II Семестр.

9.05 Лекция 1

Идеи структурной организации ОС.

Идеологии построения:

1) Монолитные (При увеличении Ф-ций необходимо переписывание всей системы)

2) Многоуровневые (ОС с кольцевой структурой) - иерархические. Недостаток в том что такое ядро тяжеловестно

3) Микроядерная архитектура - функциональная независимость и универсальность системы.
Здесь имеется ядро и функциональные блоки. Ядро здесь компактное и только передает блокам инф-ю по синхронизации и отслеживает обмение пересылки между процессами, а дальше блоки работают самостоятельно.

Структура СПО (ОС)

Если программа меняет инф-цию с которой она работает (помимо своей собственной) то это обрабатывающая программа если не меняет - то управляющая

Компиляторы
Загрузчики
Редакторы связей

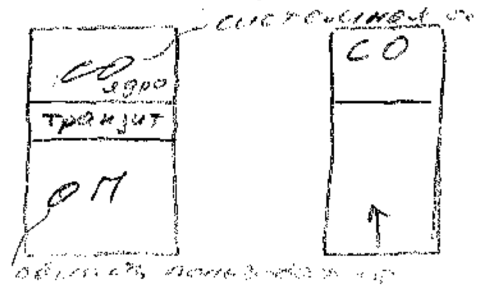
1) Управление
2) Упр. задачами (ядро супер.)
3) Система упр. файлами (9
4) Управл. памя
5) Упр. внешни устройства.

1) Отслеживает вход и выход задачи в системе
2) Отслеж. выполнение собственно задачи и управляет её выполнени
3, 4, 5) Обеспеч. их работу.

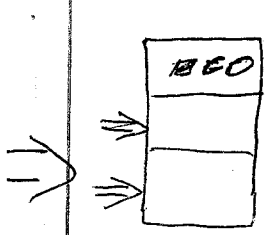
4) Управление памятью
Следует разделить на: - организацию памяти
- управление памятью

Прогр. упр. памятью обесп. эффективное функционирование памяти в соответствии с её организацией. Эта програма решает свои задачи на уровне процесса, то есть память выделяется проце

Память делится на: 1) Зона пользова
2) Зона ОС



Микропрогр. режимы работы

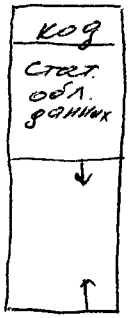


Система запрашивает у драйвера какие разделы

У каждого раздела свой дескр.

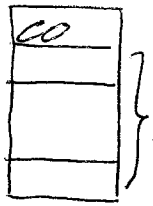
9.02 Лекция 2

Область памяти пользователя



МРТ

При загрузке вводились кол-во разделов и их размеры



При разделении памяти на разделы:
проблема фрагментации

Выделяют 3 вида фрагментации:
- внутренняя
- внешняя
- страничная

Внутр. - неиспользов. область внутри раздела (т.к. одна программа занимает 1 раздел)

Внешняя - на входе есть прог-ма, которая требует памяти меньше, чем совокупность свободн. фрагментов памяти, но загрузить ее нельзя, т.к. она требует непрерывного участка нужного размера.

Страничная (структура организ. памяти) формируется таблица страниц.

Страничная фрагментация - использ. области под структуры для виртуальной организации.

МВТ - построение памяти переменными разделами
Память делится при первой загрузке, потом на освободившемся участке новые задачи.

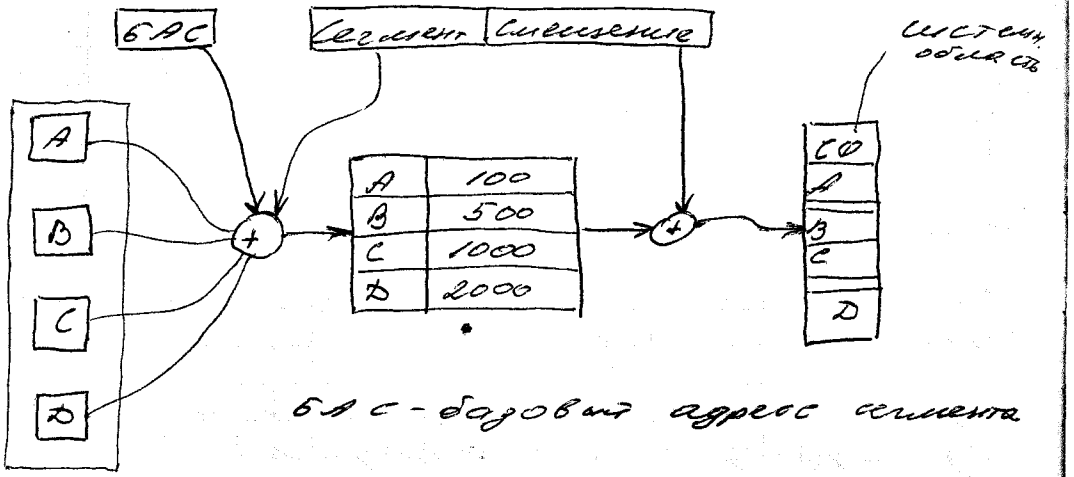
Разделение памяти перемещаемыми разделами.

Недостатки: перестроение адр. констант при перемещении разделов
(динам. настройка адр. констант)

Сегментная организация памяти - модульное прог-ние.
Каждый модуль треб. непрер. обл. памяти.

Возникает проблема защиты. Требуется таблица сегментов: определение место расположения сегментов, указ.

ствуют в формировании исполнительного адреса.



БАС - базовый адрес сегмента прог-а

По степени к страничной организации перешли.

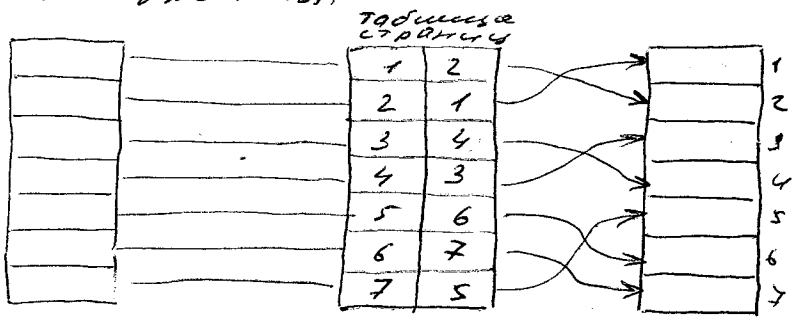
Много прог-м, для каждой страницы => необх. есть одна структура: таблица таблиц сегментов.

Страничная организация подразум. разбиение адр. пространства прог-мы на страницы фиксир. размера и разбиение ОП на фреймы того же размера.

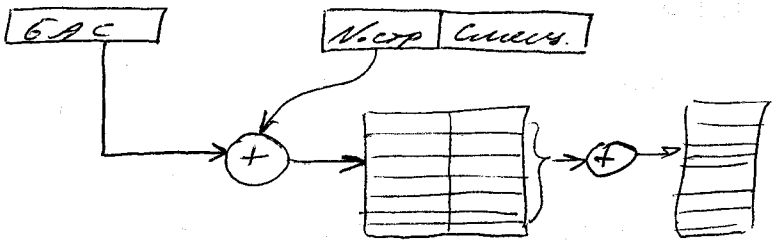
Виды страничной организации:
- чистая стран. орг-ция
- стран. организ. по запросам
- сегментно-страничная виртуальная орг-ция

Чистая страничная организация

Вся прог-ма как в памяти, грузится сразу с использ. разбиения на страницы (фреймы)



Чем меньше размер страницы, тем меньше фрагментация, но сразу растёт таблица страниц.

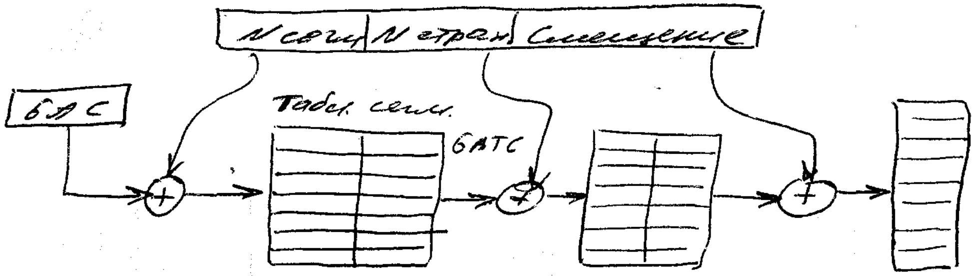


Различие между сегм. и стран. орг. не надо складывать адр. стран. и смещение, достаточно конкатенации.

