

Лекция 1

Экзамен: письменная часть (качественные вопросы, задачи)
30-40 мин, ~ 1 час проверка (идём), по итогам допуск
на устную часть (билет), оценка независимо от письма.
Кolloквиум: письменная часть экзамена, не выносятся, если
не сдать; хорошая посещаемость + коллок + 5 за зачёт = автомат

Курс: 1) базовые понятия вычислительных систем; влияние
компонентов системы друг на друга; системный взгляд
на объекты (видеть взаимосвязи и итог работы системы);
интеграция аппаратуры компьютера и ОС. 2) Понятие
ОС - программ. обеспеч., в задачу к-ой входит управление
ресурсами вычисл. сист. (аппаратные объём опер. памяти,
программные - объём хранилища для файловых
систем (реализуется программно! на аппаратуре)).
- Основные подходы (напр., к файловым системам)
- как это организовано на ОС; характеристики,
принципы организации Unix ОС

3) Организация взаимодействия процессов; изучение
инструментов и их использование для разработки
многопроцессорных программ

Ресурсы ЛБ выделены процессу монополично (только 1
процесс может его использовать) или разделяемо
(2 и более процессов)

Вопросы синхронизации доступа разделяемого ресурса
Конкуренция процессов за ресурсы $\Rightarrow$ пути планирования
вычисл. системы (организация порядка)

ОС Unix - первая система, описанная с помощью языка высокого уровня: появление Си - эвристического с т. зр. скорости и потребляемости ресурсов. Простая организация (+ для изучения), в Unix наложилась стандарт на управление процессами. Появились в 70х, Сокеты - изобретение Unix.

Процесс и файл - ключевые понятия в Unix

Внешние устройства - драйверы! (принтер и т.п.)

Унифицированный интерфейс работы

Поток информации - перенос данных в ОП

Скорость процессора $\gg$ доступа к ОП

TCP/IP - семейство протоколов, на осн ког.

строится интернет

TCP - транспортный протокол, гарантирует надёжную доставку интр. с недостоверными канальным. Может жить на всяких структурах,

Неэффект. - потребляет много

IP - протокол, позволяющий использовать уникальные именованные канд. сетевого устройства; сеть, объединяющая локал. устройства



<http://ja-fvac.cs.msu.su/mash/os/> - материалы

mash@cs.msu.su - Машечкин

terg@cs.msu.su - Терехан

Развитие вычисл. техники

Наиб. популяр. методология изуг. — разделение комп. в на некое. группы $\Rightarrow$ классификация.

Подход: разделение на поколения: по доступным технологиям и элементной базе.

Эл. база — все детали, из к-х можно строить.

I поколение компьютеров 40-е-50-е годы

Эл. база: эл.-вакуум. лампы

- для её использ. требуется разбём (есть контакты)
- большая

ENIAC ~ 160 кВт энергопотребления $\otimes$
потребл. эл. энергии

Осн. разработчики: университеты!

Ключ. проведение от больших вычислений для решения задач (напр. ядер. оружие). Компьютеры — ^{создания} военные.

$\otimes$ т.к. эл.-в. лампы сильно нагреваются

ENIAC очень сильно грелся $\Rightarrow$ устройства перегорали $\Rightarrow$ нужно заменять (~ 2000 в месяц)
лампы

$\Rightarrow$ тепловое расширение $\Rightarrow$ контакты расширяются и сужаются

$\Rightarrow$ контактные ошибки (от этого исправлять)

$\Rightarrow$ комп. на эл.-в. лампах были крайне ненадёжны

«Срема» — аналог комп. I поколения

«Урал-1»

Использование: есть система команд, которыми его можно программировать. Progr-ние в кодах машины (двоич. коды, у кажд. своё дело)

1-я программа: ^{узнает, как решить задачу (напр.,}
матрица в адр. физике)^(прикладник), алгоритмизм, инженер-электроник. $\Rightarrow$ тяжелая работа

2-е компьютеры — персональные

Программист по блок-схеме кодирует программу (в ОИ!);
переносит программу на компьютерный носитель
(перфокарта и перфолента). Переносит прог. в корпус
комп. $\Rightarrow$ устр-во ввода-вывода (считыватель
информ. с перфокарт и с перфолент). Перфокарта
(-лента) — внеш. устр-во (можно написать прог, код,
прочитать инфор. с него)

Инженерный пульс — панель с индикаторами (лампочки),
позвол. смотреть содержимое комп. ^{кнопки/индикаторы} или
тумблерные регистры (код-во ^{кнопки/индикаторы} с кодами длины адресов)
(можно на регистре набрать адрес, нажать кнопку
«считать». Можно набрать код и «записать»)

Легко ошибиться при наборе программы.

