Тупики.
(продолжение темы „взаимодействие процессов").

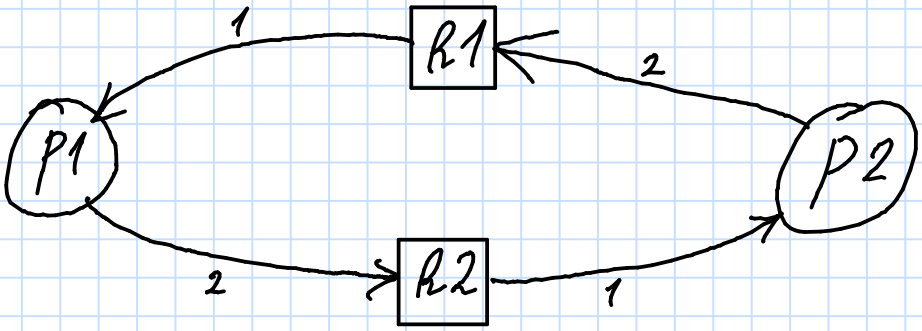
Задача об обедающих философах

Семафоры: процессы изменяют состояния двух семафоров в обратном порядке

Наиб. наглядная тупиковая ситуация в ОС:
(Графовая модель Холта
или двудольный направленный граф)

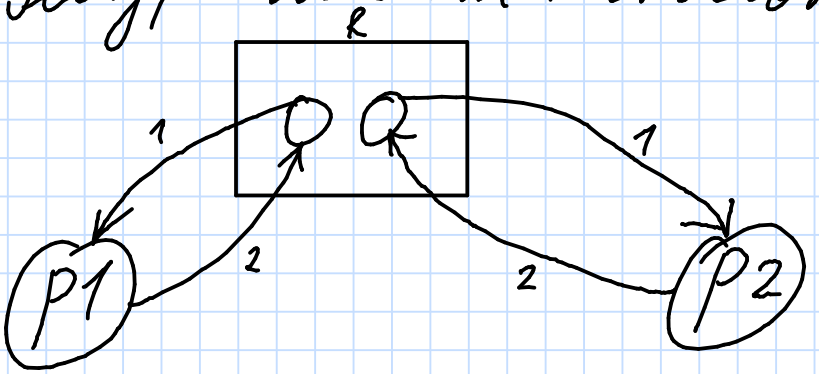
□ – ресурс; ○ – процесс;

1)



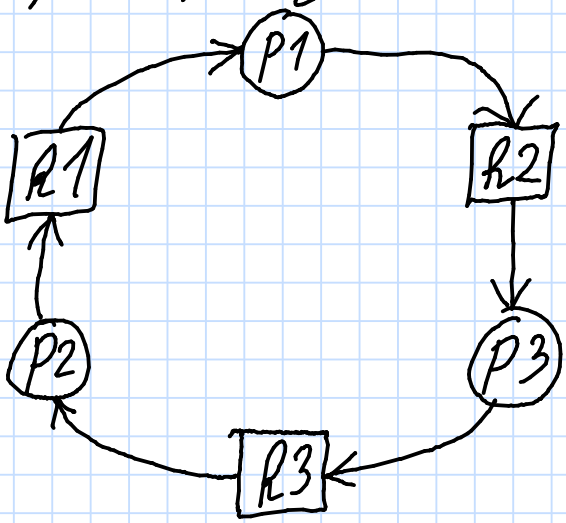
Процесс → ресурс – запрос
Ресурс → процесс – выделение

2) Ресурс имеет несколько единиц:



Опр. Тупиковая ситуация (тупик) – ситуация, возникающая в рез-те монопольного исп-я разд. ресурсов, когда процесс, владея ресурсом, запрашивает др. ресурс, занятый непосредственно или через

цепочку запросов, другим процессам, ожидающим освобождения ресурса, занятого 1-м процессом;
Пример цепочки вызовов:



„Тупик становится проблемой тогда, когда он возникает";

Типы ресурсов и их тупики. (и практические методы их обнаружения)
Самое полное изложение теории тупиков – книга Шоу – „Логическое проектирование ОС";
Вечные проблемы тупиков: глубокий анализ и поиск методов решения или перезагрузка, но не каждую систему можно перезагрузить;

Классы ресурсов:
1) повторно используемые (фарфоровая тарелка);
2) потребляемые (еда на этой тарелке);

Очередь – гарантия монопольного использования (в частности, к процессору выстраивается очередь процессов)

1 класс: аппаратное обеспечение, сист. таблицы, реентерабельные коды ОС, объекты ядра (семафоры + таймеры, очереди сообщений, программные каналы, разд. память);
2 класс: только сообщения;

(сообщения после их получения перестают существовать);

Программные каналы

Тупик на семафоре – простейший пример тупика;

С сист. таблицами могут работать только привилегированные процессы;

В системе мы работаем с повторно используемыми ресурсами (например, с таймерами);

Св-ва повторно используемых ресурсов:

1) Число единиц повторно исп. ресурсов ограничено (в текущий момент времени); (они дефицитны, но неизменны (в тек. момент вр.));
2) Если число единиц повт. исп. ресурсов изменяется, то это исключительная ситуация в системе.

