Тупики в распределённых ОС (транзакции)

Тупики при транзакциях (в распр. БД)

Современные вычислительные системы усложняются за счёт распределённости

Транзакции в распр. системе БД вып. на нескольких сайтах и использ. данные на этих сайтах. Объём данных и время выполнения распределяется между сайтами неравномерно. Т.е. транзакция м.б. активной на одних сайтах и неактивной на других.

Если на сайте есть взаимодействующие (конфликтующие) транзакции, то возможна ситуация, когда одна транзакция находится в неактивном состоянии.

Для транзакции важно её местоположение, т.е. где она выполняется (transaction location);

Проблема местоположения транзакции

Решение проблемы: модель Daisy Chain;

Хранение доп. инф. о транзакции:
При перемещении транзакции с одного сайта на другой предлагается хранить спи-
- список необходимых таблиц;
- список требуемых сайтов;
- список таблиц, к которым выполнялись обращения;
- список сайтов, на которых происходило выполнение;

- список таблиц и сайтов, которые м.б. использованы или посещены;
- список блокировок с тегами;

Это большой объём доп. информации.

Самый распространённый протокол:
протокол двухфазной фиксации транзакции;

Но после того, как распределённая транзакция фиксируется/прерывается, вся доп. информация должна быть направлена на все заинтересованные сайты, чтобы проанализировать ситуацию;

Т.е. методы для централизованных систем работают и в распр-ных, но с соотв. дополнениями (хранение доп. информации, которая м.б. проанализирована);

Заключение:

Пример обнаружения тупиков в распр. системе (передача сообщений);

Алгоритм Chandy–Misra–Haas

Процессы затрашивают сразу несколько необходимых им ресурсов ⟹ кол-во транзакций уменьшается;

Если при очередном запросе ресурс занят, то процесс генерирует спец. сообщение и посылает его другим взаимодействующим процессам;
Поля сообщения:

1) номер процесса, отправляющего сообщение; (инициатор)
2) номер текущего процесса (отправитель)
3) номер процесса, получившего сообщение; (получатель)

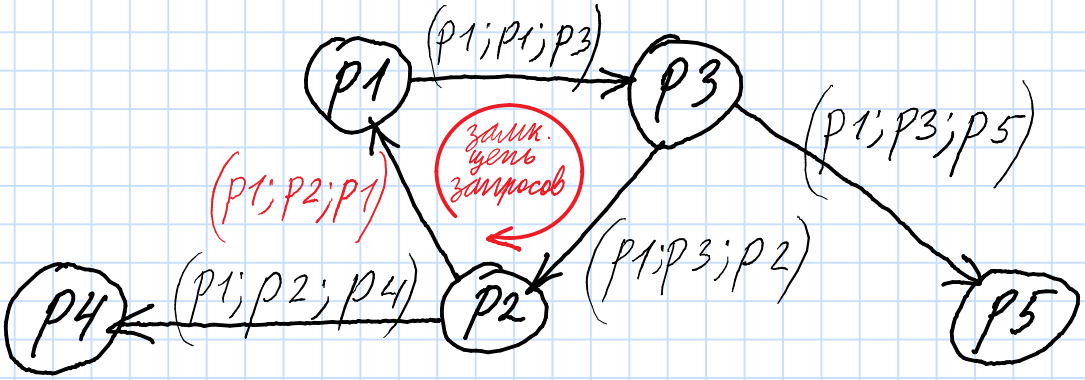
Получив такое сообщение, процесс проверяет, ждёт ли он сам ресурс, занятый др. процессом. Если ждёт, то во 2-е поле он записывает свой номер. В 3-е поле он записывает номер процесса, от которого ждёт освобождения ресурса. Затем он посылает сообщение дальше;

Если процесс получит сообщение и обнаружит свой номер в 1 и 3 полях, то система находится в тупике

Сообщение-зонд (зондирует ситуацию)

Иллюстрация:

инициатор	отправитель	получатель



P1 сформировал зонд; Сообщения - ресурс

А как блокированный процесс может что-то получать/анализировать/отправлять?