Инж. пульс — можно посмотреть спец. регистры:
напр., регистр отсчета команд $\Rightarrow$ адрес, на
какой команде произошла ошибка

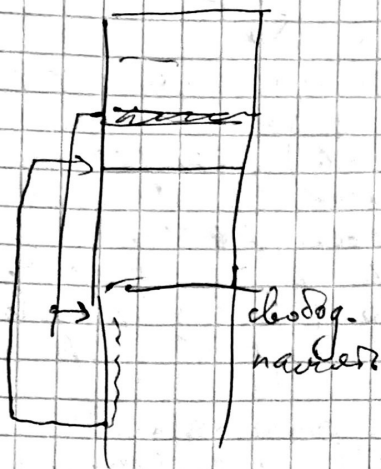
Ошибка устройства: напр., замялась перфокарта
(картон!) $\Rightarrow$ разбор машинки, замена
основки процесса,



Нужно заменить 10 перфокарт
на 6 $\Rightarrow$ вынуть 10, вставить 6 и
добавить 4 без операций

Можно перебить карту (лучше
делать новую!)

2) нужно заметить фрагмент кода на
 большем $\Rightarrow$ вместо команды блок-до-
 свобод. память, где размещ.
 новый фрагмент, после его
 выполнения - в ячейку
 замеченного фрагмента



- много некаждости
- прогр-ные крабне некаждости.
- не технологично (код крабне тяжело
обслуживать и читать)

$\Rightarrow$ задачи: 1) $\uparrow$ технологичность
 2) Борьба с энерговыделением (тепло)
 (эл. база)

- длина проводников мжу частями влияет на
 скорость работы устр-ва

Появление сервисных, управляющих программ:
 выделение части адресн памяти на программы
 чтения и загрузки в ОП

Технологичность: зарождение языков: ассемблер
 и автокод (мнемонич. обозначения команд)
 Необход. программы, переводящие асс. код в
 машинный (изучат. $\neq$ эта прог. - 1 модуль).

Позже понадобилось многомодульное прог. Появл.
 редактора внеш. связей. Объектные модули
 склеиваются в 1 прог. $\Rightarrow$ формирование

Библиотека программы (объект, модуль, ориентир,
на разн. задачи, напр. $\sin$, $\cos$, $\tan$).

Цели языков. высок. ур. — минимизация ошибок
при прогн. и мпр. — $\sim 10^{-6}$ (50-е годы), быстрее пишется,

язык $\rightarrow$ не ориентир. на конкретный комп.

Fortran — до сих пор жив (1 из первых,
достаточно надёжный)

Ассемблер: так эффектив. по себ. привнес прогн.

язык высок. ур. быстрое написание, но прогн.

было менее эффектив.

II поколение

2-я половина 50х — 60х
эпоха

полупроводники: транзисторы

транзисторы меньше эл.-век. ламп, потребляют
энергии меньше (меньше нагрева), впаивались
в платы (нет разъемов, нет контактных оч.)

^{↑ производительность}
существенно повысилась надёжность, снижались
темповые деффекты. Проблема темповых деффектов и
аварий темп. всё ещё значительна

Броили технолог. комплексы для охлаждения
(I и II покол. — вост. воздушные установки;
продув. плат)

более мощные и более функционал. решения

Прогр. обеспеч.

многопользоват. комп (прогр.-ст не
нужен в комп. зале)

пакетная обработка заданий: появилась
сервисная прогн., могла запускать загрузки, комп.

Этот прогн. В кодоу перфокарт добавимся
управляющие: напр. карта запуска компьютера
на Fortran, за ксб - прогн. на Fortran,
Запуск прогн. ~~на~~ ^{на} компилятор, после
выполнения - вернуть управление диспетчеру
(управляющ. прогн.)
пакета. (интерпрет.: 1-ая карта - управляющая,
есть обознач. кода и т.п.)