Св-ва потребляемых ресурсов:

1) Число единиц потр. ресурсов может изменяться, если не накладываются доп. ограничения (и зачастую они накладываются);

Любой процесс может произвести любое кол-во сообщений (по сущ. ограничения)

Условия возникновения тупиков: (по Ховедеру)

1) взаимоисключение, когда процессы монопольно используют предоставляемые им ресурсы (mutual exclusion);

2) Ожидание, когда процессы удерживают занятые ресурсы, ожидая предоставления доп. ресурсов для продолжения выполнения (hold and wait - удерживать и ждать);

3) Неперераспределяемость, когда у процесса нельзя отобрать ресурс до его завершения или пока он сам его не освободит;

4) Круговое ожидание, когда возникает замкнутая цепь процессов, инициирующих запросы на ресурсы, и в этой цепи каждый процесс занимает ресурс, который необходим следующему процессу для его успешного продолжения (circular wait);

Условия неравнозначны: возникновение замкнутой цепи запросов (4) фактически явл. тупиковой ситуацией, а условия 1-3 явл. предпосылками для возникновения такой ситуации.

Методы борьбы с тупиками:

Для русского языка многие понятия многозначны, хотя казалось бы...

1) Недопущение (исключение) тупиков: создание в системе такой ситуации, когда тупики в принципе невозможны;

2) Обход (предотвращение) тупика: тупик в принципе возможен, но можно приложить усилия для его предотвращения;

3) Обнаружение тупика: тупик возможен, и когда он возникает, выполняются действия по обнаружению тупика, т.е. в системе выявляются процессы, которые попали в тупик;

4) Восстановление: система попала в тупик, обнаружены процессы, попавшие в тупик, и этот тупик надо ликвидировать и восстановить работоспособность системы, если это возможно;

Обнаружение и восстановление — 2 стороны одной медали;

Теперь подробнее:

1) Предотвращение:
Стратегия Хавендера:
тупик не возникнет, если нарушено хотя бы одно из 4 условий возникновения тупика;

Решения:

1) Вызывается опережающее требование: процесс сразу запрашивает все необходимые ему ресурсы (нарушается условие ожидания: процесс запросил и м.б. сразу всё получил, а если не получил, то возможно бесконечное откладывание);

Процессы должны знать свою потребность в разных типах ресурсов, но в совр. ОС это практически невозможно;

В языке упр-я заданиями указывалось макс. объём памяти, необходимый для выполнения, макс. кол-во проц. времени, внеш. устр-ва; старые ОС в соотв-ии с этими требованиями выстраивали процессы в очередь (самый "ярки" алгоритм планирования: shortest job first)

С т.з. использования ресурсов системы операционное требование в каких-то системах скорее всего возможно, т.е. исключать такую возможность нельзя. (т.к. м.б. детерминированные процессы, про которые всё известно заранее)

Но такой метод приводит к неэффективному использованию ресурсов, т.к. процессы запрашивают и получают ресурсы, возможно, задолго до их реального использования.

Так было в старых системах, и это было необходимо.

ОСРВ в тупики попадать не могут!

(в ещё можно совмещать операционное требование с разделением задач на подзадачи)

2) Иерархическое распределение:
(часто реализуется путём упорядочивания ресурсов)
Как правило ресурсы либо являются одиночными, либо принадлежат к какому-то типу (классу) ресурсов. Тогда иерархическое распр-е предполагает, что каждому классу присваивается номер (уровень иерархии, хотя это очень условно) и ставится след. условие: процесс, удерживая ресурсы,

может запросить только ресурсы с бо́льшими номерами, чем номера ресурсов, которые он удерживает.
Т.е. ресурсы могут запрашиваться только в порядке, определённом иерархическим распр-ем. Если процессу требуется ресурс с меньшим номером, он должен освободить все ресурсы и запросить ресурсы в правильном порядке.

Метод опереж. требования предполагает, что процессы должны знать о своей макс. потребности в ресурсах, что не всегда возможно.
А здесь более гибкая система (операции перераспределения ресурсов).
Здесь также исключается условие кругового ожидания.

