# Тупики

Книга Шоу – логическое проектирование ОС

От ресурса к процессу – получение, наоборот – запрос

Определение: тупик (тупиковая ситуация) – ситуация, возникающая в результате монопольного использования разделяемых ресурсов, когда процесс, владея ресурсом, запрашивает другой ресурс, занятый непосредственно или через цепочку запросов другим процессом, который ожидает освобождения ресурса, занятого первым процессом. В результате процессы блокируют друг друга таким образом, что ни один из них не может продолжить выполнение и освободить занимаемый ими ресурс

## Типы ресурсов и тупики

Ресурсы с точки зрения особенности их использования, делятся на повторно используемые (ресурсы, использование которых никак не меняет их свойства) и потребляемые.

К повторно используемым относятся аппаратура компьютера, процессоры, память, каналы, устройства ввода-вывода и ПО (например, реентерабельные коды ОС, системные таблицы).

К потребляемым – сообщения. Взяли – и оно перестает существовать.

Память раньше была важным ресурсом, сейчас – проще. Процессор – как средство выполнения.

Простые очереди (то, что раньше называлось pulling теперь не используется).

Семафор – повторно используемый, так как структура никак не меняется после захвата и освобождения, функции также не меняются. Вообще все это называется объектами ядра.

Вся теория тупиков написана для повторно используемых ресурсов, потому что их количество в системе как правило известно и неизменно (не меняется резко), в отличие от сигналов (любой процесс может произвести или потребить любое количество сигналов). А как же семафоры – если есть приложение, которое использует набор семафоров, то мы определили количество семафоров в наборе, оно динамически не меняется, с этим можно работать.

Чтобы подчеркнуть, что теория тупиков в настоящее время не актуальны (так как там про аппаратное, а это сильно поменялось). Но это никак не коснулось используемых объектов ядра.

## 4 условия возникновения тупика

Были сформулированы 4 условия возникновения тупика (тупиковой ситуации, deadlock – захват смерти). Эти условия актуальны и в настоящее время

1. Взаимоисключение - Mutual exclusion
2. Ожидание – когда процессы, удерживая полученные им ресурсы, запрашивают и ждут получения дополнительных ресурсов, чтобы продолжить свое выполнение. Hold and wait
3. Не перераспределяемость – когда ресурсы нельзя отобрать у процесса до его завершения или добровольного освобождения ресурса. No preenption
4. Круговое ожидание – когда существует замкнутая цепь процессов, в которой каждый процесс занимает необходимый другому (следующему в цепи) процессу ресурс. Circular wait

Тупики в ОС в настоящее время не настолько актуальны, так как в ОС написаны с умом и принимаются меры по избеганию тупиков. Но тупики характерны для различных программ, выполняющих определенные вычисления, услуги, и особенно для распределенных систем.

## 3 основных метода борьбы с тупиками.

1. Недопущение тупиков (исключение самой возможности возникновения тупиков).
2. Обход тупиков (предотвращение тупиков)
3. Обнаружение и восстановление

### 1. Недопущение тупиков

1. **Первый подход (целый набор) – стратегия Харвендера**: возникновение невозможно, если нарушено хотя бы 1 из условий возникновения тупика.

Первое решение – опережающее требование. До своего начала процесс запрашивает все необходимые ему ресурсы. Процесс начнет свое выполнение только получив все запрошенные ресурсы. Очевидные недостатки: 1) процесс должен знать свою максимальную потребность во всех типах ресурсов. 2) процесс запрашивает и получает ресурсы задолго до их реального использования. При этом часть ресурсов он может вообще не использовать при данном проходе – неэффективное использование ресурсов. Это касается аппаратной составляющей.

Это также может привести к бесконечному откладыванию.

Раньше - Задания разбивались на части, каждая часть запрашивала.

Надо сказать, какое из условий возникновения тупика нарушается в этом подходе!!!! (вроде ожидание)

1. **Второй подход – реализуется с помощью упорядочивания ресурсов.** Также подход называется иерархическим распределением. Ресурсы делятся на классы. Деление должно определяться типом ресурса, быть оправданным (в один ресурс не поместят память и семафор). Каждому классу присваивается номер и устанавливается правило, по которому процессы могут захватывать только ресурсы с большими номерами, чем ресурсы, которые они удерживают.