Функционал. возможности для поддержки
программ. обеспечения. Напр., аппарат прерывания
в случ. возникн. ошибки управление передается
в определенную точку (адрес) управл. прогн.
→ реализация автоматической обработки пакета.
без участия з-ка

Самая дорогая часть - процессор
Внеш. память: устр-во магнитная лента. 1/2 в
большей памяти, надежное, медленное, ~~прямое~~
устр-во прямого доступа (читать только начало)
Во вр. чтения процессор стоит, процесс идет.

Появл. систем мультипрограммирования:
совмещение обмена с внеш. устр-вами
работой процессора → более эффектив.
использование времени и процессора.

Появление и развитие операцион. систем
Появление и развитие виртуализации
Разное управление устройствами (напр. принтеры
сериального и барабанного типа) →
виртуальные устр-ва

Работная система — программа, позволяющая
доступ по интернету графика

Революция в области системного програм. обеспечения

Задачи: управление бизнес-процессами

БЭСМ-6: Большая по объёму ОП (шкафы)

До II покол. (вкл.) комп. — техн. производство, целиком
разработанные одним разработчиком, ^{все детали и набор програм. обеспеч.} специально
под создаваемые комп.

III поколение

кон. 60х — кон. 70х

интегральные схемы (эл. база)

элементарное функциональное устр-во — компоненты

мог реализовать эл. фр-ю

компактность эл. базы

возросла надёжность

интегр. схемы менее энергопотребляющие

повысилась производительность

производство компов упростилось (существовал)

возможность занести аппаратное упрощение

узлов и устр-в: появилось желание разделить

производство частей между разными производителями

⇒ появление стандарта (первый — корпоративный):

станки для станков (куда вставлять), разъемы и т. п.

Появилась семья компьютеров разных произва-

дителей стали выпускать серии компов разных

мощности, производительности и т. п. Признак семьи —

— программная совместимость снизу вверх:

и програм. младшей модели работает на старшей.

Прован в СССР, остались в области производства и сбыта, ориентация не на произв-во своих конкурентоспособных! копиров, а на копирование значащих (не лицензир. покупки, разборка, копиров).

Прогр. обеспеч. умерно.

СИ ЭВМ (семейство малых ЭВМ)

ЭВМ-360

Компания DEC: компьютер PDP-8

Высокоур. машиннонезавис. язык: C
~~(не зависит от конструкции)~~

C предоставляет возможность работы с ОП
Машинно-независ. ассемблер

C: Кратков. списка форм и формат. параметров

Революция прогр. обеспеч.

IV поколение лет 70х - наст. вр.

Большие и сверхбольшие интеграл. схемы
появление микропроцессора (производитель - компания Intel)
1971г.

Altair - 8800 (1974): первый персонал.

Компьютер, мини интерпретатор языка

Бил Аллен и Бил Гейтс создали для этого языка интерпретатор языка Basic. Основана компания Microsoft.

Apple I (1976)

Osborne I: один из первых ноутбуков (1981)

Кардинально новых технологий нет
(2х база)

Активно развивалось параллельное прогн-е

Комп. стал удобным устр-вом:

„дружественный интерфейс“, малый объём,
надёжность, приемлемая энергоёмкость
(проблема всегда была осталась!), цена

Массовое развитие комп. сетей (рынок в
развитие интернета)

Развитие встраиваемых систем (ком. управления
компьютерами)

Высокопроизводительность вычислений
Сегодня суперкомп. — локальная сеть

Лекция 3

19.09

И. база: вакуумн. лампы → диоды и транзисторы →
малые интеграц. схемы → большие интегр. схемы →
микрпроцессоры

„5 поколение“: социализация компьютеров

Основы архитектуры вычисл. систем

Выч. система ≠ компьютер

Выч. сист. — то, из чего создаётся комп.; интеграция
аппаратных и программных средств, функционирующая как
единое целое, направлена на решение конкретных задач
(класс задач).

Иерархия уровней выч. сист.