Таблица страниц = кол-ву страниц в прог-ме. При формиров. исполнит. адреса, если бит присутствия = 0, то прог. разрыв страницы, т.е. подгруж. страницы (по прерыванию)

Сегментно-страничная организация памяти

Программист работает на уровне сегментов.



БАТС - баз. адр. таблицы страниц.

19.05 Лекция 3

Виртуальная память

Работа с программами размер которых физически больше чем объем оперативной памяти.

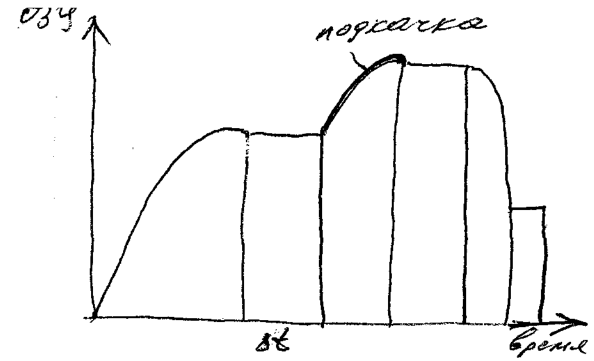
Адр. пространство ограничено размером адресного поля (4Гб для 32-разр. поля).

Совокупность страниц находящихся в кэш-памяти называется рабочей областью

Принцип пространственной и временной локальности - с большой вероятностью можно предположить что следующая команда будет n+1, если сейчас мы вып. команду n. Если мы загрузили участок прог-мы в память то какой-то промежуток Δt он будет выполнятся (временная локальность)

Если мы загрузим определ. число страниц кот. выполнены по свойству врем. и простр. локальности то они образуют рабочую зону.

∃ алгоритмы формирования рабочей зоны

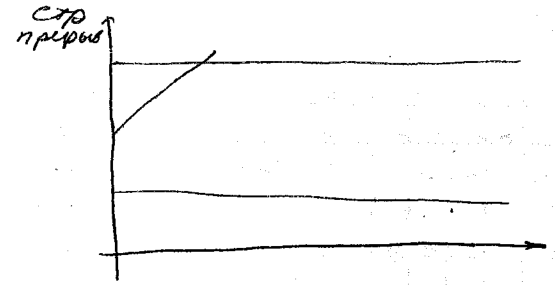


Проблема: ст...
ции локальн...
и глобальн...
размещени...
Локальная -
Если админ...
делим вам ...
делённый об...
ОП - ми...
страниц - это приведёт к удалению страниц при страничных прерываниях (замена страниц же процесса

Глобальная - замещение страниц происходит во всём адр. пространстве (нет мини... для определённого пользователя (процесс...

Модифицированная страница или нет - это же страница (чистые и грязные стр...

Свопированные и не свопированные



Частые страничн...
прерывания - при...
чрезмерной разб...
меняются стран...

Формир. исполн. адреса

Оно идет в аппаратуре (в основном).
То есть происходит аппаратно-программн...

TLB - буфер быстрого преобразования адреса (ассоциативная память)

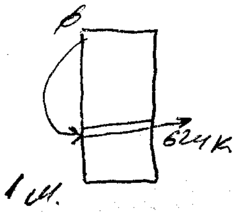
Диспетчер памяти - MMU.

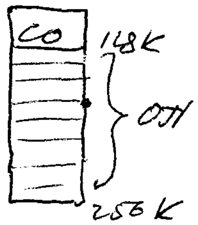
Кэш память управляет MMU. Там же есть кусочек таблицы страниц. Всё происходит при помощи конкатенации.

Ассоц. память - за одно обращение идет...

Выбор исполнит. адреса.

Формируем рабочую зону. Переносим её в кэш память. Здесь исп. диспетчер памяти, кот. из виртуального адреса на аппаратном уровне формирует исполнит. адрес.





Страницы находятся в кэше.

Вир. адр — TLB · диспетчер памяти · номер страницы · конкатенация и формир. исполнит. адреса. если нету ~~прерывания~~ то обращение в кэш, если и там отсутств. страницы то обращение в ОП (если там найдена страница то она копируется в кэш, если и там нет то происходит страничное прерывание).

~~Если каждый процесс 2^{31} степени, а страница $2^{?}$~~

В Pentium исп. двухуровневая организация корневая таблица страниц занимает 4Кб. Из каждой строки идёт ссылка на таблицу страниц. Если на каждую запись исп. 4 байта то таблица страниц занимает 4 Мбайта.

Если ср. размер процесса S

p – размер одной страницы

e – объем записи памяти для каждой страницы

$\frac{S}{p}$ – число страниц для процесса

$\frac{S}{p} e$ – место (объем) для записи таблицы страниц данного процесса

Тогда расход памяти для процесса: $\frac{S}{p} e + \frac{p}{2}$ при страничке пустует.

Оптимальный размер страницы $\left(\frac{Se}{p} + \frac{p}{2}\right)'$

$$\frac{Se}{p} = \frac{p}{2} \qquad p = \sqrt{2Se} \leftarrow$$

глоб. дескр. таблиц

LDT, GDT, LDTR, GDTR, TR, TSS

локальн. дескрипторы таблиц — регистры

Лекция 4

IDT – таблица прерываний.

Дескрипторы сегментов хранятся в LDT и GDT, которые исп. для формирования линейного адреса (для этого исп. селекторы, определяющие смещение в этих таблицах)

Селектор определяется той задачей, котор. выбрана планировщиком для исполнения (переключения)

При активизации процесса создаётся БУП (блок управления)

DT → сегмент → страница → смещение.

Для процесса выделяется память, в к. (в виду того) загружается прог. подчинённая. Расположение программы подчинённой процессу является дескриптором. Она фиксируется либо в LDT либо GDT. Где записана программа в DT определяется в блоке управления процесса. При активизации процесса определяется селектор по которому мы должны выбрать дескриптор сегмента в LDT или GDT.

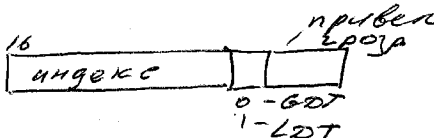
В регистр TR записывается селектор расположения TSS для активизированного процесса. При смене процессов с помощью содержимого TR в TSS записывается содержимое регистра SS, CS, DS. В этих регистрах указаны селекторы этих сегментов при восстановлении задачи.

Место расположения TSS каждой задачи находится в GDT.

При смене задач для виртуальной реализации идёт сброс задач на диск (своп-файл) и возможно 2 варианта:

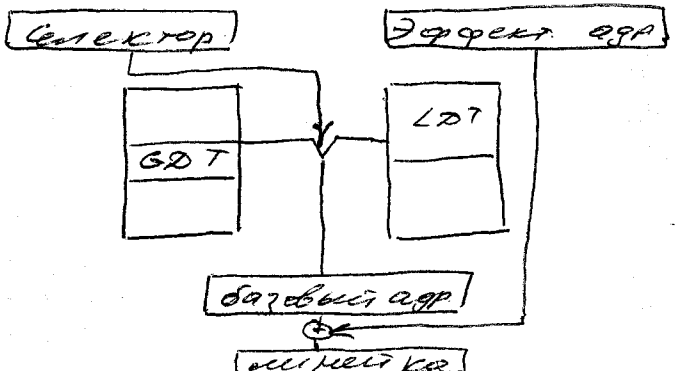
1) для каждой активизированной задачи создаётся своп-файл размером с размер задачи (прораммы)
2) создаётся динамический своп-файл, куда сбрасываются только динамические страницы.

