**09.12.2020 ИС2К Операционные системы**

**«Основы операционных систем» К.А. Коньков** <https://yadi.sk/i/SojomTcdoQ2v_g>

**Тема:** Лекция 7: Тупики

(Учебник «Основы операционных систем» К.А.Коньков стр. 64-71)

**Ответить на вопросы: (кратко)**

1. **Что такое тупик (deadlock)**
2. **Проблема тупиков: эффективное решение**
3. **Причины тупиков (deadlock)**
4. **Основные направления борьбы с тупиками**

**Ответы присылать** [kuzn117@yandex.ru](mailto:kuzn117@yandex.ru) **можно в ВК -**  **id480169637**

**Пишите тему письма и файл именуйте: Фамилия-ОС-0912**

**Для тех у кого проблемы с Яндекс диском сегодняшнюю тему**

**(**Лекция 7:

Тупики)

**выкладываю ниже**

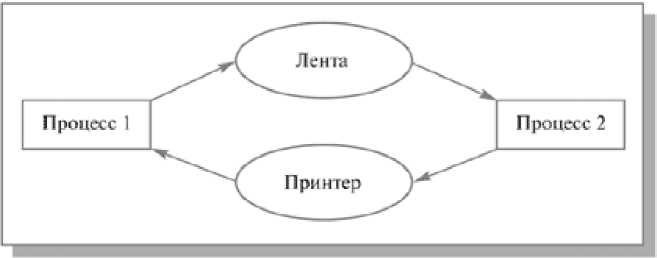
**7. Лекция: Тупики**

В лекции рассматриваются вопросы взаимоблокировок, тупиковых ситуаций и "зависаний" системы

**Введение**

В предыдущих лекциях мы рассматривали способы синхронизации процессов, которые позволяют про-  
цессам успешно кооперироваться. Однако в некоторых случаях могут возникнуть непредвиденные за-  
труднения. Предположим, что несколько процессов конкурируют за обладание конечным числом ресур-  
сов. Если запрашиваемый процессом ресурс недоступен, ОС переводит данный процесс в состояние  
ожидания. В случае когда требуемый ресурс удерживается другим ожидающим процессом, первый про-  
цесс не сможет сменить свое состояние. Такая ситуация называется тупиком (deadlock). Говорят, что в  
мультипрограммной системе процесс находится в состоянии тупика, если он ожидает события, которое  
никогда не произойдет. Системная тупиковая ситуация, или "зависание системы", является следствием  
того, что один или более процессов находятся в состоянии тупика. Иногда подобные ситуации называют  
взаимоблокировками. В общем случае проблема тупиков эффективного решения не имеет.

Рассмотрим пример. Предположим, что два процесса осуществляют вывод с ленты на принтер. Один из  
них успел монополизировать ленту и претендует на принтер, а другой наоборот. После этого оба процес-  
са оказываются заблокированными в ожидании второго ресурса (см. рис. 7.1).



**Рис. 7.1.** Пример тупиковой ситуации

**Основы операционных систем**

**65**

**Определение**. Множество процессов находится в тупиковой ситуации, если каждый процесс из множе-  
ства ожидает события, которое может вызвать только другой процесс данного множества. Так как все  
процессы чего-то ожидают, то ни один из них не сможет инициировать событие, которое разбудило бы  
другого члена множества и, следовательно, все процессы будут спать вместе.

Выше приведен пример взаимоблокировки, возникающей при работе с так называемыми выделенными  
устройствами. Тупики, однако, могут иметь место и в других ситуациях. Например, в системах управле-  
ния базами данных записи могут быть локализованы процессами, чтобы избежать состояния гонок (см.  
лекцию 5 "Алгоритмы синхронизации"). В этом случае может получиться так, что один из процессов за-  
блокировал записи, необходимые другому процессу, и наоборот. Таким образом, тупики могут иметь ме-  
сто как на аппаратных, так и на программных ресурсах.