- (5) Прикладные системы
- (4) Системы программирования
- (3) Управление логич. ресурсами
- (2) „физич.“
- (1) Аппаратные средства ЭВМ

Имеется упр. предоставляется внешнему интерфейсу, позволяющие управлять процессами.

(1) Базовый упр. совокупность физич. ресурсов ^(*) ~~материалов~~ ^(хар-кам) ~~устройства~~

(x) комплект аппаратуры, удовл. условиям:

1) правила программного использования

(т.е. процессор - физич. ресурс, т.к. явл. системой команд; ОП: правила - как мы можем использовать в командах;

внешн. устр-ва: есть интерфейсы, определяющие правила использования; можно подключить

ВУ к унифицированному интерфейсу $\rightarrow$ пользоваться ВУ по правилам этого интерфейса)

2) характеристики ёмкости и производительности (стартовая) частота процессора - временные срезы, за кот. вкл. процессы (?)

3) Степень занятости/используемости

Т.е. Программа хочет воспользоваться ресурсом, а он занят.

Программа ожидает процессор - она занята физич. ресурс

II состояние Обмен данными: пока он не завершён, программа ждёт.

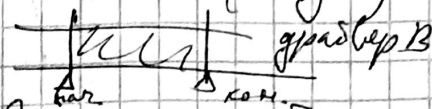
III сост.: Программа готова, но процессор занят $\rightarrow$ прога ждёт.

Программирование на аппарат. упр-те: помнить правила исполн. физич. ресурсов, интерфейсов

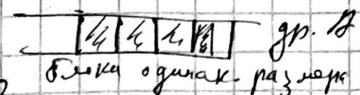
(2) Управление физич. ресурсами.

Унификация и оптимизация правил прог. использования.
Драйвер физич. устр-ва — программа, осн. на ^{правилах} использовании
правил физ. устр-ва и предназначен. для реализации доступа
к устройству (конкретному!)

Затем определяется маркерами начала и конца (годы)
Модель 1: запись произвол. длины



Модель 2: драйвер, предоставляющий возможность
работать с записями фиксир. длины



Воп. Контроль драйвов: удобнее драйвер В

(Контр. местк. диска: удобнее др. А

осн. на использ. ленты (физич. ресурс)

Логика первичной обработки ошибки ЛД реализована

внутри драйвера (аппарат. уровень — ошибки нужно
учит. в программе). Напр., чтение Блэка №N при

контр. местк. диска: если возникла ошибка — запрос
на обмен данными еще раз; N/D см. ⇒ ответ:

❗ рациональная ошибка,
Физ. ресурс — в 1-ую очередь правила использования

Сложно физич. устр., столько правил их использ.
и драйверов — неудобно.

(3) Управл. логик. (виртуальн.) устр-в

Виртуал. уровень — совокупность драйверов
для вирт. устр-в.

Вирт. устр-во — устр-во, некое хар-ки использ-я
кот. реализов. не аппаратно, а программно.

Унификац. использование однотипных устр-в
(такая причина появи. вирт. уровня) ⇒

Унифицир. правила работы с унифицир. диском $\Rightarrow$
эти правила - правила использования драйвера виртуал.
диска.

Запрос не ориентир. на правила исполз. конкретн.
диска, его аппарат. устр-ва

Драйвер транслирует полученный запрос в запрос
конкретному диску

Программа становится независима от того,
с каким диском работаем.

① Унификация доступа — для чего появились
вирт. устр-ва

Реализация виртуал. ресурса = реализация вирт.
драйвера

Нет драйвера — нет устр-ва

Вирт. устр. — для которого есть аппарат. аналог;
объединение однотипных устр-в

МБ присоединён ресурс, не относящийся к
массе устр-в: напр. PDF-принтер/переводчик
в PDF-формате)

PDF-печать: файл в формате картинки

Конвертация в PDF: текстовый файл

Файловая сист. — вирт. ресурс

Обращения за файлом: к драйверу файловой
системы. Ф. сист. размещена на вирт. диске. $\Rightarrow$
цепочка запросов к драйверу вирт. диска $\Rightarrow$
транслирует цепочку запросов к конкретным
устр-вам (напр., диск к драйверу жесткого диска
или ОП)

② схема управл. вирт. ресурсами (Введение 1.ppt)

Группы устр-ва: - аппарат. физич. устр-ва
- есть аппарат. аналоги (аппарат, у кот...)
- нет аппарат. аналогов

Ресурсы выч. сист. - совокупность физич. и вирт. ресурсов.