Тоже нарушает одно из условий.

Если процессу требуется ресурс с меньшим номером, чем ресурсы, которые он уже получил, чтобы получить ресурсы в правильном порядке, должен освободить занимаемые им ресурсы, а потом захватить в правильном порядке.

Очевидно, что набор ресурсов в системе широкий, в системах пытаются так сформировать структуру, чтобы отобразить наиболее часто используемую последовательность захвата ресурсов. Но это непросто.

Системы реального времени в тупики попадать не могут. Это техническое понятие, РВ – выполнение обработки запроса, внешнего по отношению к процессу – за строго определенный интервал времени. Это определяемая характеристиками внешнего процесса или объекта – они определяют время выполнения.

В системах РВ процессы, которые выполняются в системе, хорошо изучены, у разработчиков есть полная информация о них, и для таких систем вполне может применяться иерархический метод.

1. **Третий способ. Устраняется условие неперераспределяемости**. Если процесс не может получить нужный ему ресурс, то он должен освободить занимаемый им ресурс.

Отобрать у процесса ресурс - значит откатить процесс к состоянию до получения данного ресурса. А чтобы откатить, надо запомнить точку отката, или выполнять обратные действия. Отобрать ресурс – большая проблема. Она должна реализовываться специальным образом.

Существуют такие реализации, когда для процесса пишутся обратные действия. Нелинейно

Этот подход может привести к захвату и освобождению одних и тех же ресурсов.

Надо иметь в виду, что системы существенно изменились

### 2. Обход тупиков

**1. Алгоритм банкира**

Классика – алгоритм банкира. Его предложил Дейкстра. Он вообще теоретический. Существуют аппроксимации – например, алгоритм Хаббермана.

Базовый алгоритм банкира.

Ограничения:

1. Число процессов известно, ограничено, то есть не меняется
2. Число ресурсов и число единиц каждого ресурса известно, не меняется

Процессы до начала выполнения должны указать в своих заявках (claim) свою максимальную потребность в каждом типе ресурса и затем при выполнении процесс не может захватывать ресурсов больше, чем указано в его заявке (естественно, нельзя заказывать больше, чем используется в системе). Работа менеджера ресурсов выполняется по алгоритму, который гарантирует, что тупиковая ситуация не наступит. Для этого каждый запрос процесса на ресурс проверяется относительно свободных единиц ресурса данного типа в системе.

Если процесс попытается запросить больше, чем в заявке, такой запрос выполнен не будет, это невозможно.

Менеджер ресурсов ищет последовательность процессов, которое может гарантированно завершиться.

|  |  |  |  |
| --- | --- | --- | --- |
| Процессы | Текущие ресурсы | Свободные единицы ресурса | Заявка |
| P1 | 1 |  | 4 |
| P2 | 3 |  | 5 |
| P3 | 5 | 2 | 9 |

Задача менеджера – определить, является ли данное распределение безопасным относительно тупика.

Безопасным относительно тупика называется состояние, когда все процессы могут успешно завершиться (найдена такая последовательность). Если такая последовательность не найдена – небезопасное.

В данной таблице состояние системы является безопасным относительно тупика. Можно увидеть. Процесс 2 сможет завершиться, даже если он запросит 2 единицы ресурса, поскольку они имеются у системы. Больше он запросить не сможет – максимальная потребность 5. Завершившись, он вернет в пул свободных ресурсов занимаемые им единицы ресурсов и в сумме со свободными единицами они смогут удовлетворить 1 процесс. Завершившись, 1 освободит, они вернутся в пул свободных ресурсов системы и в сумме смогут удовлетворить максимальную потребность 3 процесса. Значит существует последовательность процессов, что все смогут завершиться.

Совсем необязательно, что если текущее состояние безопасно, то следующее будет также безопасно относительно тупика. (переход от 1 таблицы). Например, процессу 1 выделили

|  |  |  |  |
| --- | --- | --- | --- |
| Процессы | Текущие ресурсы | Свободные единицы ресурса | Заявка |
| P1 | 2 |  | 4 |
| P2 | 3 |  | 5 |
| P3 | 5 | 1 | 9 |

Здесь нельзя найти последовательность. Небезопасное относительно тупика состояние.