Селектор



GDTR - 48 бит - базовый адресс DT
32 бит - адресс
16 бит - предел

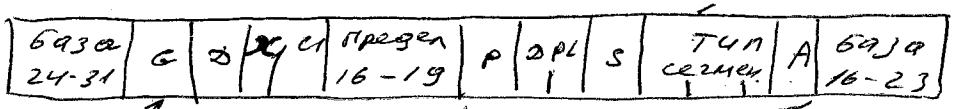
LDTR - 16 бит - там находится селектор
TR - 16 бит



Селектор выбирает либо LDT либо GD
Вытаскиваем дескриптор, из него выта
киваем смещение, которое добавля
к эффективному адресу, получаем ли
нейный адрес.

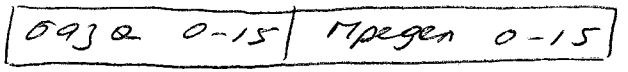
Если указано в селекторе что нужно
читать дескриптор и LDT, то адрес
этой LDT надо предварительно про
читать из GDT.

Формат дескриптора сегмента



бит гранулярности

байт AR права досту



Размер не должен быть больше предела.

G = 1 нужно мерять в страницах и пре
дел определяет кол-во страниц

D - с какой машиной работает (16
либо 32 х разрядной)

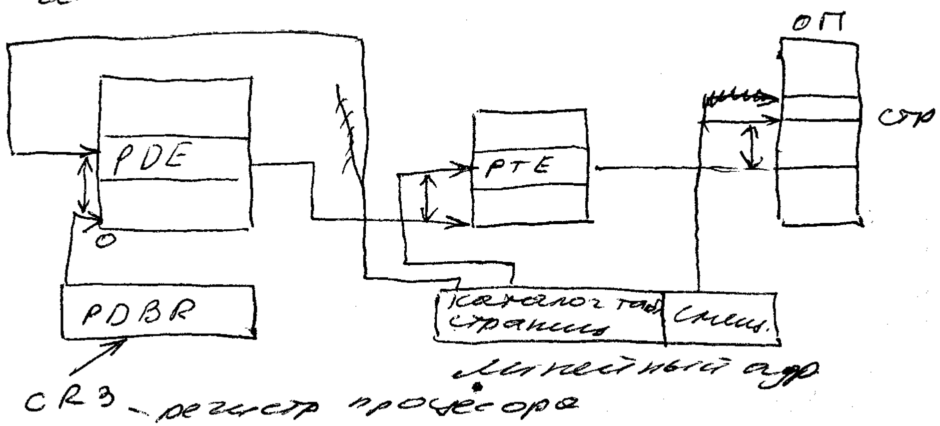
X и-не исп.

P - где находится сегменты (на диск
ими в памяти)

DPL - уровень привелегий
S - тип сегмента (системный или
прикладной (пользовательский))

Тип сегм.
000 - DS - считывать
001 - DC - считывать и грузить.
010 - SS (сегм. стэка) считывать

A - активный или пассивный сегмент.



PTE - page directory entry
PDE - page table entry

Формат таблицы страниц

31 Адр. стр кадра	12 Доступ	0	0	D	A	PCD	PWT	U/S	R/W	P 0

не использ.

Доступ - информация о старении страницы 2 бита (не для ОС)

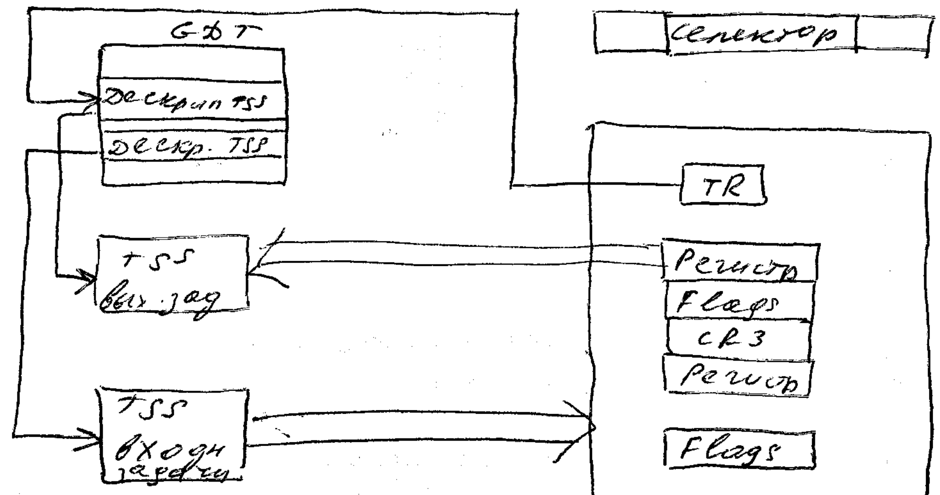
D - чистая либо грязная страница
A - признак активности
PCD, PWT - поле управления кэшированием (сквозная или обратная запись)
~~PWT~~
U/S - user/supervisor (чья страница)

R/W - чтение либо запись
P - бит присутствия (0 - страницы нет)

Лекция 5

Алгоритмы замещения страниц (переключением задач ~~страниц~~)

Ком. перекл. задач



TSS занимает 104 байта (содержит состояние регистров про-ра, регистра флагов…
TR - указатель какая текущая задача (указывает на селектор, кот. указ. где находится TSS) и туда перепис. содерж. регис… поле CR3 находится в середине TSS. При вызове из CR3 содержимое запис. в TSS.

В TSS вся инф-ция которая нужна для восстановления задачи после прерывания. Планировщик формирует селектор, по нему находится адрес входящей задачи.

В TR хранится дескриптор текущей задачи, а в CR3 лежит адрес задачи кот. нужно загрузить.

Обработка страничных прерываний

Страничный демон- прогр. кот. имеет свободные страницы.

1) Задача переключается в режим ядра с сохранением счётчиков команд, флагов и регистров в стеке

2) Запускается программа с сохранением всех регистров (обычно это групповая операция*) и вызывается ОС.

3) ОС ищет вирт. страницу и вычисляет вирт. адрес.

4) Проверяется защита и наличие свободных страничных блоков. Если их нет, то ищется жертва замещения. Если кандидат замещения грязный, то запускается программа сброса жертвы на диск.

5) Если идёт ввод-вывод, то контекст ~~вывода~~ перекл. на другой процесс для завершения ввода-вывода

6) Как только страничный блок очищается ОС ищет нужную стр. на диске и запускает программу перемещения с диска в память (в страницу)

7) Как только нужная страница ушла в памяти обновляется таблица страниц. 8) Прерванная команда возвращается в исх. состояние, а СК корректируется.

9) Вызывается процедура возврата из прерывания

10) Процедура возврата восстанавливает значение всех регистров (переписывает из TSS)

Алгоритмы замещения страниц (ЗС)

При ЗС следует учитывать какая политика ЗС принята в системе (глобальная или локальная (может заместить только страницы данного проц.-а)).

I) Оптимальный алгоритм
Желательно тут предусмотреть через какое время будет обращение к какой странице и удалить страницу к которой больше не будет обращения.
Этот алгоритм не выполним

II) Алгоритм ~~...~~ NRU
Удаляется стр. не использующаяся последнее время. При этом исп. два бита (модификации M и активности R). При любом обращ. к странице в R заис. "1".

Приоритет RM
00 (не акт. не модиф) - всё для
01
10
11

III) Алгоритм FIFO
Простой, но нужно поддерживать спи всех стр., кот. загр-ются в память. Редко используются.

IV) Алгоритм "Вторая попытка".
У прецедентов на удаление анализируется бит R.
Если R=0 то стр. удаляется
Если R=1, то он сбрасывается в "0" и ставится в конец списка, т.е. стр. считается загруженной только что.

V) Алгоритм "Часы".

05 Лекция 6

В "Часах" все стр памяти располагаются в виде кольцевого списка. Стрелочка указывает на очередную позицию списка (на очередную страницу)
Используется бит R.
если R=0 то стр. удаляется.
если R=1 то обнуление и переход на следующую страницу
Здесь указатель не сканирует всю очередь, а просто идёт по кругу.

VI) LRU - удал. страницы кот. дольше всего не исп. Нужно хранить в памяти историю загрузки страниц. Списки надо периодически сортировать по частоте обращений. Но тогда список надо обновлять при каждом обращении к стр.
Проблема - где хранить список. Поэтому этот алгоритм неэффективный, хотя логика у него хорошая.