ОС - комплекс программ, обеспечивает управление ресурсами выч. системы

Организация очередей доступа к ресурсам / планирование доступа. ($\Leftarrow$ (конечность ресурсов $\Rightarrow$ конкуренция за доступ))

(4) Системы программирования.

Жизненный цикл прог. - пом-ть знаний, связ. с разработкой и внедрением прог.

Этапы: 1) Проектирование (что и как делать)

2) Кодирование (как разрабатывать прог. обеспеч.)

3) Тестирование и отладка

4) Ввод прог. сист. в эксплуатацию и сопровождение

1) - Анализ и формализация задач

- Включение всех функциональных требований

- Спецификация

определить аппаратные, инструментальные средства

! методика тестирования и соответствие прог. обеспеч.

2) Можно не всё уместить на 1 этапе; иногда возможен возврат к 1.

3) Подготовка тестового набора, на кот. проверим работу прог.

Тест. покрытие - насколько 1. набор

покрывает функционирование прог.

Тестирование - проверка работоспособности на подготовл. наборе тестов.

Отладка - поиск и исправление ошибок, кот. обнаружены в ходе тестирования и эксплуатации.

4) Документирование (проблема: соответствие между документацией и актуальной прогой)

сопровождение

Прога должна иметь предсказуемое (детерминированное) поведение в разных ситуациях.

При эксплуатации мб разные сбои

При проектировании должна быть учтена возможность сопровождения.

Журналирование информации (логирование) - прога фиксирует точки входа и выхода (логи), по кот. можно провести анализ.

Система программирования - комплекс прог, обеспечивающий технологию автоматизации проектирования, кодирования, тестирования, отладки и сопровождения прог. обеспечения; - комплекс прог, обеспечивающий поддержание жизненного цикла прог в выч. системе

Восп. средства: предыдущ. ур. и программ. средства и комплексы сист. программ-ки, обесп. поддерж. жиз. цикла прог

(5) Прикладная система

— программ. сист., ориентированная на решение или автоматизацию решения задач из конкретной предметной области.

На ранних этапах прикладные системы создавались под каждую задачу $\Rightarrow$ унифицирование $\Rightarrow$ пакет прикладных программ (офисы) $\Rightarrow$ автоматизация бизнес-процессов

Основы компьютерной архитектуры

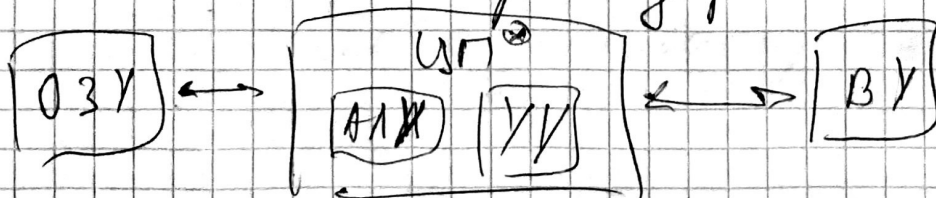
Моч и Эккерс создали ENIAC

дон Кейман оказался в группе с ними, ~~первым~~ написал первый отчет по ENIAC $\Rightarrow$ считается создателем первого языка машины

Принципы построения компа дон Кеймана:

- 1) Двоич. кодирование
- 2) Программное управление (последоват. обработка данных)
- 3) Принцип хранения программы (для хранения команд и данных исполз. един. записи. устройств — оперативное. Команды и данные представляются един. образом. Интерпретация содержимого ячейки (в зависимости от обращения) неавтоматич. образом во время выполнения прог.

Опасность потери управления! (догто)



⊗ во времена р. Н. был только 1 процессор
ОЗУ - оперативное запоминающее устройство
Внеш. запомин. устр-во
может находиться исполняемая программа!

Вот. от ВЗУ

ЦП - обеспечение выполнения программ (порядок исполнения команд)

АЛУ - ариф. логич. устр.: выполнение ариф. операций

ОЗУ состоит из ячеек памяти

Ячейка - элементарная ед. хранения инфы, мб
состоит из ^{независимых} 2^х полей: машинного слова, поле
признаков (тегов) - служебная инфа для обесп.
контроля целостности и корректности использования
данных. Содержимое тэга используется схемой
контроля памяти (размер машин. слова?)

