Лекция 9: Тупики / Deadlocks

Алексей Линёв
Александр Мощук
Кирилл Погорельский

somes slides are adapted from the OS course at the University of Washington

Тупики

Что такое тупик?

- Существуют неразделяемые аппаратные ресурсы
  - некоторые могут иметься более, чем в одном экземпляре
  - Принтеры, семафоры, ленточные накопители, ЦП
- Процессам необходим доступ к этим ресурсам
  - Получить/захватить (acquire) ресурс
    - если ресурс свободен, он предоставляется процессу
    - если ресурс занят, процесс блокируется
  - Использовать ресурс
  - Освободить (release) ресурс
- Нежелательная ситуация
  - Процесс A захватил ресурс 1, и ожидает предоставления ему ресурса 2
  - Процесс B захватил ресурс 2, и ожидает предоставления ему ресурса 1

  ⇒ Это **тупик**! (**deadlock**, **взаимоблокировка**)

Пример: Семафоры

```
semaphore:    mutex1 = 1   /* управляет доступом к ресурсу 1 */
              mutex2 = 1   /* управляет доступом к ресурсу 2 */
```

Код процесса A:

```
{
  /* инициализация */

  P(mutex1)
  P(mutex2)

  /* использование
     обоих ресурсов */

  V(mutex2)
  V(mutex1)
}
```

Код процесса B:

```
{
  /* инициализация */

  P(mutex2)
  P(mutex1)

  /* использование
     обоих ресурсов */

  V(mutex2)
  V(mutex1)
}
```

Тупик

- Множество процессов попало в тупик, если
  - Каждый процесс ожидает некоторого события
  - Это событие может быть вызвано только действиями другого процесса из данного множества
    - Частный случай ожидаемого события - освобождение некоторого ресурса

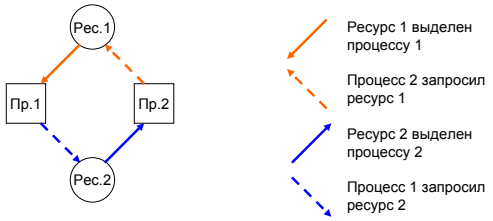
Четыре условия существования тупика

- Коффман, Элфик и Шошани в 1971 г. сформулировали
- Mutual Exclusion (Взаимное исключение)
  - По крайней мере один из запрашиваемых ресурсов является неделимым (то есть должен захватываться в эксклюзивное использование)
- Hold and wait (Удержание ресурсов при ожидании)
  - Существует процесс, владеющий некоторым ресурсом и ожидающий освобождения другого ресурса
- No preemption (Неперераспределяемость ресурсов)
  - Ресурсы не могут быть отобраны у процесса без его желания
- Circular wait (Циклическое ожидание)
  - Существует множество процессов $\{P_1, P_2, \ldots P_N\}$, таких что
    - P_1 ждет P_2, P_2 ждет P_3, …. и P_N ждет P_1
- **Для появления тупика должны выполниться все четыре условия**

Граф "Процесс-Ресурс"

- Тупик может быть проиллюстрирован на графе "процесс-ресурс"
- Направленный граф "процесс-ресурс" включает:
  - Множество вершин $V = P \cup R$,
    - где $P=\{P_1,P_2,\ldots,P_n\}$ – множество процессов, $R=\{R_1,R_2,\ldots,R_m\}$ – множество ресурсов
  - Дуги запросов – направлены от процессов к ресурсам
    - дуга $P_i \rightarrow R_j$ означает, что процесс P_i запросил ресурс R_j
  - Дуги распределения – направлены от ресурсов к процессам
    - дуга $R_j \rightarrow P_i$ означает, что ресурс R_j выделен процессу P_i
- Если граф "процесс-ресурс" не имеет циклов, тупиков нет
- Если граф "процесс-ресурс" имеет цикл, возможно, тупик существует

Граф "процесс-ресурс"

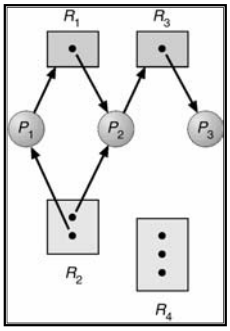


- Тупик имеет место, если в графе "процесс-ресурс" имеется неприводимый цикл (например такой, как на рисунке)

Редукция графа "процесс-ресурс"