VII) LRU модифицированный
Делается матрица N×N (число стран[иц])

0 1 2 3 1.

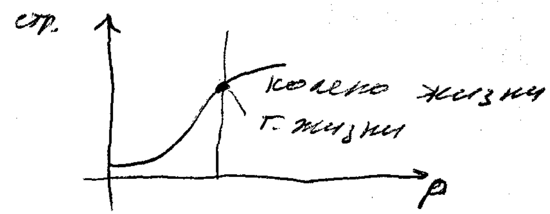
0	0	1	1	1
1	0			
2	0			
3	0			

0	0	1	1
1	0	1	1
	0		
	0		

0	0	0	1
1	0	0	1
1	1	0	1
		0	

0	0	0	0
1	0	0	0
1	1	0	0
1	1	1	0

Если время между соседними страничными прерываниями достигло т. жизни то процесс попал в рабочее множество (т.е. стало минимум кол-во страничн. прерываний) т.к. дальнейшее увел. страниц. не приведёт к улучшению.

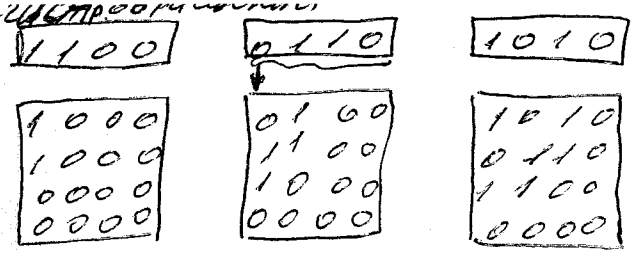


Алгоритм:
1) Вызывается 0-вая страница
Строка 0-вой страницы заполняется 1, а столбец "0"

2) Вызыв. 1-я страница и т.д.

VIII) NFU (модиф. LRU)
Удаляется редко используемая страни[ца].
Идея та же, что LRU.
За таймер накопилось обращени[й] к страницам
Регистр обращ. фиксируется в соотве[тствующей] странице.
Всё содержимое всех регистров здвигается вправо на один разряд.

регистр



IX) Алгоритм рабочий набор

Нужно удалить страницу, не находящуюся в рабочем множестве (наборе). Для этого система для каждого процесса введет характеристику называющую "текущее виртуальное время" (работ. время работы проц-са)

Задаётся время τ или ΔT (фиксированное) и есть время последнего использования процесса.

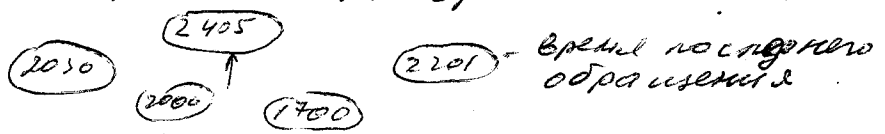
При стран. прерывании анализируется время последнего исп-ния. Если сканирующий стр. имеет $R=1$, то текущее вирт. время становится временем последнего исп. и сбрасывается в "0".

Если $R=0$, то вычисл.: вирт. время минус время последнего использ. если эта разница $> \tau$ то стр. удаляется., если $< \tau$ то помещается в список кандидатов на удаление

X) WSClock

Кольцевой список. В каждой клеточке запит. время последнего обращения

2504 - текущее вирт. время



При каждом тике проверяется страница на кот. указ. стрелка. Если $R=1$ то сбрасывается его в "0", текущее время по стрелке становится текущим виртуальным временем.

Если $R=0$ и разница $> \tau$, то стр. не входит в рабочее множество, а если она чистая то она удаляется

Если разница $< \tau$ и страница грязная то её не трогают так как возможно идет операция ввода-вывода

XI) Алгоритм с двумя стрелками (первой и задней)

Первая стрелка устанавливает $R=0$ страницы за ней, через опред. время идет вторая стрелка, если $R=0$ то стр. помещается в список кандидатов на удаление.

Устанавливается частота сканирования и расстояние между стрелками

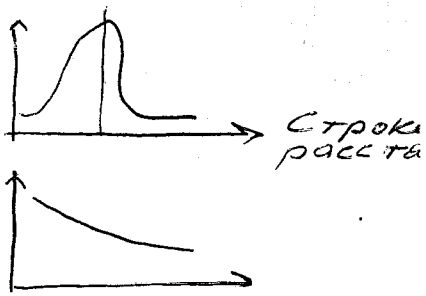
Аномалия Билэди

Стр. вызываются в таком порядке:

0 1 2 3 0 1 4 0 1 2 3 4

0	1	2	3	0	1	4	4	4	2	3	3
-	0	1	2	3	0	1	1	1	4	2	2
-	-	0	1	2	3	0	0	0	1	4	4
P	P	P	P	P	P	P			P	P	

0	1	2	3	3	3	4	0	1	2	3	4
	0	1	2	2	2	3	4	0	1	2	3
		0	1	1	1	3	3	4	0	1	2
			0	0	0	1	2	3	4	0	1
P	P	P	P			P	P	P	P	P	P



05 Лекция 7

XII) Магазинный алгоритм
Иногда исп. для предсказания размера рабочего множества или обоснования, что это рабочее множество есть.

порядок вызова страниц.
0 2 1 3 5 4 6 3 7 4 7 3 3 5 5 3 1 1 1 7 1 3 4 1

0	2	1	3	5	4	6	3	7	4	7	3	3	5	5							
	0	2	1	3	5	4	6	3	7	4	7	7	3	3							
		0	2	1	3	5	4	6	3	3	4	4	7	7							
			0	2	1	3	5	4	6	6	6	6	4	4							
				0	2	1	1	5	5	5	5	5	6	6							
					0	2	2	1	1	1	1	1	1	1							
						0	0	2	2	2	2	2	2	2							
							6	0	0	0	0	0	0	0							
∞	∞	∞	∞	∞	∞	∞	∞	4	0	0	4	2	3	1	5	1	2	6	1	1	4	2

Строка расселения

$F_m = \sum_{k=m+1}^{кол.} + C_m$ (считаем кол-во перевалений числа без сложения, если бы просили выкинуть одну страницу, две стр, три стр. ...)

C_1	4	20
C_2	3	17
C_3	3	14
C_4	3	11
C_5	2	9
C_6	1	8
C_7	0	8
C_8	1	8

$F = C_2 + C_3 + C_8$

⇒

кол-во цифр в строке расселения

Управление памятью в Linux

каждый процесс получает 3 Гб, размер страницы 4К. Используется модель трехуровневая схема линей-ного адреса, либо двухуровневая.
Вирт. пространство делится на 3 сегмента
- текст (код)
- данных
- стэка

Память выделяется на основании дружественного алгоритма, ядро системы пытается выделить непрерывную область стр.
В системе 7 списков непрерывных блоков разных размеров, однако в системе присутствует внутренняя фрагментация !!!!

Иногда исп. ленивый алгоритм. При котором формируется всё один спец. для добавления разрывной области (нужный кусок фрагмента а не целый фрагмент)

Загрузка систем. по требованию. Страницы данных проецируются сегодня сегодня и вспоминаются в раз.
При каждом прогоне увеличивается параметр срочности.
Срочность - кол-во страниц, кот. он должен проверить
Вначале проверяются легкодоступные страницы, потом труднодоступные

1й проход - удал. стр. к кот. не было обращ.
2й проход - совместно используемые, но давней
3й - удаляются стр. одиночных пользователей
Дальше исп. часовой алгоритм с двумя стрелками.

Когерентность

Различают для программиста метод

доступа к данным явные и неявные

Тогда

1) Явное размещение данных - явный доступ (все операции send / receive задаются явно).

2) Неявное разм. - неявный доступ (доступ к данным прозрачный для программиста)

3) Неявное разм. стр. в памяти - явное указание доступа (исп. разделяемое множество страниц на разных устройствах)

4) Явное разм. - неявное указание доступа. Исп. команда Load - store.

Реализация когерентности в однопроц. машинах. Три способа организации памяти:

- с прямым отображением
- частично ассоциативная
- ассоциативная

Для обесп. когерентности необходимо выбрать правила согласования.