На практике это м. б. доп. поток

Всё это – неоднозначные вещи, требующие серьёзной проработки

Совр. системы имеют тенденцию к усложнению;

Всё, с чем мы сейчас сталкиваемся – – распр. системы;

Практическое использование распределённых БД диктуется ограниченными ресурсами систем, и это накладывает свои особенности;

Проблема обнаружения тупиков – одна из основных в распределённых системах;

Эти проблемы не имеют однозначного решения, мы рассмотрели лишь базовые решения, которые м. б. модифицированы в зависимости от того, о какой системе идёт речь;

Управление памятью

Подразумевается оперативная память;

Память как ресурс явл. неоднородной, т.е. может рассм-ся иерархия памяти в зав-ти от близости к процессору;

Совр. процессор - программно-управляемое (не компьютер) устройство;

Ассемблерные команды - фактически интерфейс, т.е. представление пользователю привычных ему средств, но это программно реализовано в процессоре с помощью наноламяти;

Микропрограммное управление

Эти команды делятся вертикально и горизонтально;

Регистры не явл. отдельной памятью

Самая быстродействующая память, с которой работает процессор, - оперативная;

Процессор собственной памяти не имеет и постоянно обменивается информацией с оперативной памятью (считывает команды/данные из ОЗУ и записывает данные в ОЗУ)

нельзя говорить про команды!

ОЗУ ближе всего находится к процессору;

По архитектуре фон Неймана процессор может выполнять только ту программу, которая находится в оперативной памяти.

При этом в память программа загружается в последовательные адреса, и это привело (когда-то очень давно) к возможности исключения из команды адреса след. команды, т.к. он вычисляется при считывании команды;
Т.е. в начале вып-я программы в счётчик команд процессора записывается адрес точки входа, и дальше при каждом считывании, если команды выполняются одна за другой, происходит увеличение содержимого счётчика команд на размер считанной команды;
Т.о. в счётчике команд находится адрес следующей команды, которую нужно считать из памяти;

Следующая по уровню память – вторичная; (дисковая)
Вся вторичная память явл. блочным устр-вом;
А были ещё стримеры (ленточные устр-ва);
Вторичная память явл. энергонезависимой и предназначена для долговременного хранения данных;
Далее будем говорить о дисковой памяти;

У винчестеров есть ещё одна задача: поддержка paging в совр. ОС;

Дисковое адр. пр-во делится на 2 неравные части: бо́льшая - для хранения данных (долговр. хранения обычных файлов);

↳ regular File

Диск не может хранить нелишенованные данные, т.к. должна быть возможность обратиться к ним;

Файл - поименованная совокупность данных;

В UNIX файл идентифицируется i node, а в системе - именем;

ОЗУ строится по принципу близости к процессору;

ОЗУ должно иметь такое же быстродействие, как у процессора, т.к. он постоянно обменивается данными с ОЗУ;

Иначе нет смысла увеличивать быстродействие процессора, если все остальные устр-ва не будут соответствовать ему по быстродействию;

Совр. системах есть ситуация, при которой быстродействие ОЗУ более чем на порядок отстаёт от быстродействия процессора (из-за разных технологий производства) ⟹ необходимость иметь

кеши в самом кристалле процессора;

В Intel 3 уровня кеша
(должны выравнивать ситуацию с отставанием);

Вторичная память тоже претерпела изменения

Адресация в старых винчестерах выполнялась считывающей головкой с вращающегося диска;

В совр. системах часть дискового адр. пр-ва используется для paging (раньше был swapping);

Можно прочитать у Рихтера про файлы, отображаемые в память (т.н. одноуровневая память), т.е. способ, при котором для paging используется адр. пр-во файла;

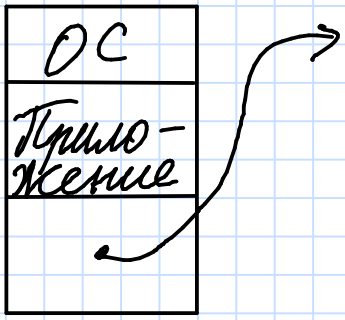
Только основываясь на опыте
можно открыть новые решения

