляров ресурса j, которое занимает процесс P(i).

R(ij) - количество экземпляров ресурса j, которое хочет получить процесс P(i).

$$\sum_{i=1}^{n} C_{ij} + A_j = E_j$$

Общее количество ресурсов равно сумме занятых и свободных ресурсов

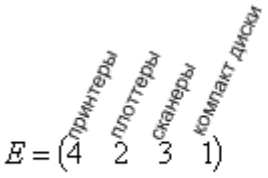
Рассмотрим алгоритм поиска тупиков.

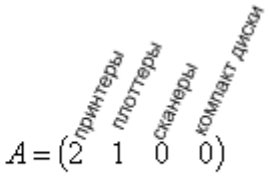


Алгоритм поиска тупиков при наличии нескольких ресурсов каждого типа

Если остаются не маркированные процессы, значит, есть тупик.

Рассмотрим работу алгоритма на реальном примере.





Матрица текущего распределения

$$C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \end{bmatrix}$$

Матрица запросов

$$R = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 \end{bmatrix}$$

Используем алгоритм:

1. Третий процесс может получить желаемые ресурсы, т.к. R (2 1 0 0) = A (2 1 0 0)
2. Третий процесс освобождает ресурсы. Прибавляем их к **A**. A = (2 1 0 0) + (0 1 2 0) =(2 2 2 0). Маркируем процесс.
3. Может выполняться процесс **2**. По окончании A=(4 2 2 1).
4. Теперь может работать первый процесс.

Тупиков не обнаружено.

Если рассмотреть пример, когда второму процессу требуются ресурсы (1 0 3 0), то два процесса окажутся в тупике.

Когда можно искать тупики:

- Когда запрашивается очередной ресурс (очень загружает систему)
- Через какой то промежуток времени (в интерактивных системах пользователь это ощутит)
- Когда загрузка процессора слишком велика

Выход из взаимоблокировки

Восстановление при помощи принудительной выгрузки ресурса

Как правило, требует ручного вмешательства (например: принтер).

Восстановление через откат

Состояние процессов записывается в контрольных точках, и в случае тупика можно сделать откат процесса на более раннее состояние, после чего он продолжит работу снова с этой точки.

С принтером опять будут проблемы.

Восстановление путем уничтожения процесса

Самый простой способ.

Но с принтером опять будут проблемы.

В реальных системах они не годятся.

5.3.3 Динамическое избежание взаимоблокировок

В этом способе ОС должна знать, является ли предоставление ресурса безопасным или нет.

Траектории ресурсов

Рассмотрим модель из двух процессов и двух ресурсов.