1) Со сквозной записью (самая быстрая)
2) С обратной записью (при кэш-промахе или при смене страниц)
3) С буферизацией (когда в буф накопл.)

Сквозная - изменения переносятся вниз, как только они возникают в кешах.

Многопроц. системы

Общение через шину.

Каждый модуль имеет лок. память и общую раздел. память. Подкл. лок памяти через шину.

Исп. алгоритм с обратной и сквозной записью.

Каждая строка в кэш-памяти может иметь 4 состояния:

1) M - строка модифицирована (доступна только в том модуле кот. её имеет в кешах). По R/W доступна только в этом модуле.

2) E - локальная строка копирована. Доступна по R/W в этом модуле и в опер. памяти

3) S - строка множественно копиров. на. По R/W доступна во всех модулях где она скопирована

4) I - строка невозможная к использованию.

Для обесп. когерентности система отслеживает состояние строк памяти. Если строка множ.-копированная и в каком модуле происходит операция записи

Физически распределенная память (MESI DASH)

Оперативная память распределена по узлам выч. системы. Мы можем исп. копии, копии должны поддерживаться.

Там где страница находилась изначально - резедентная страница. А модуль где она находится - резедентный модуль. В других модулях эти стр. являются транзитными

Резедентная страница имеет глобальное состояние, а копированные страницы имеют локальные состоян.

Каждая стр. у владельца имеет признаки глоб. сост. и кажд. стр. имеет список адресов копий. Каждая копия

имеет сок. состояние.

05 Лекция 8

Тупики

Самый простой метод борьбы с тупиками - "метод страуса".
Тупик невозможен, если в системе присутствуют 4 условия возникновения тупика:
1) Условие взаимоисключения (процесс монопольно владеет ресурсом до своего завершения - ресурс неразделяем)
2) Условие ожидания и удерживания (процесс владеет ресурсом и требует новый)
3) Условие неперераспределяемости (ресурс нельзя отнять у процесса и передать другому)
4) Условие кругового ожидания (процесс требует ресурс, захваченный след. по кругу процессором)

Направления исследований по тупикам.

1. Предотвращение тупиков
2. Обход тупиков
3. Обнаружение тупиков
4. Восстановление системы при обнаружении тупиков

Предотвращение: необх. методы, чтобы тупик был невозможен. Обычно убирают одно из условий возникновения тупика.

I способ: исключается второе условие. Процесс запрашивает все ресурсы сразу, но система будет работать неэффективно. Кроме того, программист может неправильно предусмотреть кол-во необх. ресурсов; может возникнуть ситуация бесконечного откладывания.

II способ: исключает первое и третье условие. Если процесс пытается захватить новые ресурсы, а система не может их предоставить, то у этого процесса отбираются все ресурсы.
Недост: можно потерять всю работу, которая была до этого. Опять может возникнуть ситуация бесконечного откладывания.
Решение: можно решать задачу модулями, тогда не потеряем всё.

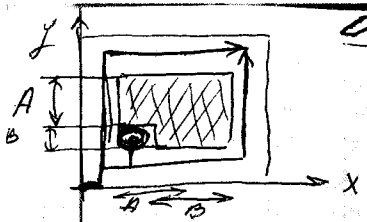
III способ: исключение четвертого условия. Все ресурсы выстраиваются в заранее определенном порядке. Круговое ожидание исключается.
Недост: нельзя переставлять через ресурсы (если нужны 1 и 5, нельзя пропустить 2, 3, 4)

IV способ "алгоритм банкира". Предусматривает: 1) знание кол-ва ресурсов каждого вида 2) знание max. кол-ва ресурсов для каждого процесса 3) сколько ресурсов занимает каждый процесс. 4) в системе можно освобождать и захватывать ресурс по одному 5) понятие надёжного состояния.
Система находится в надёжном состоянии, если при наличии свободных ресурсов хотя бы 1 процесс может завершиться до конца.

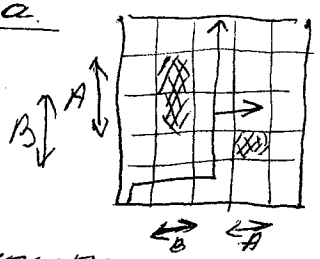
ER общее 10

A	2	4
B	3	4
C	3	10
D	1	2

Обход тупика

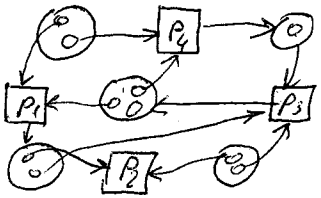


● - тупик
x, y - процессы



Обнаружение тупика

Если система чувствует, что процесс не выходит из тупика по своей эффективности работает Р. Ведется граф состояний процессов и ресурсов



□ - процесс
○ - вид
• - ресурс

□→○ необр. захват ресурса
□←Ⓡ ресурс захвачен

Имеем матричное отображение графа. Имеем любой процесс, кот. может быть завершен и удалены его.

P_2 имеет все ресурсы - удаляем его. Потом удаляем P_1 и т.д. Если граф редуцировался до конца, то мы избежали тупик.

Восстановление системы при обнаружении тупика

Убиваем один из процессов, находящихся в тупике, и забрать у него все ресурсы

В системе исп. контрольные точки откат. Нужно сохранять сост. процессов, восстановление предыдущих действий.

Лекция 9

Межпроцессное взаимодействие

С точки зрения ОС поток - объект выделения ресурса, а поток - объект планирования.

Взаимод процессов возможно при совместном использ кода и данных (когда-нибудь два проц обращаются к одному компилятору). Если мы имеем n-е число пользователей, которые хотят обратиться к диску, то в системе должно быть n-ое кол-во процессов для каждого клиента, а для реализации этого нужен хотя бы один процесс обращения к диску.

Варианты реализации:

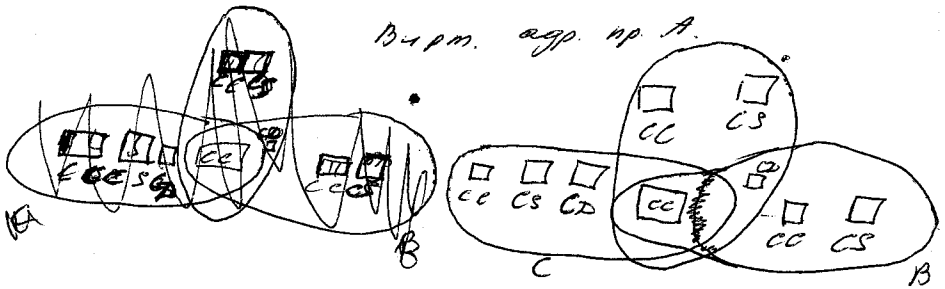
1) Не делать промежуточный процесс. Позвольте пользов. процессу напрямую обратиться к модулю (ВУ) с помощью вызова метода или процедуры. Однако тогда могут быть проблемы при б/з системных данных.

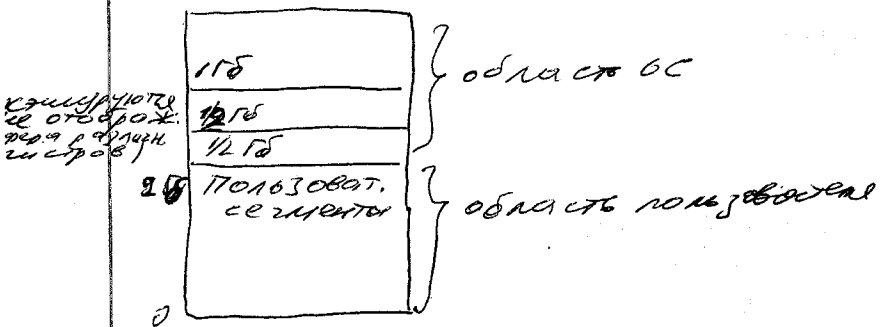
2) Исп. единственный менеджер драйвов, обслуживающий клиентов.
Эти менеджеры как правило многопоточные (для исключ. задержек).

3) Можно связать с каждым модулем набор выделенных процессов.
Можно один процесс поставить на прослушку.