При упр-ии памятью принципы расп. связное и несвязное распределение памяти
Связное — программа в памяти занимает непрерывное адр. пр-во;
Несвязное — программа в памяти может занимать какие-то непрерывные её участки, т.е., грубо говоря, программа хранится в памяти блоками, каждый из которых занимает непр. адр. пр-во;
Классификация:

1) Одиночное непр. распр-е

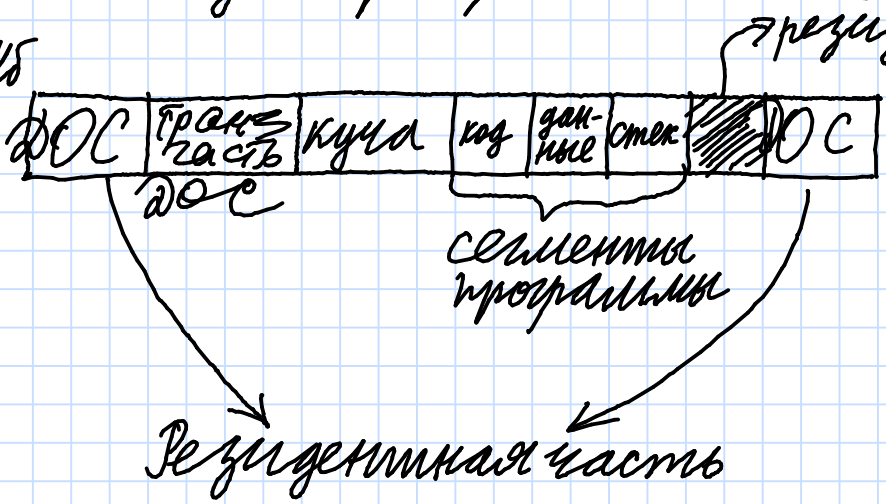
В однопрограммных системах в каждый момент времени в памяти находится одно приложение;

Но на самом деле там 2 программы:



DOS не не очень вписывается в эту концепцию;

DOS - однопрограммная (однозадачная) ОС;



В DOS фактически не одиночное непр. распр, т.к. там было выравнивание сегментов по параграфу;

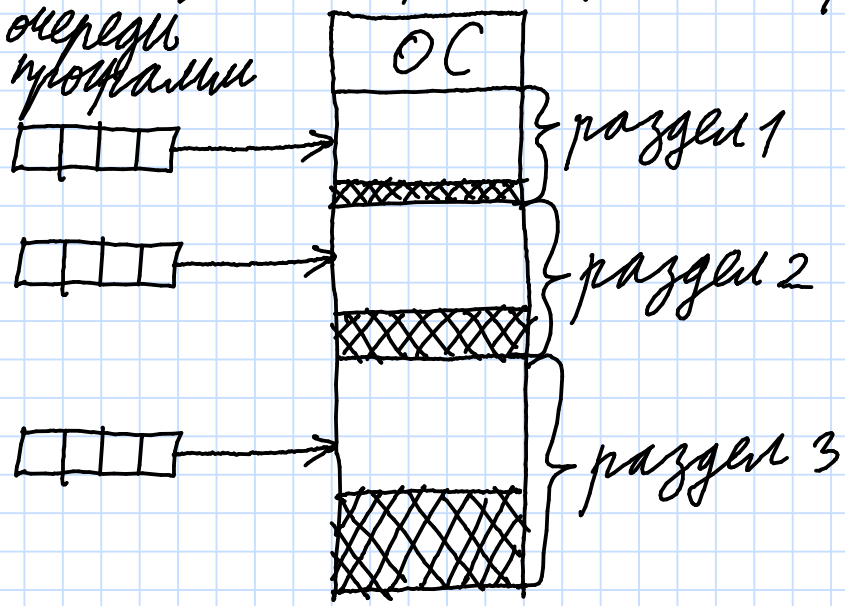
Элементная база совершенствовалась ⟹
⟹ увеличение быстродействия процессора и объёма ОЗУ ⟹
⟹ появилась возможность загрузки в ОЗУ нескольких программ;