Тупики также могут быть вызваны ошибками программирования. Например, процесс может напрасно  
ждать открытия семафора, потому что в некорректно написанном приложении эту операцию забыли пре-  
дусмотреть. Другой причиной бесконечного ожидания может быть дискриминационная политика по от-  
ношению к некоторым процессам. Однако чаще всего событие, которого ждет процесс в тупиковой си-  
туации, - освобождение ресурса, поэтому в дальнейшем будут рассмотрены методы борьбы с тупиками  
ресурсного типа.

Ресурсами могут быть как устройства, так и данные. Некоторые ресурсы допускают разделение между  
процессами, то есть являются **разделяемыми** ресурсами. Например, память, процессор, диски коллек-  
тивно используются процессами. Другие не допускают разделения, то есть являются **выделенными**, на-  
пример лентопротяжное устройство. К взаимоблокировке может привести использование как выделен-  
ных, так и разделяемых ресурсов. Например, чтение с разделяемого диска может одновременно осущест-  
вляться несколькими процессами, тогда как запись предполагает исключительный доступ к данным на  
диске. Можно считать, что часть диска, куда происходит запись, выделена конкретному процессу. По-  
этому в дальнейшем мы будем исходить из предположения, что тупики связаны с выделенными ресурса-  
ми , то есть тупики возникают, когда процессу предоставляется эксклюзивный доступ к устройствам,  
файлам и другим ресурсам.

Традиционная последовательность событий при работе с ресурсом состоит из запроса, использования и  
освобождения ресурса. Тип запроса зависит от природы ресурса и от ОС. Запрос может быть явным, на-  
пример специальный вызов request, или неявным - open для открытия файла. Обычно, если ресурс занят  
и запрос отклонен, запрашивающий процесс переходит в состояние ожидания.

Далее в данной лекции будут рассматриваться вопросы обнаружения, предотвращения, обхода тупиков и  
восстановления после тупиков. Как правило, борьба с тупиками - очень дорогостоящее мероприятие.

Тем не менее для ряда систем, например для систем реального времени, иного выхода нет.

**Условия возникновения тупиков**

Условия возникновения тупиков были сформулированы Коффманом, Элфиком и Шошани в 1970 г.

1. Условие взаимоисключения (Mutual exclusion). Одновременно использовать ресурс может только  
один процесс.

2. Условие ожидания ресурсов (Hold and wait). Процессы удерживают ресурсы, уже выделенные им,  
и могут запрашивать другие ресурсы.

3. Условие неперераспределяемости (No preemtion). Ресурс, выделенный ранее, не может быть при-  
нудительно забран у процесса. Освобождены они могут быть только процессом, который их  
удерживает.

4. Условие кругового ожидания (Circular wait). Существует кольцевая цепь процессов, в которой ка-  
ждый процесс ждет доступа к ресурсу, удерживаемому другим процессом цепи.

Для образования тупика необходимым и достаточным является выполнение **всех четырех** условий.

Обычно тупик моделируется циклом в графе, состоящем из узлов двух видов: прямоугольников - про-  
цессов и эллипсов - ресурсов, наподобие того, что изображен на рис. 7.1. Стрелки, направленные от ре-

**Основы операционных систем**

**66**

сурса к процессу, показывают, что ресурс выделен данному процессу. Стрелки, направленные от процес-  
са к ресурсу, означают, что процесс запрашивает данный ресурс.

**Основные направления борьбы с тупиками**

Проблема тупиков инициировала много интересных исследований в области информатики. Очевидно,  
что условие циклического ожидания отличается от остальных. Первые три условия формируют правила,  
существующие в системе, тогда как четвертое условие описывает ситуацию, которая может сложиться  
при определенной неблагоприятной последовательности событий. Поэтому методы предотвращения  
взаимоблокировок ориентированы главным образом на нарушение первых трех условий путем введения  
ряда ограничений на поведение процессов и способы распределения ресурсов. Методы обнаружения и  
устранения менее консервативны и сводятся к поиску и разрыву цикла ожидания ресурсов.