Очевидно, что в такой системе номера ресурсов должны отражать наиболее вероятный порядок использования ресурсов. При этом, как правило, самые дефицитные ресурсы получают наибольшие номера.

Однако, несмотря на то, что данный подход решает проблему тупиков (за счёт систематизации ресурсов путём их нумерации), часто оказывается невозможным определить порядок нумерации, удовлетворяющий всех, т.к. повт. используемые ресурсы совершенно разные (включая сист. таблицы, объекты ядра, файлы и просто разд. данные), т.е. число потенциальных

ресурсов и вариантов их использования очень большое, поэтому достаточно сложно предложить их удовлетворительную систематизацию.

Самый негативный вариант: процесс запрашивает ресурсы в порядке, обратном их нумерации (такую возможность исключать нельзя). В этом случае процесс будет действовать как процесс, вынужденный выполнить опережающее требование, прежде этим он выполнит n-ное кол-во запросов и освобождений ресурсов.

3) Устранение условия неперераспределяемости:
Если процесс в рез-те сделанного запроса не может получить нужный ресурс, он блокируется и освобождает часть занимаемых ресурсов (или все), т.е. переходит в состояние ожидания более благоприятной ситуации в системе.

Такой подход приводит к запросу и получению одних и тех же ресурсов; (вероятностные связи)

Все стратегии характеризуются непредсказуемостью времени выполнения процессов (сколько они будут ждать)

(2) Обход тупиков:
Тупики возможны, чье возникновение предотвращается (обходится);
Самое известное решение:
Алгоритм банкира (Дейкстра):
действия подобны действиям банкира, кото-

рый имеет капитал и ссужает заёмщиков деньгами. При этом банкир должен быть осторожен и анализировать, сможет ли заёмщик вернуть полученные деньги.
Часто для того, чтобы вернуть первоначально полученную сумму, заёмщик должен брать доп. ссуду.
Банкир - менеджер ресурсов;
Заёмщики - процессы, которые делают заявки на ресурсы.

При этом заявка на каждый тип ресурса должна отражать макс. потребность процесса в данном ресурсе (макс. кол-во ед. ресурса, который может потребоваться процессу). На каждый тип ресурса процесс выдаёт соотв. заявку. В итоге процесс должен знать свою макс. потребность в ресурсах, чтобы сформулировать заявки.

Ограничения:
1) Процесс не может во время выполнения не может затребовать ресурсов больше, чем он указал в своей заявке.
2) Число процессов в системе фиксировано (известно). Число ресурсов в системе известно и неизменно.

Эти ограничения позволяют менеджеру ресурсов проводить анализ ситуации в системе.

Заявка - advanced claim;
(предварительная заявка на каждый тип ресурса)

Условия выполнения алгоритма банкира:
1) Процесс должен сформулировать предварительные заявки.
2) Процесс не может требовать ресурсов больше, чем

имеется в системе.

3) Процесс не может запрашивать ресурсов больше, чем указано в его заявке.

4) Сумма всех выделенных ресурсов данного класса не может превышать общего числа единиц ресурса данного класса в системе.
(сумма уже выделенных единиц и запрашиваемых не может превышать общего кол-ва единиц данного ресурса в системе)

На основе этих ограничений менеджер ресурсов гарантирует, что тупиковая ситуация не наступит. Для этого каждый запрос проверяется по отношению к кол-ву ресурсов в системе и по отношению к кол-ву свободных единиц данного ресурса в тек. момент времени. Менеджер ресурсов ищет посл-ть процессов, которая в сложившейся текущей ситуации может гарантированно завершиться. Если такая посл-ть существует, то такое состояние наз. безопасным по отн-ю к тупику, т.е. в рез-те удовлетворения запросов процессов тупик не наступит.

Пример:

Процессы	Текущее распр-е	Заявки (макс. потребность процессов)	Свободных единиц ресурса
P1	4	12	
P2	2	4	
P3	4	8	
	Σ: 10		2

(больше запросить не могут)

→ 13 запросить не могы бы, т.к. у системы есть только 10+2=12 ед.

P2 может гарантированно завершиться, даже

если он запросит макс. кол-во единиц ресурса в соответствии с его заявкой.

Освободившиеся процессом P2 единицы ресурса совместно со свободными позволят удовлетворить потребность процесса P3 и он также сможет успешно завершиться.

В рез-те завершения процессов P2 и P3 освободившиеся единицы ресурса удовлетворяют потребность процесса P1.

Т.о. в системе существует посл-ть процессов, которая может успешно завершиться.

Это был пример безопасной ситуации, на след. лекции будет пример опасной ситуации отн. тупиков и алгоритм Хобермана;