A1 - запрос принтера процессом A

A2 - запрос плоттера процессом A

A3 - освобождение принтера процессом A

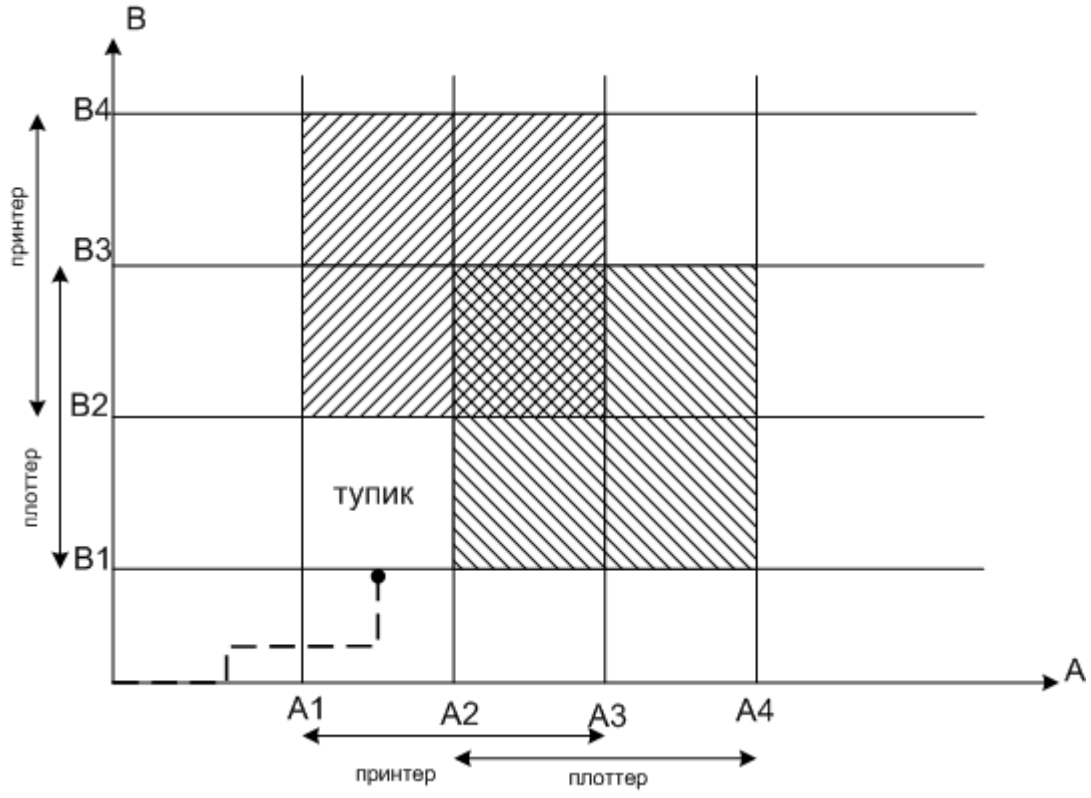
A4 - освобождение плоттера процессом A

B1 - запрос плоттера процессом B

B2 - запрос принтера процессом B

B3 - освобождение плоттера процессом B

B4 - освобождение принтера процессом B



Динамическое избежание взаимоблокировок

Т.к. процессор предоставляется поочередно, траектория может продолжаться только параллельно осям.

Чтобы избежать тупика, процессам надо обойти прямоугольник, охватывающий всю заштрихованную область.

Безопасные и небезопасные состояния

В безопасном состоянии система может гарантировать, что все процессы закончат свою работу.

Рассмотрим систему.

10 экземпляров ресурса

3 процесса

	имеет	max
A	3	9
B	2	4
C	2	7

Свободно:3

	имеет	max
A	3	9
B	4	4
C	2	7

Свободно:1

	имеет	max
A	3	9
B	0	-
C	2	7

Свободно:5

	имеет	max
A	3	9
B	0	-
C	5	7

Свободно:0

	имеет	max
A	3	9
B	0	-
C	0	-

Свободно:7

Процесс A занял 3 экземпляра, но ему необходимо 9.

В этой ситуации можно спланировать так, сначала запустить процесс B, потом C и потом A.

Процессы заканчивают работу без тупиковой ситуации.

Рассмотрим другую ситуацию.

Процесс A занял 4 экземпляра.

	имеет	max
A	3	9
B	2	4
C	2	7

Свободно:3

	имеет	max
A	4	9
B	2	4
C	2	7

Свободно:2

	имеет	max
A	4	9
B	4	4
C	2	7

Свободно:0

	имеет	max
A	4	9
B	0	-
C	2	7

Свободно:4

Возникает небезопасное состояние.

В принципе, процесс A может в какой то момент ресурс освободить и тупика не возникнет.

Видно, что в этом случае не стоило давать ресурс процессу A.

Алгоритм банкира для одного вида ресурсов

"Банкира", потому что аналогия такая, клиенты-процессы, кредиты-ресурсы.

Рассмотрим систему:

Банкир может дать 10 кредитов (ресурсы).

К нему попеременно обращаются 4-ре клиента.