Итак, основные направления борьбы с тупиками:

• Игнорирование проблемы в целом

• Предотвращение тупиков

• Обнаружение тупиков

• Восстановление после тупиков

**Игнорирование проблемы тупиков**

Простейший подход - не замечать проблему тупиков. Для того чтобы принять такое решение, необходи-  
мо оценить вероятность возникновения взаимоблокировки и сравнить ее с вероятностью ущерба от дру-  
гих отказов аппаратного и программного обеспечения. Проектировщики обычно не желают жертвовать  
производительностью системы или удобством пользователей для внедрения сложных и дорогостоящих  
средств борьбы с тупиками.

Любая ОС, имеющая в ядре ряд массивов фиксированной размерности, потенциально страдает от тупи-  
ков, даже если они не обнаружены. Таблица открытых файлов, таблица процессов, фактически каждая  
таблица являются ограниченными ресурсами. Заполнение всех записей таблицы процессов может при-  
вести к тому, что очередной запрос на создание процесса может быть отклонен. При неблагоприятном  
стечении обстоятельств несколько процессов могут выдать такой запрос одновременно и оказаться в ту-  
пике. Следует ли отказываться от вызова CreateProcess, чтобы решить эту проблему?

Подход большинства популярных ОС (Unix, Windows и др.) состоит в том, чтобы игнорировать данную  
проблему в предположении, что маловероятный случайный тупик предпочтительнее, чем нелепые пра-  
вила, заставляющие пользователей ограничивать число процессов, открытых файлов и т. п. Сталкиваясь с  
нежелательным выбором между строгостью и удобством, трудно найти решение, которое устраивало бы  
всех.

**Способы предотвращения тупиков**

Цель предотвращения тупиков - обеспечить условия, исключающие возможность возникновения тупи-  
ковых ситуаций. Большинство методов связано с предотвращением одного из условий возникновения  
взаимоблокировки.

Система, предоставляя ресурс в распоряжение процесса, должна принять решение, безопасно это или  
нет. Возникает вопрос: есть ли такой алгоритм, который помогает всегда избегать тупиков и делать пра-  
вильный выбор. Ответ - да, мы можем избегать тупиков, но только если определенная информация из-  
вестна заранее.

**Способы предотвращения тупиков путем тщательного распределения ресурсов. Алгоритм банкира**

Можно избежать взаимоблокировки, если распределять ресурсы, придерживаясь определенных правил.  
Среди такого рода алгоритмов наиболее известен алгоритм банкира, предложенный Дейкстрой, который  
базируется на так называемых **безопасных** или **надежных** состояниях (safe state). Безопасное состояние -

**Основы операционных систем**

**67**

это такое состояние, для которого имеется по крайней мере одна последовательность событий, которая  
не приведет к взаимоблокировке. Модель алгоритма основана на действиях банкира, который, имея в на-  
личии капитал, выдает кредиты.

Суть алгоритма состоит в следующем.

• Предположим, что у системы в наличии n устройств, например лент.

• ОС принимает запрос от пользовательского процесса, если его максимальная потребность не пре-  
вышает n.

• Пользователь гарантирует, что если ОС в состоянии удовлетворить его запрос, то все устройства  
будут возвращены системе в течение конечного времени.

• Текущее состояние системы называется надежным, если ОС может обеспечить всем процессам  
их выполнение в течение конечного времени.

• В соответствии с алгоритмом банкира выделение устройств возможно, только если состояние сис-  
темы остается надежным.

Рассмотрим пример надежного состояния для системы с 3 пользователями и 11 устройствами, где 9 уст-  
ройств задействовано, а 2 имеется в резерве. Пусть текущая ситуация такова:

|  |  |  |
| --- | --- | --- |
| **Пользователи** | **Максимальная потребносгь в ресурсах** | **Выделенное поль зователям количесгво ресурсов** |
| **Первый** | **9** | **6** |
| **Второй** | **10** | **2** |
| **Третий** | **3** | **1** |

Рис. 7.2. Пример надежного состояния для системы с 3 пользователями и 11 устройствами.

Данное состояние надежно. Последующие действия системы могут быть таковы. Вначале удовлетворить  
запросы третьего пользователя, затем дождаться, когда он закончит работу и освободит свои три устрой-  
ства. Затем можно обслужить первого и второго пользователей. То есть система удовлетворяет только те  
запросы, которые оставляют ее в надежном состоянии, и отклоняет остальные.