4) Сод. процесс динамически. Процесс стоит на прослушивание, как только поступает заявка он её обрабатывает, а на прослушку ставит другой процесс

3) Процессы с раздельными адресными пространствами
Если сегмент кода только для чтения
то тогда этот сегмент кода может быть
исп. многими процессами как разделяемый. Но
тогда он должен находиться в вирт. адр. простр.
всех данного процесса и отображаться на все
страничные фреймы и блоки физической памяти.

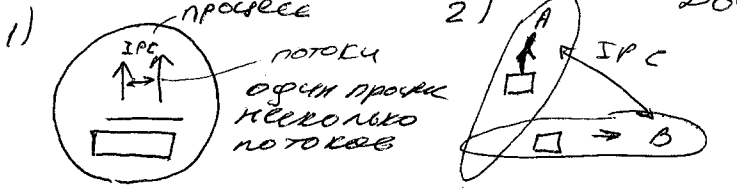




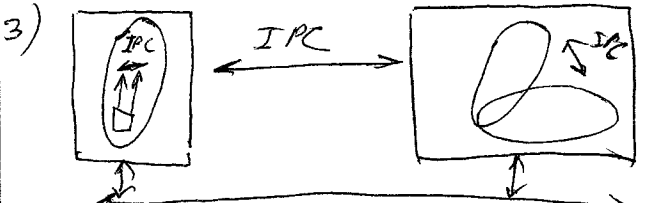
кэшируются не отображ. на физич. стр.

Адр. ОС не кешируется и не отображается

Заявки



Два процесса на одном ком-ре (тут нужно именование процессов)



Тут ещё нужен адрес клиента именование в рамках процессов и машин

Необходимость взаимод. процессов

Взаимод необх. 1) когда нужно передать данные (по запросу или нет)
2) Совместное использ. данных. Чтобы не делать дополнительно нужно (разделить данные в разделяемом адр. простр
создав совм. исп.
3) Извещение один проц сообщ. другому о каком-то событии)

Если процессы взаимодействуют друг с другом при вып. общей задачи или же процесс заказывает сервис у другой задачи и ждет выполнения. то в системе должны быть предусмотрены две операции: - ждать
- Сигнализировать

Типы взаимод. процессов:

1) Один к одному (известно откуда и куда)
2) Любой к одному. Тут нужна система именования
3) Один ко многим (broadcast-вые сообщения) или к одному из многих.
4) Любой ко многим

требующий выдел. один из пр. ист обслуж. только эт. требования

Лекция 10 22.11.05

Средства межпроцессорного обмена:
1) Сигналы
2) Каналы (неименованные)
3) Именованные каналы (FIFO)
4) Сообщения
5) Семафоры

6) Мьютексы
7) Мониторы
8) Счётчики событий
9) Секвенторы
10) Разделяемая память
11) Сокеты

1) Сигналы

Изначально применялись как средство сообщения об ошибках. Это средство представляло собой простейшие команды. Однако они ресурсоёмкие, так как отправка требует системного вызова, а доставка - прерывания процесса получателя.

Они малоинформативны и число их ограничено. То есть они служат для информир. процесса или процессов о наступлении события.

Они могут быть синхр. и асинхр. Поэтому в системе реализован механизм управления сигналами.

Две фазы:

1) Генерация и отправка
2) Доставка и обработка

Время между двумя фазами может быть достаточно большим.

Виды сигналов зависят от того какой процесс генерирует сигнал и кто является получателем.

Причины вызывающие отправку сигнала:

1) особые ситуации (/0, ...)
2) терминальные прерывания (нажат. кнопки)
3) другие процессы
4) системы управления заданиями
5) сигналы для управл. фоновыми и текущими задачами
6) сигналы квоты (если процесс превышает квоту, то ему отпр. сигнал)
7) сигналы уведомления (о наступлении какого-то события)
8) alarm - связанный со счётчиком таймера.

Доставка и обработка сигналов.

Для каждого сигнала предусмотрена обработка по умолч. Однако пользователь может предусмотреть по сигналу завершение процесса.

Процесс может при сигнале:

1) завершить своё выполнение
2) проигнорировать сигнал
3) Остановиться и продолжить потом
4) Отложить обработку сигнала на время

Следует аккуратно относиться к блокированию и игнорированию сигналов. Особенно сигналов искл. ситуаций. Некот. сигналы нельзя обрабатывать самим (kill, stop)

Процесс может обработать сигнал только в том случае, если он активен, что может привести к большим задержкам, при большом кол-ве процессов. То есть сигнал не может быть обработан, если нужный процесс не выбран планировщиком и ему не предоставл. ресурсы.

Сигнал может быть доставлен если только ОС от имени процесса сформирует спец. системный вызов для доставки сигнала. Этот вызов issig() формируется:

- Непосредственно перед возвращением процесса из режима ядра в режим задач после обработки системного вызова.

- Непосредственно перед переходом в состояние сна
- сразу после пробуждения.

Если процедура IS8i8() обнаруживает сигнал, ожидающий доставки, то вдро вызывает функцию самой доставки, вызов по указателю, либо функцию зависящей от системы функции обработки.

Типичные ситуации:

Исп работает с прерыванием и запускает Delete, вызывается аппаратное прерывание, по прерыванию драйвер прерывания определяет что была нажата клавиша и определяет сигнал ждущ. процессу, связанной с прерыванием.

Когда процесс будет выбран планировщиком и запущен на выполн., то при переходе в режим задачи, он обнаружит сигнал и выполнит его; а если он уже был исполнен, то сигнал будет обработан после перехода этого процесса в режим задачи, после выхода из режима прерывания.

Каналы:

Именованные каналы - любой процесс может прочесть информацию или задать канал.

Неименов. - могут быть созданы только между родственными процессами.

Канал имеет вход, выход и буфер.
Проблема каналов: если в канале есть информация (сообщения), то мы должны считывать, пока не освободим канал, обмен считываемой информации определяется пользователем.

Если описатель пишет в канал, а он заполнен, то блокируется (но это не критическая блокировка). Проблемы при освобождении (попытка считать пустой буфер) и заполнении буферов.

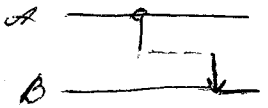
Очереди сообщений или сообщения считаются более информативными в межпроцессорном взаимодействии обеспечивают сов. систему ос и разделяемыми операционными ресурсами.

В системе существует или может существовать несколько очередей сообщений. Поэтому каждая очередь именована. Можно создать мультиплексирование сообщений.

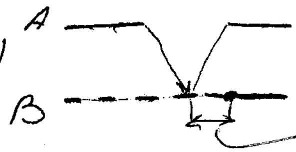
Система поддерживает свои структуры для каждой очереди ввода, связного списка, ук. списка-указателя на сами сообщения. Надо знать о очереди → адрес, размер и тому подобное.
Процессу надо знать: или очереди и идентификатор сообщения.

Лекция 11

Варианты межпроцессного взаимодействия.

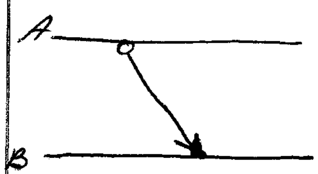


создание асинхронной связи

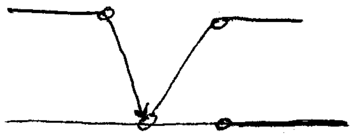
а) 

синхронное сохранное взаимодействие.

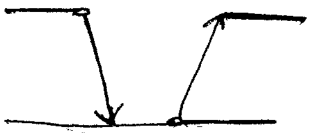
Запуск по таймеру, а не по прерыванию (поэтому задержка

в) 

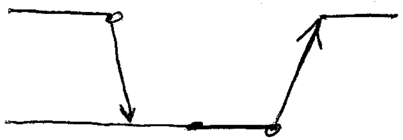
нерезидентная асинхронная связь

г) 

Синхронная связь
Синхронизация по приёму.

д) 

Нерезидентн. синхр. связь. Синхронизация по доставке

е) 

Нерезидентная синхр. связь. Синхронизация по ответу.
Получил, обработал, а затем отправил подтверждение.