Небезопасное состояние не обязательно приведет к тупику – процессы могут не запросить заявленного ими количества ресурсов.

Формально состояние системы процессов является безопасным относительно тупика, если существует последовательность процессов такая, что:

1. первый процесс в найденной последовательности гарантированно завершится, так как даже если он запросит максимально заявленное количество единиц ресурса, у системы имеется необходимое количество свободных единиц ресурса для удовлетворения его запроса.
2. Второй процесс в найденной последовательности может гарантированно завершиться, если завершился первый процесс и вернул системе все занимаемые ей единицы ресурса, что в сумме со свободными единицами ресурса позволить удовлетворить максимальную потребность этого ресурса и тд.
3. I-й сможет завершиться, если все предыдущие i-1 процессов завершились и сумма ресурсов сможет удовлетворить максимально возможную потребность в единицах ресурса.
4. В результате если все процессы могут завершиться, то система находится в безопасном состоянии

Для выполнения требуются n! проверок. То есть время выполнения алгоритма банкира очень велико. Кроме того, у алгоритма имеются очень серьезные ограничения.

В современных системах процессы выполняются по мере …, а ресурсы выполняются по мере надобности. Все – динамически. Но нельзя исключать существование систем, в которых характеристики процессов хорошо изучены и все детерминировано.

Поскольку алгоритм дейкстры очень затратный, он имеет теоретическое значение

**2. Алгоритм хаббермана**

Одним из самых известных алгоритмов, который рассматривается как развитие алгоритма дейкстры – алгоритм хаббермана

Deadlocks avoidance Haberman’s algorithm

Существует несколько изложений.

Чтобы узнать, каким является текущее состояние системы, работа начинается с некоторого заданного состояния. И алгоритм Д, и Х, имитирует выполнение каждого процесса, выделяя ему максимальное количество запрошенных ресурсов. Если все процессы при этом завершаются, то состояние безопасное.

В алгоритме Х также фиксированное количество процессов N, фиксированное количество типов ресурса M, о каждом типе ресурса известно количество единиц этого ресурса. То есть имеется матрица доступных ресурсов



* Max-avail matrix A=(a1, a2, … am); где ai=|Ri| - количество единиц i-го ресурса.

Заявки могут быть представлены только двумерным массивом.

* Max-claim matrix B=|b11…..b1m| … |bn1…..bnm| = |B1…..Bn|

Bij-max|Rj| максимальное количество ресуса Rj, заявленное процессом Pi

Матрица распределения

* Allocation matric C

Cij - Количество единиц ресурса Rj, которые выделены ресурсу Pi

1) Для всех k, Bk <= A, то есть процесс не может требовать больше единиц ресурса, чем доступно. Это отражено в векторе А.

2) C <= B – процесс не может пытаться запросить больше ресурсов, чем указано в его заявке

3) сумма (k=1;n) Ck<=A – никогда не выделяется больше ресурса, чем доступно

4) дополнительно для анализа вводится вектор доступных ресурсов D:

D = (d1, … dm)=A-сумма(k=1;n)Ck – разница между имеющимися и уже выделенными единицами ресурса.

5) Матрица запросов E

Need matrix E – возможные запросы

6) Матрица запросов Request matrix F

7) Предварительное состояние. При этом предположим, что удовлетворяются все запросы. Тогда

D<-D-Fi (доступно-запросы)

Ci<-Ci+Fi (распределено + запрошено)

Ei<-Ei-Fi (нужно – запрошено)

Запрос процесса может быть удовлетворен только в том случае, если предыдущее состояние системы безопасно

Алгоритм проверки безопасного состояния.

1. Выбирается незавершившийся процесс Pi такой, что Ei<=D. То есть доступно больше ресурсов, чем нудно процессу Pi. Если такого процесса нет, то переходим на шаг 3, иначе - 2

2. D<-D+Ci. Pi отмечается как завершившийся и переходим на шаг 1

3. Если все процессы отмечены как завершенные, то система находится в безопасном состоянии, иначе – в опасном. Если система находится в небезопасном состоянии, то запрос блокируется и состояние системы сбрасывается следующим образом. Безоп/не безопасный