	имеет	max
A	0	6
B	0	5
C	0	4
D	0	7

Свободно:10

	имеет	max
A	1	6
B	1	5
C	2	4
D	4	7

Свободно:2

	имеет	max
A	1	6
B	2	5
C	2	4
D	4	7

Свободно:1

Алгоритм банкира:

1. Банкиру поступает запрос от клиента на получение кредита
2. Банкир проверяет, приводит ли этот запрос к небезопасному состоянию.
3. Банкир в зависимости от этого дает или отказывает в кредите.

Алгоритм банкира

Алгоритм банкира для несколько видов ресурсов

Рассмотрим систему:

C

	принтеры	плоттеры	сканеры	устройства компакт-дисков
A	3	0	1	1
B	0	1	0	0
C	1	1	1	0
D	1	1	0	1
E	0	0	0	0

Распределенные ресурсы

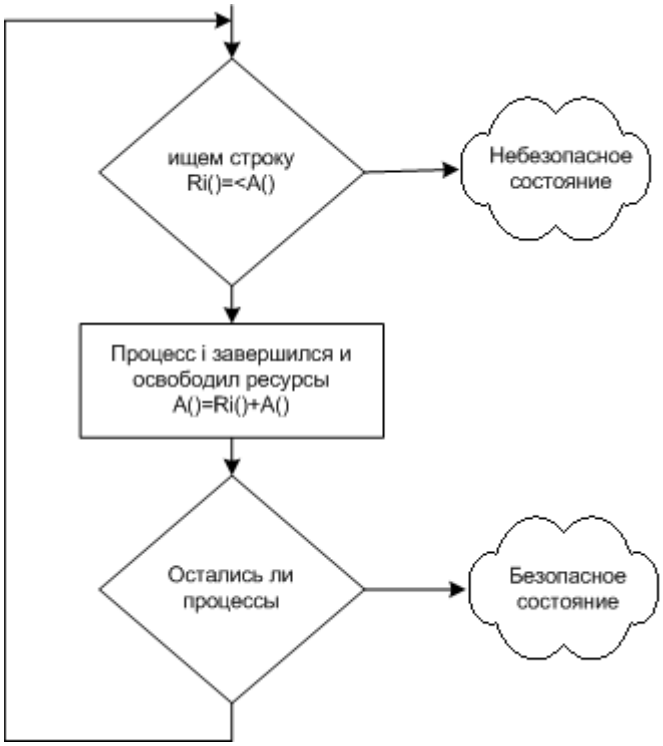
R

	принтеры	плоттеры	сканеры	устройства компакт-дисков
A	1	1	0	0
B	0	1	1	2
C	3	1	0	0
D	0	0	1	0
E	2	1	1	0

Ресурсы, которые еще нужны

вектора:
E=(6342) - существующие ресурсы
P=(5322) - занятые ресурсы
A=(1020) - доступные ресурсы

Алгоритм поиска безопасного или небезопасного состояния:



Алгоритм банкира для несколько видов ресурсов

Если состояние безопасное то ресурс дать можно, если нет то нельзя.

На практике все эти алгоритмы тяжело реализовать.

5.3.4 Предотвращение четырех условий, необходимых для взаимоблокировок

Предотвращение условия взаимного исключения

Можно минимизировать количество процессов борющихся за ресурсы.

Например, с помощью спулинга для принтера, когда только демон принтера работает с принтером.

Предотвращение условия удержания и ожидания

Один из способов достижения этой цели, это когда процесс должен запрашивать все необходимые ресурсы до начала работы. Если хоть один ресурс недоступен, то процессу вообще ничего не предоставляется.

Предотвращение условия отсутствия принудительной выгрузки ресурса

Можно выгружать ресурсы, но могут быть проблемы с принтером.

Предотвращение условия циклического ожидания

Способы предотвращения:

- Процесс сначала должен освободить занятый ресурс, прежде чем занять новый.
- Можно пронумеровать все ресурсы (и упорядочить), и процессы должны запрашивать ресурсы только по возрастающему порядку.