Термин ненадежное состояние не предполагает, что обязательно возникнут тупики. Он лишь говорит о  
том, что в случае неблагоприятной последовательности событий система может зайти в тупик.

Данный алгоритм обладает тем достоинством, что при его использовании нет необходимости в перерас-  
пределении ресурсов и откате процессов назад. Однако использование этого метода требует выполнения  
ряда условий.

• Число пользователей и число ресурсов фиксировано.

• Число работающих пользователей должно оставаться постоянным.

• Алгоритм требует, чтобы клиенты гарантированно возвращали ресурсы.

• Должны быть заранее указаны максимальные требования процессов к ресурсам. Чаще всего дан-  
ная информация отсутствует.

Наличие таких жестких и зачастую неприемлемых требований может склонить разработчиков к выбору  
других решений проблемы взаимоблокировки.

Предотвращение тупиков за счет нарушения условий возникновения тупиков

В отсутствие информации о будущих запросах единственный способ избежать взаимоблокировки - до-  
биться невыполнения хотя бы одного из условий раздела "Условия возникновения тупиков".

**Основы операционных систем**

**68**

**Нарушение условия взаимоисключения**

В общем случае избежать взаимоисключений невозможно. Доступ к некоторым ресурсам должен быть  
исключительным. Тем не менее некоторые устройства удается обобществить. В качестве примера рас-  
смотрим принтер. Известно, что пытаться осуществлять вывод на принтер могут несколько процессов.  
Во избежание хаоса организуют промежуточное формирование всех выходных данных процесса на дис-  
ке, то есть разделяемом устройстве. Лишь один системный процесс, называемый сервисом или демоном  
принтера, отвечающий за вывод документов на печать по мере освобождения принтера, реально с ним  
взаимодействует. Эта схема называется спулингом (spooling). Таким образом, принтер становится разде-  
ляемым устройством, и тупик для него устранен.

К сожалению, не для всех устройств и не для всех данных можно организовать спулинг. Неприятным по-  
бочным следствием такой модели может быть потенциальная тупиковая ситуация из-за конкуренции за  
дисковое пространство для буфера спулинга. Тем не менее в той или иной форме эта идея применяется  
часто.

**Нарушение условия ожидания дополнительных ресурсов**

Условия ожидания ресурсов можно избежать, потребовав выполнения стратегии двухфазного захвата.

• В первой фазе процесс должен запрашивать все необходимые ему ресурсы сразу. До тех пор пока  
они не предоставлены, процесс не может продолжать выполнение.

• Если в первой фазе некоторые ресурсы, которые были нужны данному процессу, уже заняты дру-  
гими процессами, он освобождает все ресурсы, которые были ему выделены, и пытается повто-  
рить первую фазу.

В известном смысле этот подход напоминает требование захвата всех ресурсов заранее. Естественно, что  
только специально организованные программы могут быть приостановлены в течение первой фазы и  
рестартованы впоследствии.

Таким образом, один из способов - заставить все процессы затребовать нужные им ресурсы перед вы-  
полнением ("все или ничего"). Если система в состоянии выделить процессу все необходимое, он может  
работать до завершения. Если хотя бы один из ресурсов занят, процесс будет ждать.

Данное решение применяется в пакетных **мэйнфреймах** (mainframe), которые требуют от пользователей  
перечислить все необходимые его программе ресурсы. Другим примером может служить механизм  
двухфазной локализации записей в СУБД. Однако в целом подобный подход не слишком привлекателен  
и приводит к неэффективному использованию компьютера. Как уже отмечалось, перечень будущих за-  
просов к ресурсам редко удается спрогнозировать. Если такая информация есть, то можно воспользо-  
ваться алгоритмом банкира. Заметим также, что описываемый подход противоречит парадигме модуль-  
ности в программировании, поскольку приложение должно знать о предполагаемых запросах к ресурсам  
во всех модулях.

**Нарушение принципа отсутствия перераспределения**

Если бы можно было отбирать ресурсы у удерживающих их процессов до завершения этих процессов, то  
удалось бы добиться невыполнения третьего условия возникновения тупиков. Перечислим минусы дан-  
ного подхода.