D<-D+Fi

Ci<-Ci-Fi

Ei<-Ei+Fi

Откат по пункту 7

Дополнительно процесс, помеченный как завершенный на шаге 2, сбрасывается

D<-D-Ci

И повторно Pi помечается как незавершенный

(пример с почты)

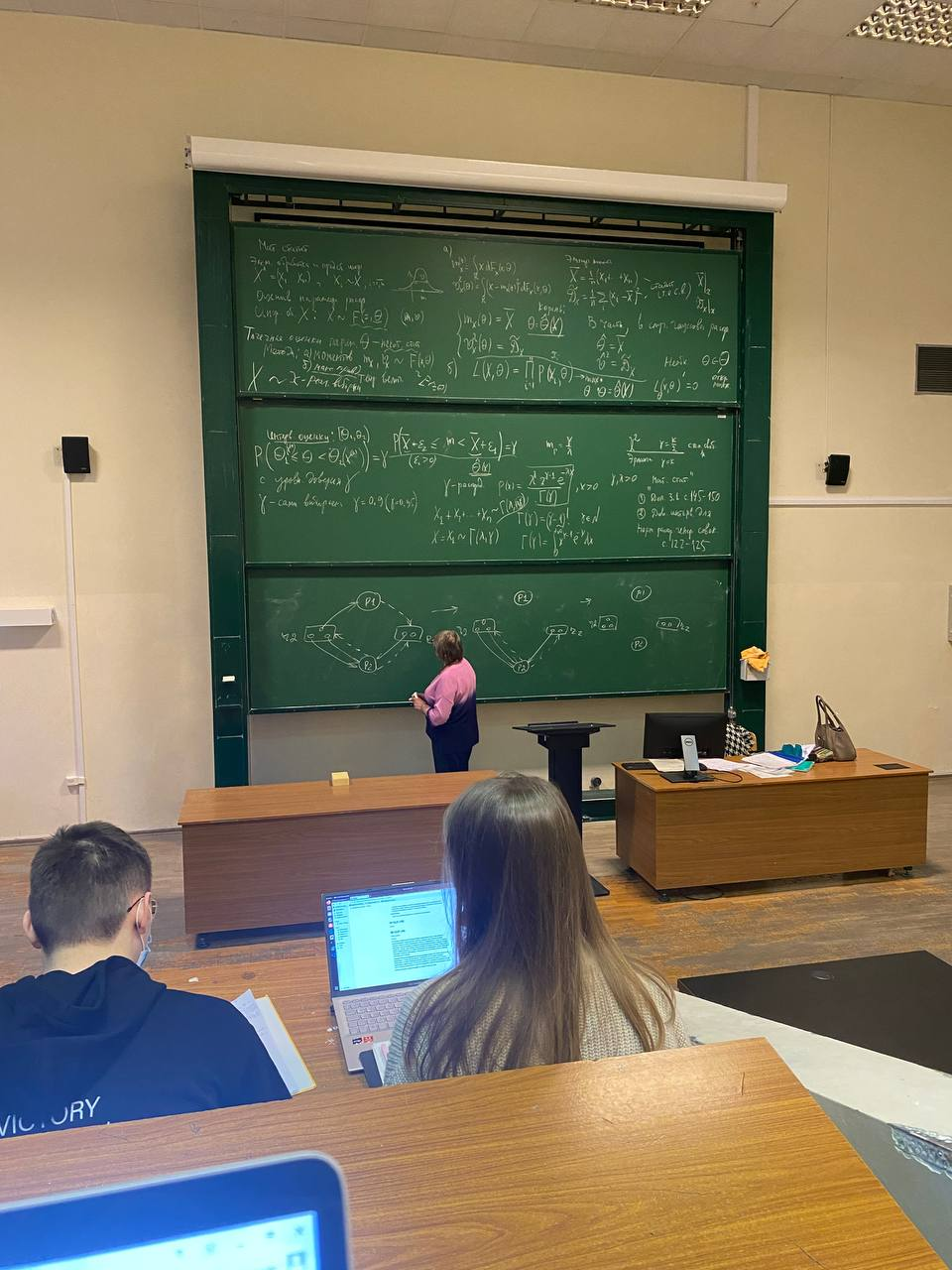
### 3. Обнаружение тупиковых ситуаций

Для этого используется двудольный направленный граф (градовая модель Холда). Имеется 2 непересекающихся подмножества вершин – подмножество вершин процессов и подмножество вершин ресурсов. При этом дуга не может соединять вершины одного подмножества. Граф называется направленным, потому что существует 2 типа дуг – выделение, когда дуга выходит из вершины подмножества ресурсов и входит в вершину подмножества процессов, запрос – когда наоборот.