MPI_Bsend (б)
MPI-Send (г)
MPI-Ssend (д)
MPI-Sendrecv (е)
MPI-isend - послать и продолжить работу
MPI_recv - прин. сообщ., если его нет продолжить работу
MPI-irecv - принять сообщ, но если его нет не блокировать.

socket - создать новую точку коммутации
bind - назначить сокету лок. адрес
listen - обозначить готовность к соединению
accept - заблокировать вызыв. сторону до приб. с[illegible]
connect - попытка установить соединение
send - read
recieve - write
close - разорвать соединение.

socket - новая точка коммутации

Удалённый вызов процедур (RPC) Ис[illegible] RPC на нижнем уровне.

Управление данными (файловые системы)

Лекция 12

UMA - uniform memory access (время доступа к памяти одинаковое) Для опр. кол-ва процессоров доступ одинаков.

NUMA - not uniform memory access. Время доступа к памяти различное у разных процессоров.

Доступ к памяти UMA и NUMA происходит через чтение/запись.

NORMA - общение между 2 процессами идёт на уровне сообщений.

Файловая система

Файл - именованная совокупность данных, которая с точки зрения пользователя представляет собой единое целое.

Файловая система отлич. от библиотеки функциями (имеет ф-цию оптимизации)

Иерархия данных (логическая)

низ ↓ верх

- бит - неделимая единица инф-ции
- байт (символ) - последовательность бит
- символьный набор
  - внутренний - кодировка внутри машины
  - внешний - общение между машинами
- поле - различаемая ин-ция на уровне программы, это совокупность символов.
- запись - набор полей
  - логическая (объем инф-ции считываемый из файлов за одно обращен.
  - физическая
- файл - совокупность записей
- база данных (значений) - совокупн. файлов

Физическая иерархия данных

- том - объем инф-ции доступный одному ус-ву чтения/записи.
- многотомный файл
- многофайловый том
- запись (физическая) - объем инф-ции считываемой за одно обращение к диску

Размер физ. (больше) и логич. записи не совпадает

Для того чтобы связать физ и лог. запись использ. операцию блокирования (для записи) и разблокирования (для чтения) записи.

ОС для выполн. этих операций должно создать спец. системные объекты для проведения работы с файлами.

Дескр. файлов в сист. области памяти для открытия памяти. В дескр. описан буферный кэш, куда будет считана информация.

Делаются доп. операции (счит. FAT в ОП), если в конфигурации прописано 0, то наз. назначается вторичный буфер - для записи следующего кластера.

При инициализации ядра, когда в файле конфигурац. счит. строка к-ва файлов система выделяет к-во дескрипторов в ядре, соотв. этому числу - к-во файлов, которые можно откр. одновремено

Блокировка - записи в файл нет, есть запись в буфер, сбрасывание кластера на диск происходит при заполнении буфера, или закрытии файла.

Проблема ускорения работы с Ф. - копир. FAT. (проблема согласиров. Ф... -и неправильно выйти, то на диске ...тура файла одна, а реальн... другая.

Способы физ... организа...

- последовательные фай... с физич. носителям... довательный дост...
- произвольный (к... прямой доступ...
...писи фиксиров... атрибуты не рег файл - з... ...зуты записью) ...ово-ть

- индексный файл (индексно-последоват. организация памяти) файл отсортирован либо по значению ключей, либо по значению
- библиотечная оптимизация - иерархическая структура.

Методы доступа

1) базовые методы
2) Методы доступа с очередями. Предусмотр. возможность считывания следующего кластера.

Для работы файловых систем необх. иметь или определить метод доступа, средства управления файлом, средства управления ~~временем~~ внешними устройствами, средства защиты файловой системы.

Система управл. внеш. носит. должна уменьш. число перемещений головки.
Сис. упр. файлом должна расположить записи так чтобы за одно обращение считать всю запись.

1) Связное расположение (непрерывн. запись файла на диск)
2) Не связное расположение

Связн. проявл. фрагментация. Связная делит. на блоки (дружественная система).

Лекция 13

Файловые системы

Именование

8 символов + расширение.
Некот. системы чувствительны к расширениям, некоторые нет.
Некот. файловые системы чувствительны к регистру символов, некот. нет.

Атрибуты

Каждый файл имеет имя и служебную инф-цию. Атрибуты:
- Защита (кто и когда может получить доступ к файлу)
- Пароль
- Создатель
- Имя текущего владельца
- Флаги доступа (скрытые, временные, чтение, запись)
- Длина записи
- Позиция ключа
- Длина ключа
- Время создания файла
- Время последнего изменения
- Размер файла
- Максимально допустимый размер

Операции
- создать
- удалить
- открыть
- закрыть
- читать
- писать
- добавить
- искать (позиционирование)
- получить атрибуты
- установить атрибуты
- переименовать

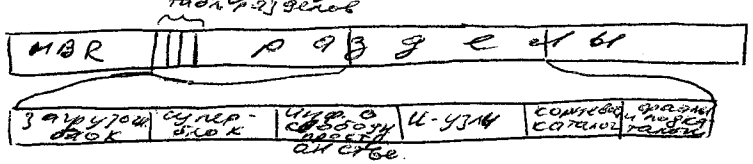
Имя пути.

Если указан полный путь, то это абсолютное имя, если в текущем каталоге то это относительное имя.

Операции с каталогами

- создать
- удалить
- открыть
- закрыть
- прочитать следующую эл-нт в текущем каталоге
- переименовать
- связать (ссылка на один файл из нескольких)
- убрать связь

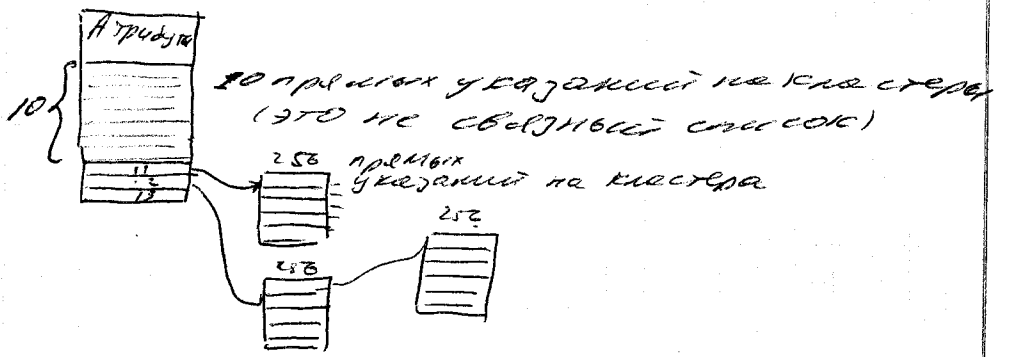
Типовая структура файловой системы



В MBR может быть boot загрузчик
В суперблоке параметры файловой системы
и-узлы = i-node

Реализация файлов

1) Непрерывное расположение
2) Связные списки (каждый блок имеет адрес с продолжением)
Сфера сильного обращения к диску, это плохо. 3) Поэтому список переносят в специальную таблицу (FAT)
Плюс в таблице надо хранить весь диск, и таблица выходит очень объемной.
4) Создание i-узлов. i-узел создаётся только тогда, когда открывается файл, имеется ограниченное количество



Файл может занимать $f = 10 + 250 \times 256 + 256^2 + 256^3$

Реализация каталогов

Каталог содержит дескрипторы файлов, входящих в него.
Атрибуты могут храниться либо в самом дескрипторе (старший вариант), либо в i-узлах

Длинные имена

Если они фиксированные, то они могут быть в самом каталоге (так для них выделено место)
Либо записи описания файла переменной длины, тогда в этой записи должно находиться поле длины записи.
Либо сделать дескриптор файла фиксированной длины, а имена находятся в куче.

Совместно используемые файлы

Важна синхронизация по данным. Проблема с изменением файлов можно создать файл обязательно тогда изменения видны там. В этом файле связывания можно создать счётчик пользователей этого файла

Учет свободных мест

Битовая карта - каждый кластер на диске это бит (1 - свободен, 0 - занят)

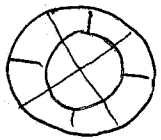
~~связанный список~~

Прямое указание (в FAT таблице признак: своб., занят, испорчен)

Список свободных кластеров.