Во-первых, отбирать у процессов можно только те ресурсы, состояние которых легко сохранить, а позже  
восстановить, например состояние процессора. Во-вторых, если процесс в течение некоторого времени  
использует определенные ресурсы, а затем освобождает эти ресурсы, он может потерять результаты ра-  
боты, проделанной до настоящего момента. Наконец, следствием данной схемы может быть дискрими-  
нация отдельных процессов, у которых постоянно отбирают ресурсы.

Весь вопрос в цене подобного решения, которая может быть слишком высокой, если необходимость от-  
бирать ресурсы возникает часто.

**Основы операционных систем**

**69**

**Нарушение условия кругового ожидания**

Трудно предложить разумную стратегию, чтобы избежать последнего условия из раздела "Условия воз-  
никновения тупиков" - циклического ожидания.

Один из способов - упорядочить ресурсы. Например, можно присвоить всем ресурсам уникальные номе-  
ра и потребовать, чтобы процессы запрашивали ресурсы в порядке их возрастания. Тогда круговое ожи-  
дание возникнуть не может. После последнего запроса и освобождения всех ресурсов можно разрешить  
процессу опять осуществить первый запрос. Очевидно, что практически невозможно найти порядок, ко-  
торый удовлетворит всех.

Один из немногих примеров упорядочивания ресурсов - создание иерархии спин-блокировок в Windows  
2000. Спин-блокировка - простейший способ синхронизации (вопросы синхронизации процессов рас-  
смотрены в соответствующей лекции). Спин-блокировка может быть захвачена и освобождена процес-  
сом. Классическая тупиковая ситуация возникает, когда процесс P1 захватывает спин-блокировку S1 и  
претендует на спин-блокировку S2, а процесс P2, захватывает спин-блокировку S2 и хочет дополнитель-  
но захватить спин-блокировку S1. Чтобы этого избежать, все спин-блокировки помещаются в упорядо-  
ченный список. Захват может осуществляться только в порядке, указанном в списке.

Другой способ атаки условия кругового ожидания - действовать в соответствии с правилом, согласно ко-  
торому каждый процесс может иметь только один ресурс в каждый момент времени. Если нужен второй  
ресурс - освободи первый. Очевидно, что для многих процессов это неприемлемо.

Таким образом, технология предотвращения циклического ожидания, как правило, неэффективна и мо-  
жет без необходимости закрывать доступ к ресурсам.

**Обнаружение тупиков**

Обнаружение взаимоблокировки сводится к фиксации тупиковой ситуации и выявлению вовлеченных в  
нее процессов. Для этого производится проверка наличия циклического ожидания в случаях, когда вы-  
полнены первые три условия возникновения тупика. Методы обнаружения активно используют графы  
распределения ресурсов.

Рассмотрим модельную ситуацию.

• Процесс Р1 ожидает ресурс R1.

• Процесс Р2 удерживает ресурс R2 и ожидает ресурс R1.

• Процесс Р3 удерживает ресурс R1 и ожидает ресурс R3.

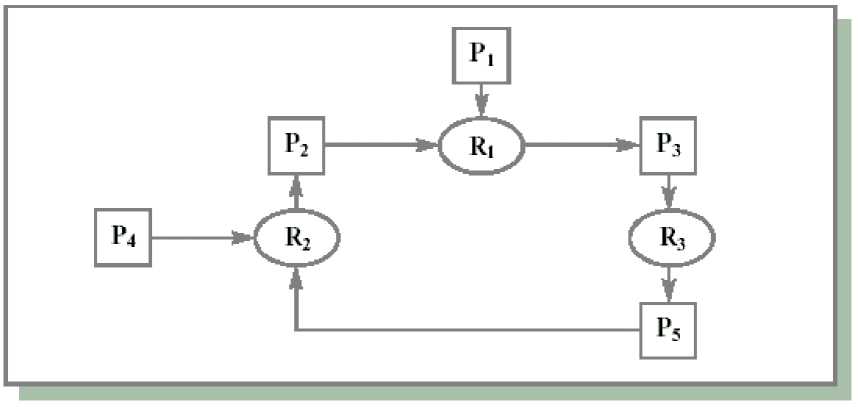
• Процесс P4 ожидает ресурс R2.