Граф описывает ситуацию в системе, которая может в какой-то момент возникнуть. При этом эти графы используются для обнаружения тупиков. Фактически такой граф становится необходимым, чтобы убедиться, что система пришла в тупик и причины. Речь идет о большой системе взаимодействующих процессов.

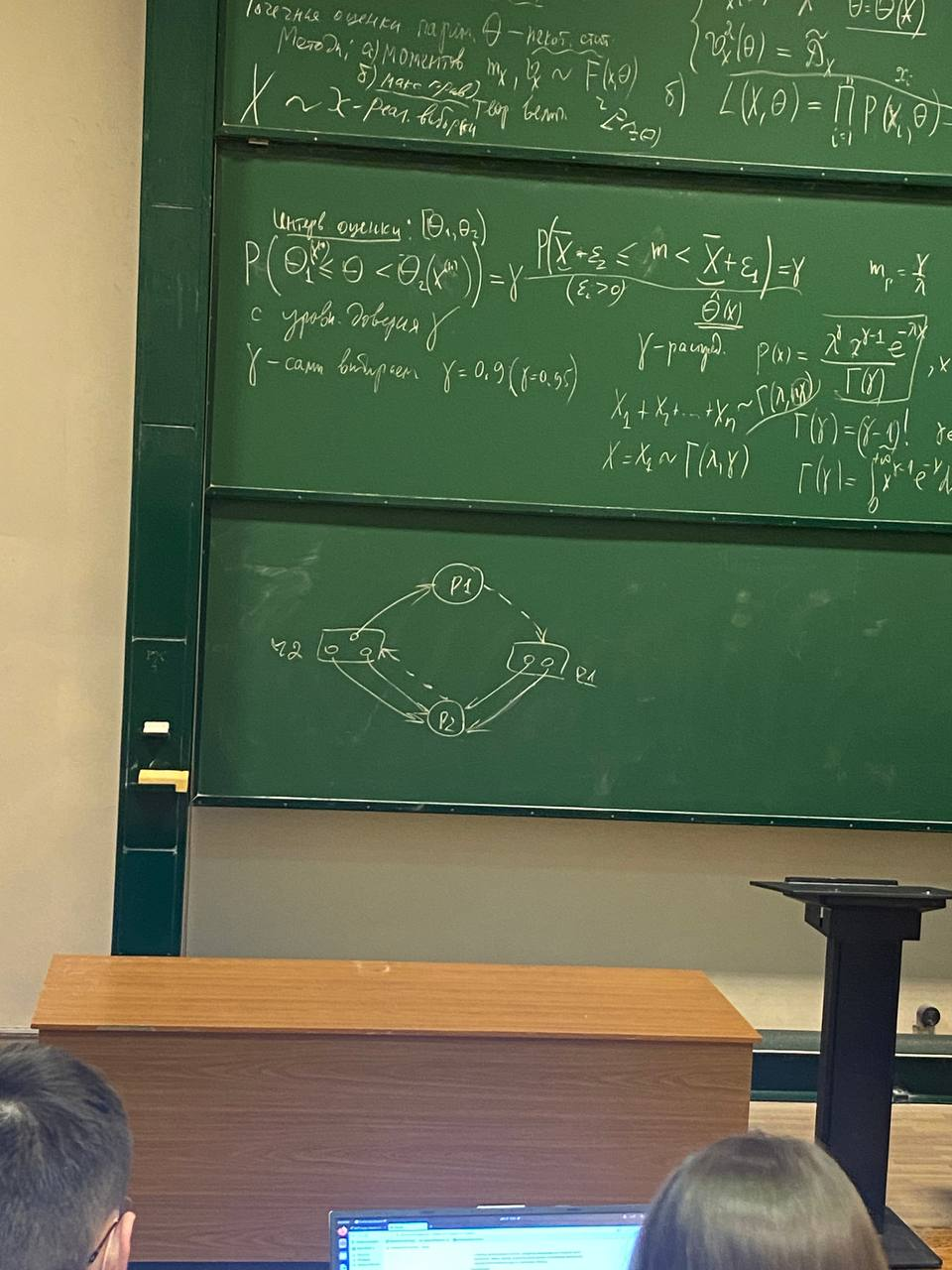
Тупиковая ситуация с использованием ГМХ обнаруживается методом редукции графа. Если запрос процесса мб удовлетворен, то такая дуга мб удалена. В результате того, что запрос удовлетворен, он может освободить ресурсы, которые ранее занял, и эти ресурсы могут удов другие процессы. Если в рез редукции все дуги удаляются и все вершины становятся изолированы, то система не находится в тупике. Если же все дуги удалить невозможно (те дуги, которые и определяют петлю запросов), значит система в тупике.