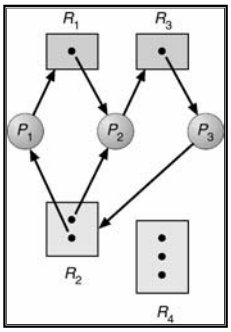
- Граф может быть сокращен за счет процесса, все запросы которого могут быть удовлетворены
  - в подобной ситуации, если поток завершается – все его ресурсы освобождается – все дуги распределения ресурсов этому потоку в графе удаляются
- Существует ряд теорем, связанных с данной темой
  - Если граф полностью редуцируем, тупиков не существует
  - Порядок выполнения редукции не имеет значения
- Имеется тонкость при работе со счетными ресурсами

Граф "процесс-ресурс" без циклов

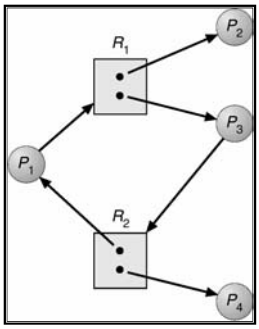


Какие запросы могут привести к возникновению тупика?

Граф "процесс-ресурс" с тупиком



Граф "процесс-ресурс" с циклом, но без тупика



Решения проблемы тупиков

- Предупреждение тупиков
  - Предотвращение тупиков (prevention)
    - устранение одного из 4 условий существования тупика не допускает возникновения тупиков
  - Избегание тупиков (avoidance)
    - аккуратное распределение ресурсов на основании имеющейся информации об их использовании в будущем позволяет избежать возникновения тупиков
- Устранение тупиков
  - Обнаружение тупика и восстановление после возникновения тупика (detection & recovery)
    - позволяем тупику возникнуть, затем обнаруживаем и устраняем его
- Игнорирование проблемы
  - Считаем, что тупики никогда не возникают
  - Самообман, "Алгоритм страуса"

Предотвращение тупиков

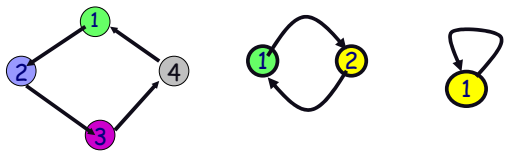
- Может ли ОС предупредить возникновение тупиков?
- Предотвращение – устранение одного из условий существования тупика
  - Mutual exclusion (взаимное исключение)
    - Можно попытаться сделать ресурсы разделяемыми
    - Это не всегда возможно
  - Hold and wait (удержание ресурсов при ожидании)
    - Не оставлять за собой владение уже имеющимися ресурсами при запросе дополнительных
    - ⇒ Можно запрашивать все необходимые ресурсы до начала выполнения
    - Процессы не знают, какие ресурсы им понадобятся
    - Голодание (при ожидании множества популярных ресурсов)
    - Неэффективное использование ресурсов (возможно, часть ресурсов нужна на очень короткое время)
    - Другой вариант: при возникновении потребности в дополнительных ресурсах – освобождаем все уже имеющиеся, затем запрашиваем те, что необходимы в настоящий момент
      - Две последние проблемы остаются

Предотвращение тупиков

- Предотвращение - устранение одного из условий существования тупика
  - No preemption (неперераспределяемость ресурсов):
    - Сделаем ресурсы перераспределяемыми (2 подхода)
    - 1. Если процесс затребовал ресурсы, часть их которых недоступны, перераспределяем принадлежащие ему ресурсы
    - 2. Перераспределяем ресурсы процессов в состоянии ожидания для удовлетворения поступившего запроса
    - Подходящее решения для ситуаций, когда состояние ресурса просто сохранить и восстановить
      - регистры ЦП, виртуальная память
  - Circular wait (циклическое ожидание) – 2 решения
    - Процессам позволяется владеть только одним ресурсом в каждый момент времени (Предположите недостатки подобного подхода)
    - Для ресурсов вводится нумерация, процессы должны запрашивать ресурсы в порядке возрастания их номеров

Нарушение условия циклического ожидания

- Ресурсам присваиваются номера (lock1, lock2, …)
- Позволяется запрашивать ресурсы строго в порядке возрастания (или убывания) их номеров
- Идея: Цикл требует, чтобы в нем были дуги как от вершин с меньшими номерами к вершинам с большими номерами, так и наоборот; либо, должна быть дуга из какой-то вершины в нее же