• Процесс P5 удерживает ресурс R3 и ожидает ресурс R2.

Вопрос состоит в том, является ли данная ситуация тупиковой, и если да, то какие процессы в ней участ-  
вуют. Для ответа на этот вопрос можно сконструировать граф ресурсов, как показано на рис. 7.3. Из ри-  
сунка видно, что имеется цикл, моделирующий условие кругового ожидания, и что процессы P2,P3,P5, а  
может быть, и другие находятся в тупиковой ситуации.

**Основы операционных систем**

**70**



**Рис. 7.3.** Граф ресурсов

Визуально легко обнаружить наличие тупика, но нужны также формальные алгоритмы, реализуемые на  
компьютере.

Один из таких алгоритмов описан в ГТаненбаум, 20021, там же можно найти ссылки на другие алгоритмы.

Существуют и другие способы обнаружения тупиков, применимые также в ситуациях, когда имеется не-  
сколько ресурсов каждого типа. Так в ГДейтел, 19871 описан способ, называемый редукцией графа рас-  
пределения ресурсов, а в ГТаненбаум, 20021 - матричный алгоритм.

**Восстановление после тупиков**

Обнаружив тупик, можно вывести из него систему, нарушив одно из условий существования тупика. При  
этом, возможно, несколько процессов частично или полностью потеряют результаты проделанной рабо-  
ты.

Сложность восстановления обусловлена рядом факторов.

• В большинстве систем нет достаточно эффективных средств, чтобы приостановить процесс, вы-  
вести его из системы и возобновить впоследствии с того места, где он был остановлен.

• Если даже такие средства есть, то их использование требует затрат и внимания оператора.

• Восстановление после тупика может потребовать значительных усилий.

Самый простой и наиболее распространенный способ устранить тупик - завершить выполнение одного  
или более процессов, чтобы впоследствии использовать его ресурсы. Тогда в случае удачи остальные  
процессы смогут выполняться. Если это не помогает, можно ликвидировать еще несколько процессов.  
После каждой ликвидации должен запускаться алгоритм обнаружения тупика.

По возможности лучше ликвидировать тот процесс, который может быть без ущерба возвращен к началу  
(такие процессы называются идемпотентными). Примером такого процесса может служить компиляция.

С другой стороны, процесс, который изменяет содержимое базы данных, не всегда может быть корректно  
запущен повторно.

В некоторых случаях можно временно забрать ресурс у текущего владельца и передать его другому про-  
цессу. Возможность забрать ресурс у процесса, дать его другому процессу и затем без ущерба вернуть  
назад сильно зависит от природы ресурса. Подобное восстановление часто затруднительно, если не не-  
возможно.

В ряде систем реализованы средства отката и перезапуска или рестарта с контрольной точки (сохранение  
состояния системы в какой-то момент времени). Если проектировщики системы знают, что тупик вероя-

**Основы операционных систем**

**71**

тен, они могут периодически организовывать для процессов контрольные точки. Иногда это приходится  
делать разработчикам прикладных программ.

Когда тупик обнаружен, видно, какие ресурсы вовлечены в цикл кругового ожидания. Чтобы осущест-  
вить восстановление, процесс, который владеет таким ресурсом, должен быть отброшен к моменту вре-  
мени, предшествующему его запросу на этот ресурс.

**Заключение**

Возникновение тупиков является потенциальной проблемой любой операционной системы. Они возни-  
кают, когда имеется группа процессов, каждый из которых пытается получить исключительный доступ к  
некоторым ресурсам и претендует на ресурсы, принадлежащие другому процессу. В итоге все они оказы-  
ваются в состоянии бесконечного ожидания.

С тупиками можно бороться, можно их обнаруживать, избегать и восстанавливать систему после тупи-  
ков. Однако цена подобных действий высока и соответствующие усилия должны предприниматься толь-  
ко в системах, где игнорирование тупиковых ситуаций приводит к катастрофическим последствиям.