Пример.



Пробуем редуцировать. Если будет удовлетворен запрос П1, то он сможет завершиться и освободить. И тогда можно будет удовлетворить и П2. В результате получаем изолированные вершины процессов в рассматриваемом графе –> система не находится в тупике.

Пример. Усложним ситуацию. Имеется петля (цикл) запросов



Тупик наступает только в результате запроса, причем запроса, который не может быть сразу удовлетворен.

Очевидно, что для того, чтобы обнаружить, находится ли система в состоянии тупика, необходимо описать ее.

Можно описать 2 матрицами – матрицами запросов и матрицами выделения.

Матрица выделения A

aij – сколько единиц j-го ресурса получил i-й процесс

Матрица запросов B

Bij – сколько единиц j-го ресурса запрашивает i-й процесс

Обычно процессы запрашивает ресурсы по одному. Хотя если взять страницы…. Но для того, чтобы проводить анализ состояния системы кроме указанных матриц надо иметь вектор свободных ресурсов F. Fi – сколько единиц ресурса свободно.

Для каждого процесса есть строка – одномерный массив. Можем сравнивать строки с вектором свободных ресурсов. Наша задача – редуцировать граф. Останутся процессы, которые попали в тупик.

Никаких ограничений на запросы процесса не накладываются (то есть запрашивают и получают), то чтобы обнаружить тупик, может выполняться проверка каждого запроса. Тогда меньше проверок. Или если таймаут истекает.

## Восстановление работоспособности системы.

2 глобальных подхода

1. Последовательно завершаются процессы, попавшие в тупик. У них отбираются ресурсы и другие могут продолжить

2. Завершение других процессов, которые выполняются самостоятельно, не имеют отношения к тупику

Либо kill, либо откат до возникновения запроса. Для этого надо хранить состояния.

## Тупики в распределенных системах

В распределенных системах процессы могут обмениваться только сообщениями. 1 тип – известны пот, неизвестны пр, 2 – известны пр, неизвестны пот, 3 – никто неизвестен

Особенности:

* Задержка передачи данных в сетях затрудняет точную оценку состояния процессов в распределенной системе. В результате переход в тупиковое состояние мб не сразу замечен или, наоборот, мб принято решение, что система находится в тупике, хотя она продолжает оставаться работоспособной, то есть в тупике не находится