- Нумерация не всегда возможна
- Неэффективное использование ресурсов

Избегание тупиков

- Если у нас имеется информация о будущем
  - Максимальные требования всех процессов к ресурсам известны до начала выполнения процессов
- Можем ли мы гарантировать, что тупик не возникнет?
- Стратегия избегания тупиков, Алгоритм Банкира:
  - Перед выделением ресурса проверяем, является ли новое состояние **безопасным**
  - Если новое состояние безопасно ⇒ тупика не будет!
    - выделяем ресурс
  - Если новое состояние небезопасно, блокируем процесс, выполнивший запрос

"Безопасное" состояние

- Состояние называется **безопасным**, если для него имеется последовательность процессов

  $\{P_1, P_2, \ldots, P_n\}$, такая, что для каждого P_i,

  ресурсы, которые затребовал P_i, могут быть предоставлены за счет имеющихся незанятых ресурсов и ресурсов, выделенных всем процессам P_j, где $j < i$
- Состояние безопасно, поскольку ОС может гарантированно избежать тупика
  - посредством блокирования любых новых запросов пока не выполнится безопасная последовательность
- Такой подход позволяет избежать возникновения циклического ожидания
  - Процессы ждут до тех пор, пока не будет гарантировано сохранение безопасного состояния

Пример безопасного состояния

- Предположим, имеются 12 устройств хранения на магнитной ленте

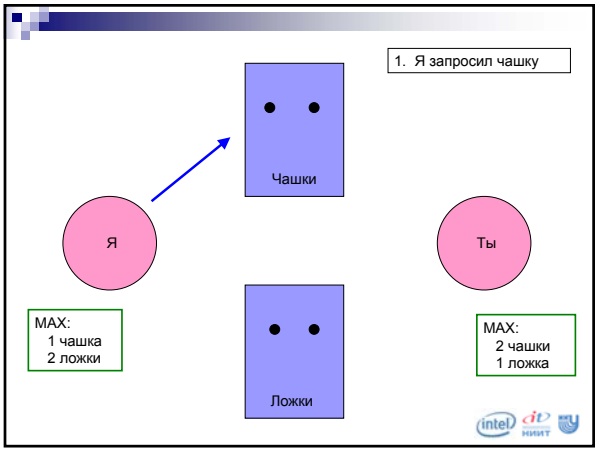
Процесс	MAX потребность	Владеет	Может запросить
p0	10	5	5
p1	4	2	2
p2	9	2	7

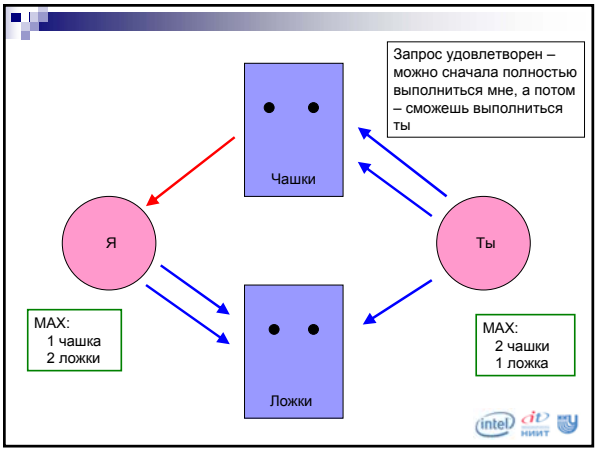
3 устройства свободны

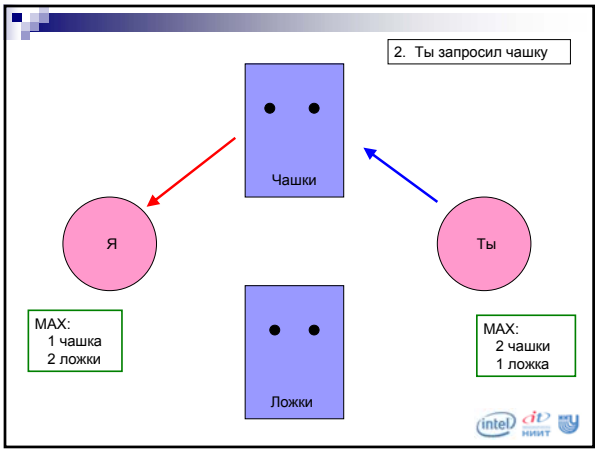
- Текущее состояние безопасно, поскольку существует безопасная последовательность: <p1,p0,p2>
  - p1 может завершиться, используя только свободные ресурсы
  - p0 может завершиться, используя свободные ресурсы + ресурсы, которые сейчас выделены процессу p1
  - p2 может завершиться, используя свободные ресурсы + ресурсы, которые сейчас выделены процессам p1 и p0
- Если p2 запросит еще 1 устройство, удовлетворение запроса будет отложено, поскольку оно переведет систему в небезопасное состояние
  - почему?