Но это всё равно непр. распр-е: каждая программа занимала в памяти непр. адр. пр-во;

⇓

2) Распр-е памяти разделами

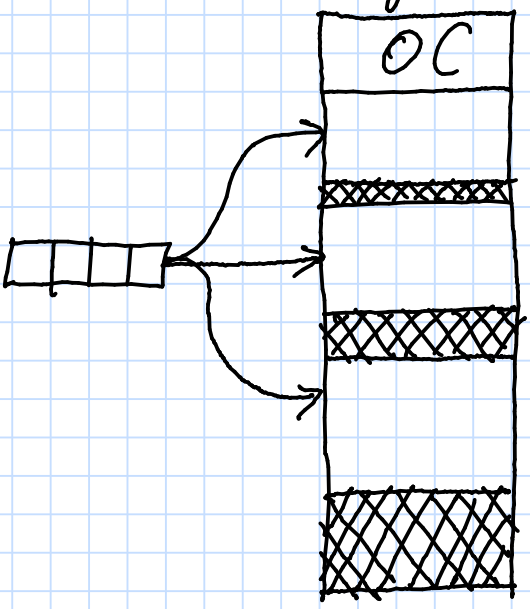
Самое неповоротливое;
До начала работы системы определяются размеры разделов по самым часто встречающимся размерам программ (статически);



Вер-ть найти программу точного размера раздела мала

Крайне неэффективный метод
Когда в памяти м. б. несколько программ, возникают новые задачи распр-я памяти
Программы могут распределяться по очередям неравномерно (например, к большему разделу м.б. меньше всего программ);

Решение – <u>единая очередь задач</u>:

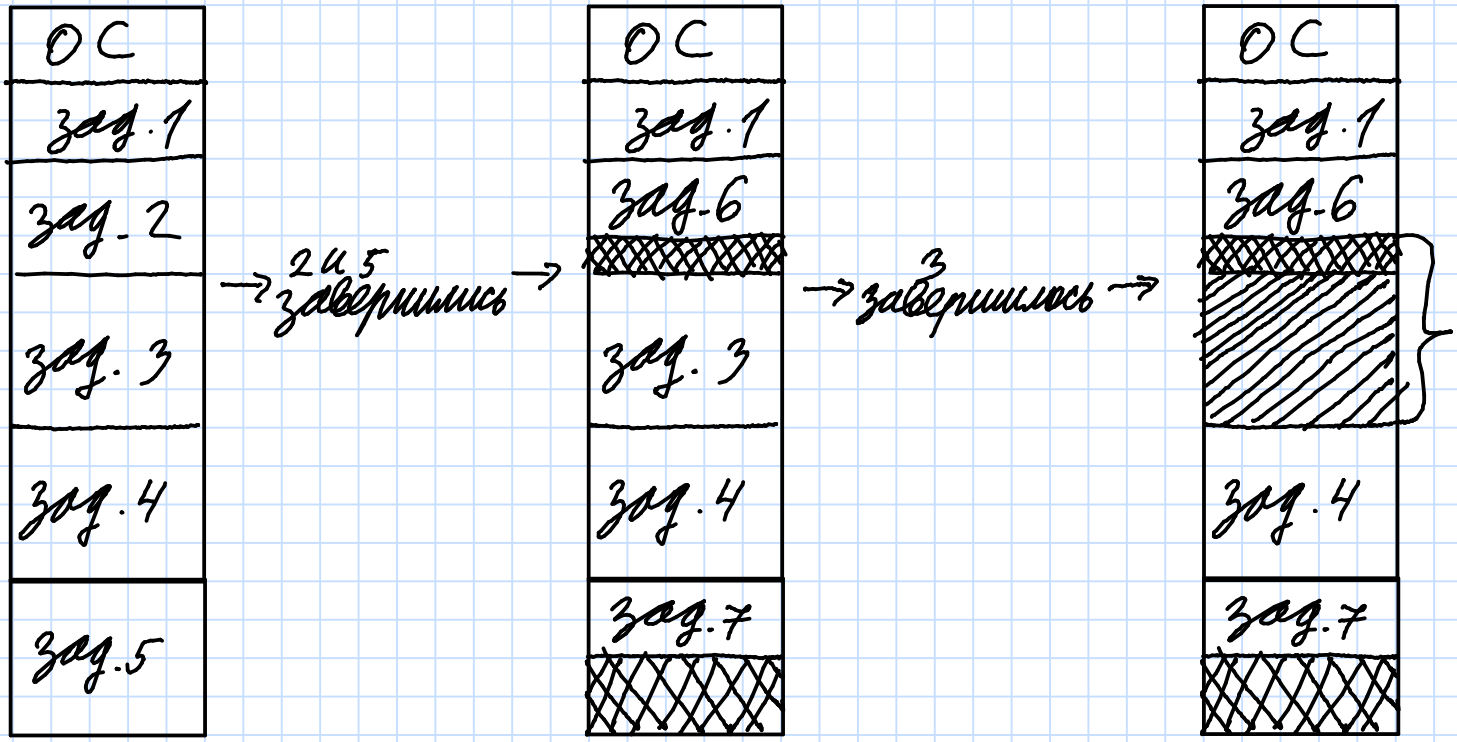


Появляется доп. задача выбора сегмента;
слойсно

следующий шаг:

3) Разделы, размер которых определяется динамически

Т.е. в системе эти размеры не определялись в нач. момент времени вся память свободна, программы могут загружаться в память одна за другой без потери адр. пр-ва, но они имеют св-во завершаться:



Чтобы динамически определять разделы, необходимо как минимум 2 таблицы (распределённых и свободных разделов);

Границы разделов определяются динамически ⇒ мы не можем абсолютно эффективно использовать память, возникнут свободные участки;

+ новая задача перед памятью: объединение свободных участков;

Размеры разделов определяются во время работы системы

⇓

Проблема выбора раздела для загрузки программы.

3 стратегии:

1) первый подходящий раздел; (по размеру)

Пропуск слов – св-во технических текстов;

2) самый тесный раздел: из всех разделов выбирается самый близкий по размеру (минимальное свободное пр-во)

Сортировка разделов по размеру

3) самый широкий: после загрузки программы в разделе может остаться достаточно свободного пр-ва для ещё одной программы;

Разделы динамические ⟹ учёт их размеров ведётся постоянно;

Учёт разделов может вестись с помощью таблиц или связных списков;
Связные списки подходят лучше, но табличный способ проще, рассмотрим его. („потому что потому")

ПВР – таблица выделенных разделов:

Номер	Размер	Адрес	Состояние
1	8К	312К	распр.
2	32К	320К	распр.
3	—	—	нераспр.

ТСО – таблица свободных областей:

Номер	Размер	Адрес	Состояние
1	32К	352	доступен
2	520К	504	доступен
3	—	—	пуст. эл.

Кроме того, разделы м.б. недоступны;

Система должна поддерживать эти таблицы в актуальном состоянии, т.е. регистрировать изменения (новые участки памяти) при объединении своб. участков, изменении своб. участков при загрузке программы;

В результате долгой работы системы возникало (тем не менее,) большое кол-во мелких участков памяти, в которые нельзя загрузить никакую программу;

Это явление наз. фрагментацией памяти;

До 30% памяти не исп. и не м.б. исп. из-за фрагментации памяти;

Решение проблемы – перезагрузка системы;