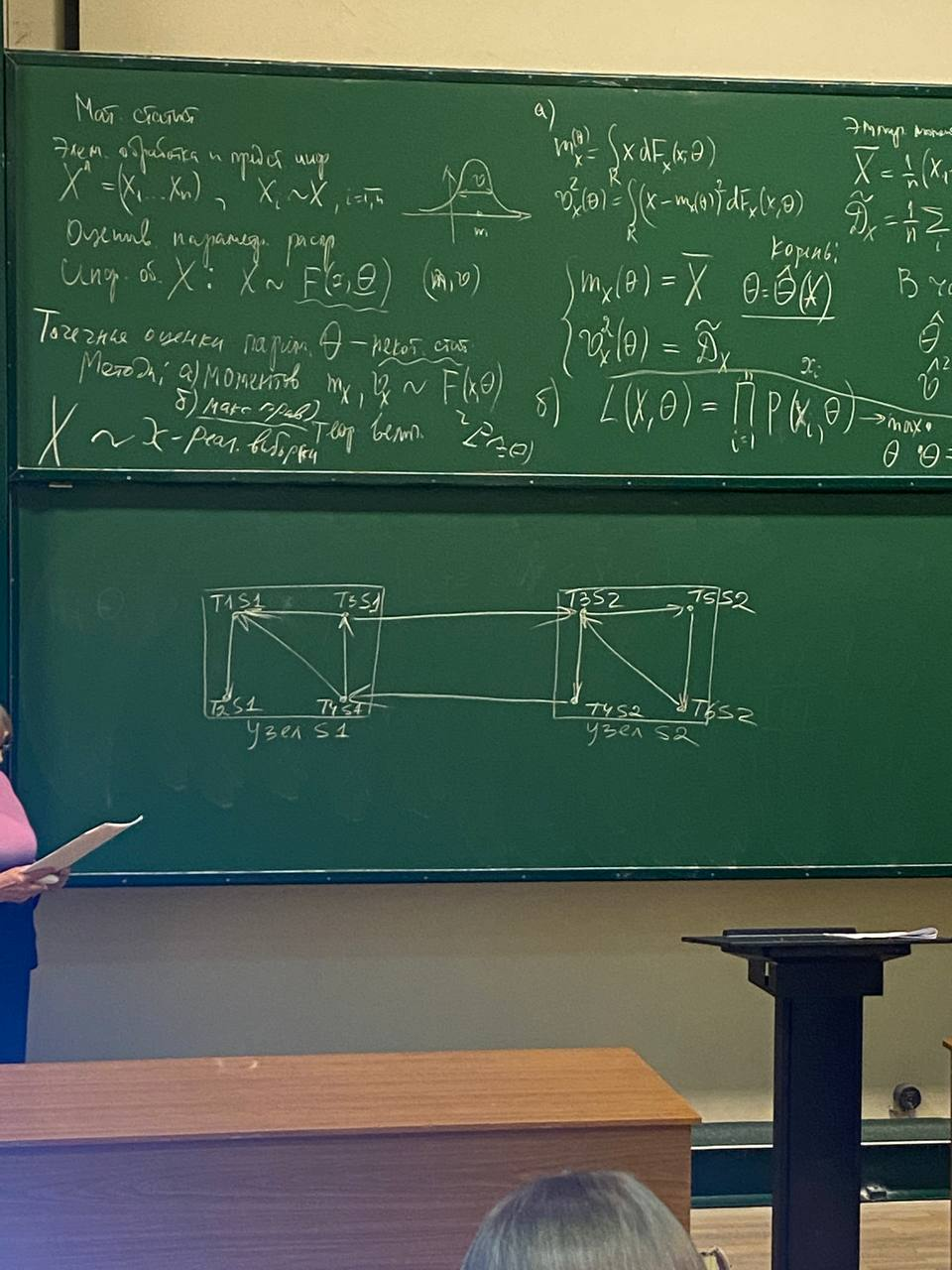
В РС задача обнаружения и недопущения тупиков сводится к известным методам, которые описаны для повторно используемых ресурсов, а именно – 3 подхода – предотвращение тупиковых ситуаций, обход, обнаружение и устранение

Особенностью рассуждения является понятие блокировка.

#### Тупики при транзакциях

Транзакция может находиться в активном состоянии или в состоянии блокировки. Состояние блокировки – необходимый ресурс используется в другой транзакции. Для отображения состояния всех транзакций будем использовать графовую модель, в которой каждый узел графа отображает состояние процесса, при этом каждый узел обозначим парой имен. Именем транзакции и именем узла. Имена узлов должны быть уникальными.

(граф ожиданий)



*1. Проведем направленную дугу из точки TiSk в точку TjSk, если транзакция Ти … в блокирована в ожидании, когда транзакция ТЖИ освободит ресурс, который нужен транзакции ТИ*

*Будем говорить, что узел ТИСК ожидает ресурс от узла ТЖСК*

*2. Проведем направленную дугу из ТКСИ в ТКСЖ, если транзакция ТК в узле ЫЖ блокирована в ожидании осо. Будем го*

Необходимым и достаточным условием возникновения тупика является наличие цикла в графе.

Существуют 3 подхода.

При этом предотвращение (недопущение) связано с предварительным получением всех блокировок. То есть транзакция получает все блокировки перед началом выполнения. И сохраняет все блок на протяжении всей транзакции. Если другой транзакции требуется занятая блокировка, то она должна дождаться, пока все необходимые ей блокировки не будут доступны.