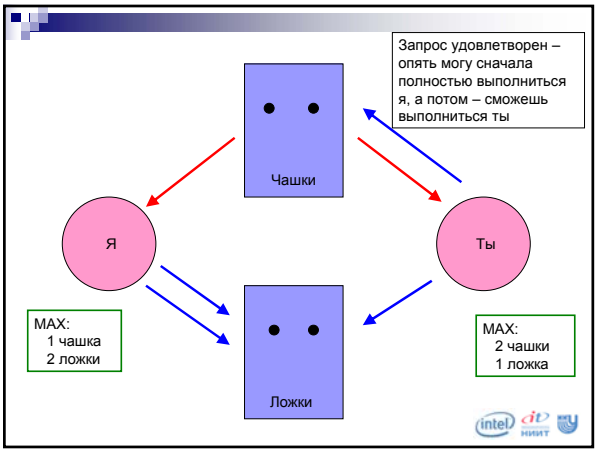
Алгоритм Банкира с графами "процесс-ресурс"

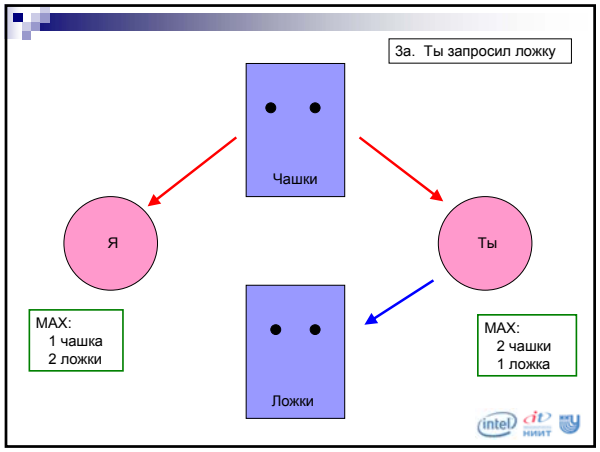
- При выполнении запроса
  - представляем, что мы его удовлетворили
  - представляем, что все остальные возможные запросы удовлетворены
  - проверяем, может ли граф "процесс-ресурс" быть полностью редуцирован?
    - если да, выделяем запрошенный ресурс
    - если нет, блокируем поток, выполнивший запрос