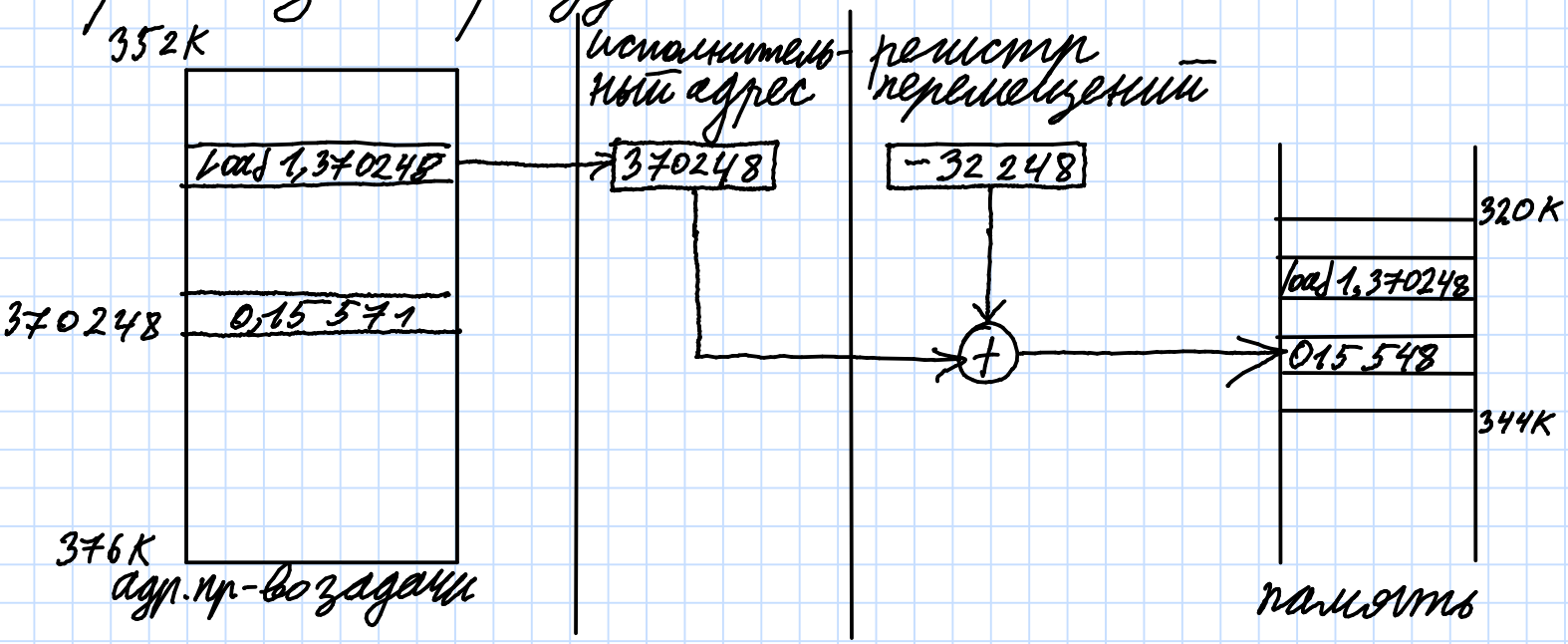
Цена решения – потеря результатов работы программ;

Другое решение — перемещаемые разделы

В рез-те перемещения разделов возникает новое распр-е, т.е. новая свободная память, куда м.б. загружены новые разделы;

Всё в технике имеет свою цену

Перемещение разделов:



Чтобы иметь возможность перемещать разделы, необходимо иметь соотв. аппаратную поддержку;

Это потребовало включения в (систему) процессор спец. регистра перемещений;

Чтобы обратиться к нужному адресу, выполняется преобразование по схеме;

Т.е. перемещение программ потребовало преобразовывать адреса (чтобы иметь возможность обратиться по правильному адресу)

Преобразование адресов — цена данного решения;

Такое преобразование выполняется на каждой команде, а это несколько раз;

Но это решение дало толчок к качественному скачку;

Один из законов диалектики:
переход количества в качество

Люди поняли, что программа на самом деле не связана с нач. адресом из первичного распр-я памяти;

В рез-те было введено понятие логического адр-пр-ва, т.е. любая программа считает, что она начинается с нулевого адреса;

Т.е. мы в наших программах имеем смещение;

Но это приводит к необходимости формирования адреса;