Обход предполагает, что возможность тупика анализируется до тупика, то есть анализируются блокировки (запрос транзакции). Анализируется, приводит ли ожидание запрошенного узла к тупику.

Коротко: транзакции начинают выполняться и запрашивают элементы данные, которые необходимы. В результате получения, они их блокируют.

Диспетчер блокировок проверяет, доступна ли блокировка. Если да, то диспетчер блокировок выделяет элемент данных, и транзакция получает блокировку. Однако если элемент данных заблокирован другой транзакцией в тн несовместимом режиме, диспетчер блокировок запускает алгоритм, чтобы проверить, приведет ли оставление транзакции в состоянии ожидания к deadlock.

Другими словами, определяется, может ли транзакция ждать, или одна должна быть прервана.

Существует 2 алгоритма

1) wait-die – ожидание смерть

2) wount - wait – ожидание ранение

Первый. Если Т1 старше, чем Т2, то Т1 может подождать. Иначе – если Т1 младше, то Т1 прерывается и перезапускается потом.

Второй. Если Т1 старше Т2, Т2 прерывается и перезапускается позже. Иначе – если Т1 младше Т2, то Т1 может подождать

Обнаружение. Периодически запускается алгоритм обнаружения взаимоблокировок и если обнаружены, то удаляются. Когда транзакция запрашивает блокировку, диспетчер проверяет, доступна ли она. Если да, то Т разрешается заблокировать элемент данных, иначе Т разрешается ждать. То есть при выполнении запросов на блокировки нет никаких мер предосторожности, поэтому некоторые транзакции могут попасть в тупик. Для обнаружения тупиков диспетчер периодически проверяет, есть ли у графа циклы. Wait-forgraph.

Если обнаружен тупик, менеджер блокировок выбирает Т-жертву из каждого цикла. Жертва прерывается и откатывается, что для Т возможно – так как есть журнал фиксации.

Методы выбора Т-жертвы.

1. Прерывают самую молодую Т
2. Выбирают Т с наименьшим количеством элементов данных
3. Выбирают Т, для которой было выполнено наименьшее количество обновлений
4. Выбирают Т с наименьшими затратами на перезапуск
5. Выбирают Т, которая является общей для 2 или более циклов.

Последний подход подходит в первую очередь для систем с низким уровнем Т, в которых требуется быстрый ответ на запросы блокировок.

## Обработка тупиков в распределенных системах

Распределенная БД – БД, которая находится на нескольких сайтах и использует данные с разных сайтов. Это сложнее, так как объем данных распределяется неравномерно между сайтами и время обработки сильно варьируется. В результате одна и та же Т может выполняться (быть активной) на одних сайтах и неактивна для других.

Пример: на сайте 2 конфликтующие Т. Одна из них может находится в неактивном состоянии. Такого в централизованной системе случиться не может. Эта проблема получила название – проблема местоположения транзакции

В этой модели Т несет определенные детали при перемещении с одного сайта на другой. Под деталями, например, понимается список необходимых таблиц, список требуемых сайтов, список посещенных таблиц и сайтов. Список таблиц и сайтов, которые еще нужно посетить. Список блокировок с указанием типов.

После того, как Т заканчивается или фиксацией, или прерыванием, информация должна быть отправлена на все заинтересованные сайты (репликация, согласование данных)

Решение строится на известных методах – недопущение, обход, обнаружение. Недопущение – транзакция должна запросить все блокировки. Для этого должна использоваться централизованная система, то есть сайт, на котором выполняется транзакция определяется как контролирующий. Он отправляет сообщения на сайт, на котором расположены элементы данных для блокировки этих элементов и ждет подтверждения. Если все сайты ответили, что заблокирован, то Т начнет выполняться. Иначе Т должна будет подождать. Недостатки: длительное время задержек связи между сайтами. В результате время между Т увеличивается. Если Т переходит в состояние ожидания, не получив все ответы, это мб долго, причем уже заблокированные все еще блокированы, не дают выполняться другим Т.

Если управляющий сайт потерпел крах, он не может связаться с другими сайтами, которые удерживают блокировки.

Обход