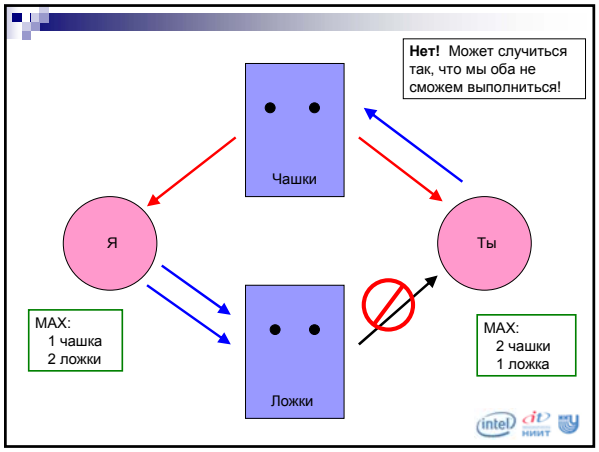


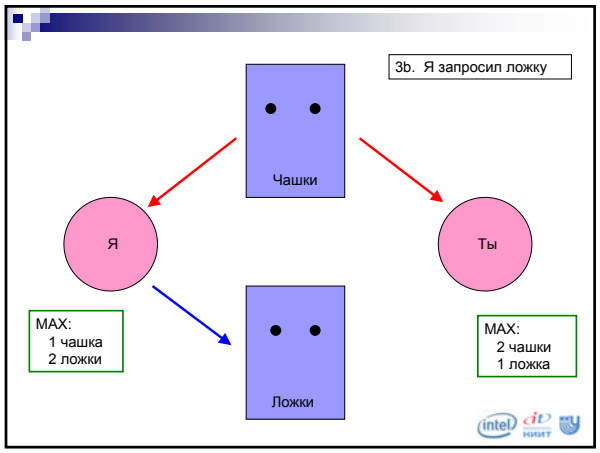


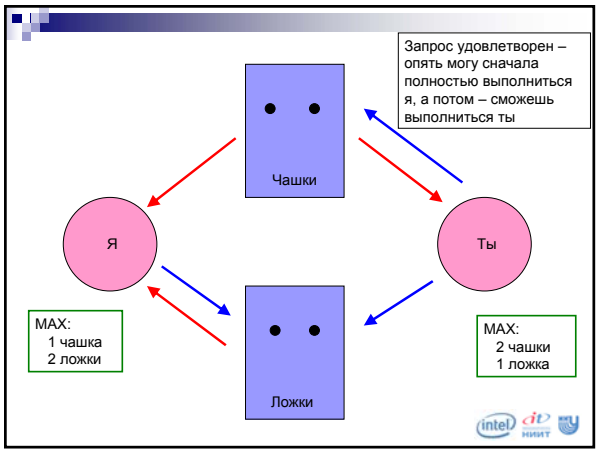












Алгоритм Банкира: еще один пример

A,B,C – ресурсы

	Выделено			Max			Доступно		
	A	B	C	A	B	C	A	B	C
P0	0	1	0	7	5	3	3	3	2
P1	2	0	0	3	2	2			
P2	3	0	2	9	0	2			
P3	2	1	1	2	2	2			
P4	0	0	2	4	3	3			

Безопасно ли текущее состояние?

Алгоритм Банкира: еще один пример

A,B,C – ресурсы

	Выделено			Max			Доступно		
	A	B	C	A	B	C	A	B	C
P0	0	1	0	7	5	3	3	3	2
P1	2	0	0	3	2	2			
P2	3	0	2	9	0	2			
P3	2	1	1	2	2	2			
P4	0	0	2	4	3	3			

Можно ли удовлетворить запрос P0 на (1,2,2)?

Обнаружение тупика
Восстановление после тупика

- Если ни один из перечисленных подходов не используется, может возникнуть тупик
- В таком случае необходимо:
  - Обнаружить возникновение тупика. Для этого нужно:
    - отслеживать выделение ресурсов (какой процесс каким ресурсом владеет)
    - отслеживать поступающие запросы (какой процесс какой ресурс ожидает)
  - Иметь решения для восстановления из тупика
- Очень накладно поддерживать и обнаружение, и восстановление

Когда запускать алгоритм обнаружения тупика?

- При каждом запросе на выделение ресурса?
- При каждом запросе, который не может быть немедленно удовлетворен?
- Каждый час?
- Когда средняя загрузка ЦП станет меньше 40%?

Восстановление после тупика

- Уничтожение одного/всех процессов, участвующих в тупике
  - Грубо, но эффективно
  - Можно продолжать уничтожать процессы, пока тупик не распадется
  - Затем все вычисления повторяются
- Перераспределение ресурсов между процессами вплоть до разрушения тупика
  - Отобрать ресурс у владельца и отдать другому процессу
    - выбор жертвы?
- Откат выбранного процесса к некоторой контрольной точке или к началу (partial or total rollback)

SQL Server

- Запускает алгоритм обнаружения тупика:
  - Периодически или
  - По запросу
- Выполняет восстановление посредством уничтожения:
  - наименее "дорогого" процесса
  - процесса, выбранного в соответствии с приоритетом, указанным пользователем

"Transaction (Process ID xxx) was deadlocked on (zzz) resources with another process and has been chosen as the deadlock victim. Rerun the transaction."

Текущее положение дел

- **Страусовый алгоритм**
- Избегание и предупреждение тупиков часто невозможны
- Полный перебор всех сценариев – слишком дорогая процедура
- Во всех ОС возможно возникновение тупиков
- Инженерный подход:

  За производительность и удобство работы пользователей стоит заплатить такую цену, как нечастые сбои в работе.
  (The price of infrequent crashes in exchange for performance and user convenience is worth it)