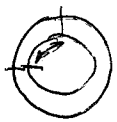
Лекция 14

Диски



От края к центру диск делится на зоны, в каждой из которых уменьшается кол-во секторов

Переход с цилиндра от скорости ω и от кол-ва секторов рассчитается где будет смещение



Согласование файловой системы

В случае двух файловой системы для согласования составляется два вектора (битовых массива)

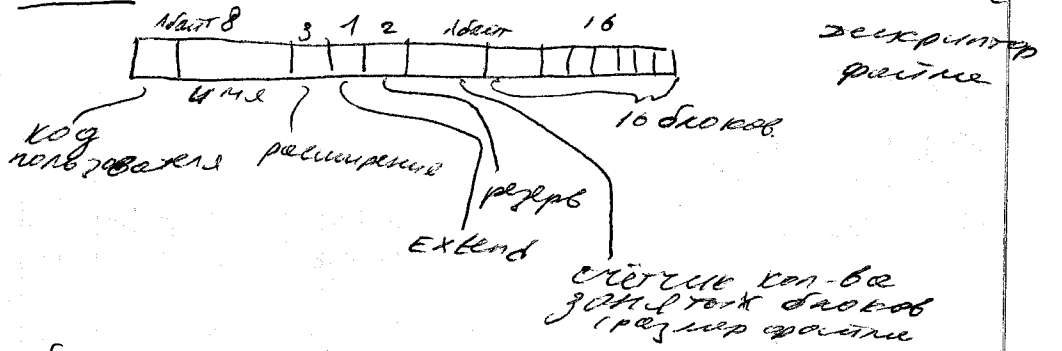
1 0 1 1 1 0 0 0 1 1 1 1 1	для занятых
0 1 0 0 0 1 1 1 0 0 0 0 0 0	для свободных

если кто-то не занял не свободен, то и система считает, что он свободен (как др.)

если для сектора 1 и бит указывает, и бит свободности, то в вектор свободных ставится 0.

если в занятых 2, то значит, что из двух файлов есть обращение к этому сектору, тогда система копирует этот сектор и рассматривает эти два указателя (рассматривает)

CP/M

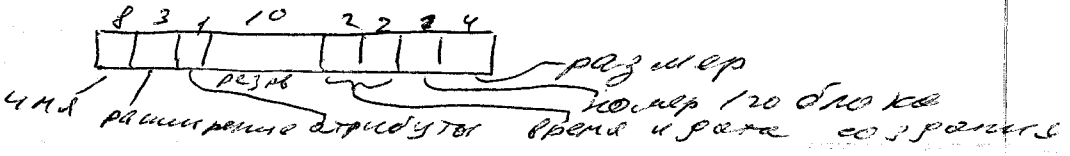


блоки по 1 Кбайту

Здесь реализовано прямое указание блоков. Свободные блоки объединены по битовой карте (в первые принесены)

Если у нас есть продолжение файла (в 16 блоках не вмещ) то в Extend указывается адрес следующей записи в каталоге этого файла.

MS DOS



FAT12

размер диска 64 Мбайта
размер раздела 16 Мбайт

FAT16

раздел 2 Гбайта

FAT32

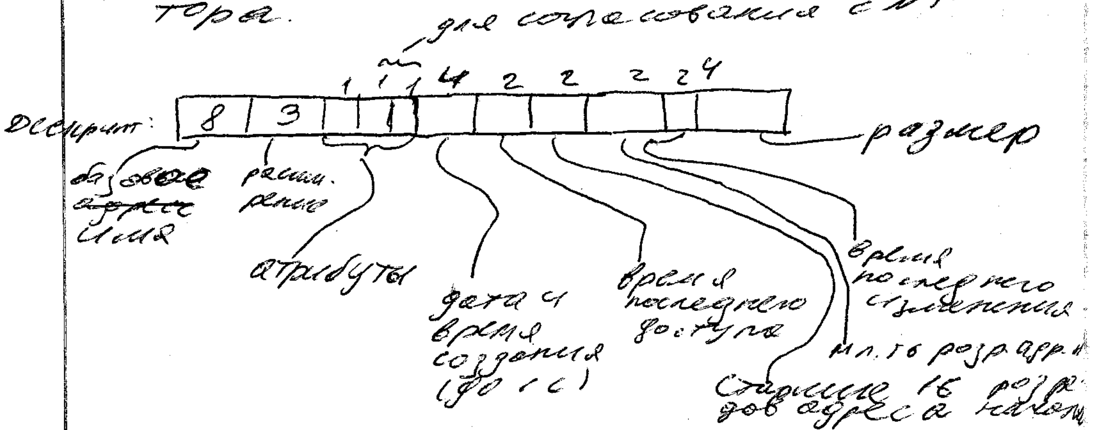
диск 8 Терабайт
раздел 2 Терабайта

В зависимости от размера диска FAT системы кластеризуют от 1 до 64 кластеров (по 32 Кбайта)

Раз в основу Windows 95 положен MSDOS, то исп. ~~FAT32~~ MSDOS структура (в первой версии), а во второй версии исп. FAT32 и длинные имена.

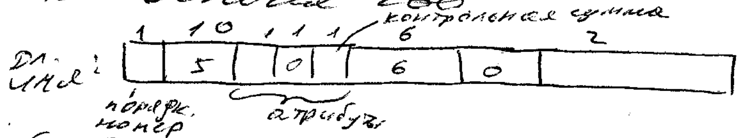
Win 98

Изменили структуру дескриптора.



Длинные имена

(8/3 символ теперь тоже можно сделать). Обычно исп. не более 260



Вместо дескриптора берется 6 символов из длинного имени. Если исп. длинное имя, то в атрибутах ставится недопустимая комбинация и в дескрипторе указыв. начало длинного имени, сот. находится в каталоге как дескриптор.

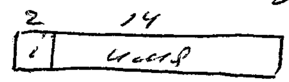
Длинное имя идёт не сначала, а с конца.

В каждой записи длинного имени 13 символов.

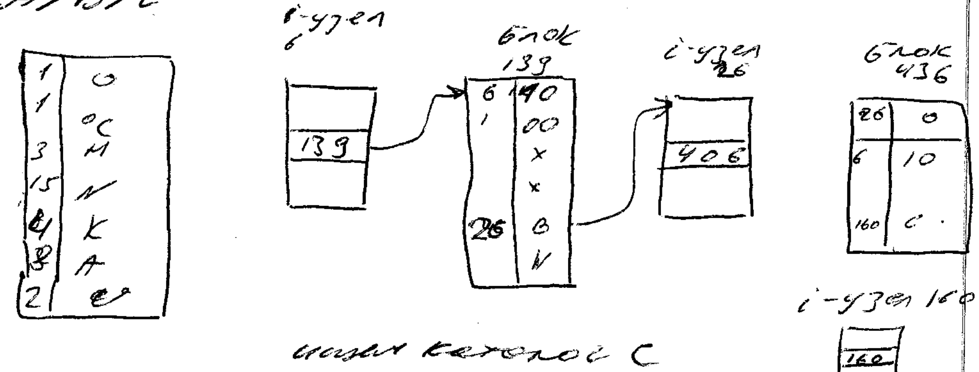
Unix V7

Изначально исп. имена до 14 символов, потом до 255 символов (размер дескриптора файла переменный).

Каталог имеет структуру: 14 байт имя, 2 байта - указатель на i-тый узел.



A/B/C



поиск каталога C

Лекция 15.

Оптимизация доступа к диску

1) FCFS (первый пришел, первый обслуживается)
2) SSTF (с наименьшим временем поиска).

3) SCAN - сканирующий алгоритм. Головка движется вперед по дорожке обслуживает все запросы.

4) C-SCAN - запросы обрабатываются от внешней дорожки к внутренней до конца, потом переход на внешнюю дорожку, до конца запро

5) N-Step-SCAN - тоже так же, как SCAN, но после начала движения запросы ставятся в отдельную очередь.

6) SLTF - с наименьшим временем ожидания первым. Обрабатываются запросы под головкой в рамках всего цилиндра.

Схема лифтового чтения обрабатывает только одну дорожку.