Адрес ячейки - номер для идентификации
Заполнение тэга: при записи машин. слова считывается
целая бита, дополняется до четности (если
 Σ бит нечет, в тэге 1). При чтении считаем
 Σ бит машин. слова, проверяем, соответствует
ли тэгу, если нет - произошла ошибка
(ошибка).

Поле служеб. инфы: контроль - за целостностью данных
- доступ к командам/данным
- доступ к машин. типам
данных

тэгу: команда или данные (раскраска)

Мб раскраска по типам данных. Если 1 из
операндов для выполняемого умножения не того
типа $\Rightarrow$ ошибка прерывания (2 ж 1)
напр.

Производительность ОП - скорость доступа процессора к данным в ОЗУ

Хар-ки: 1) Время доступа - t_{acc} - время между запросом на чтение слова из ОП и получением содержимого этого слова (t_{access})

2) Длительность цикла памяти - t_{cycle} - время между началом текущего и послед. обращения к памяти. (t_{cycle})

$t_{cycle} > t_{access}$ из-за регенерации памяти

Расположение памяти - ОП представляется в виде объединения к независимым работающим устройствам, адреса n -го устройства расположены циклически:

(*) 1-ая в 1-ом устр-ве, $(k+1)$ -ая - в $(k+1)$ -ом, k -я - в 0-ом и т.д. по кругу

Независ. подустройства - банки, $k = 2^L$ банков

Потенциальное ускорение

Лекция 4

26.09

Поле тэга - поле, в ^{ког. автоматич. формируется для контроля} ~~ког. используется процессором~~ и/или устройством ОП

n -разрядное устройство - n -разр. машинное слово
Содержимое тэга зависит от архитектуры и подается к кодовой памяти

Устр-во ОП: после использования необход. привести в канонич. состояние, т.к. после чтения информация теряется $\Rightarrow$ нужно восстановление

Этим занимается аппаратура ОП: сохраняет данные в регистрах и возвращает

По сравнению со скоростью обработки в процессора скорость доступа к ОП мала. $\Rightarrow$ рассеянная память

(*) Младшие L разрядов указывают на адрес банки

Модель рассеянной памяти: единый контроллер

При запросе контроллер выдает L разрядов $\Rightarrow$ найдет нужный банк по этому адресу, после выполнения операции вернет содержимое банка на время

выпуска заблокирован. При обращении к тому же банку нужно ждать, к следующему - нет.

Второй уровень управления - контроллер каждого банка $\Rightarrow$ время доступа $\sim t_{access}$

Без второго ур. нет параллельной обработки банков, $\sim 3 \times t_{access}$

Центральный процессор

Логическая структура:

1) Устр-во управления - обеспечивает последовательное выполнение команд, реализует "быстрые" команды (ког. связаны с передачей управления, услов. безуслов.), обычно логические команды

2) Арифметико-логическое устр-во - реализует команды, требующие обработки операндов; обычно арифм.: разное сложение (разные порядки данных) и т.п.
целое знаковое, с фикс. точкой, с плавающей точкой

Внутренняя память: регистровый и кэш 1-го уровня (кэш мб отсутствует)

3) Регистровая память (р. файл) → Спец. регистры (*)

Регистры общего назначения - программно управляемая память, появились для сглаживания разницы скорости ЦП и доступа к ОП.

Регистровая память работает без процессора - не требует дисбаланса. РОН - для размещения наиболее часто используемых операндов → сокращение времени ⇒ оптимизация.

РОН: - для организации индексации (работа с массивами)
- хранение традиционных операндов (для определенных типов: 32 / 64 / с точкой)
- вектора (вектор машинных слов) ⇒ векторная обработка

РОН в каждый момент времени отражает актуальное состояние программы,

во время работы мультипрограмины: при переключении процессов нужно менять РОН! Но сохранить предыдущее

(*) предназначены для обеспечения корректной работы процессора. Набор спец. р.-ов может варьироваться от модели процессора. Содержимое спец. рег. меняется и контролируется процессором автоматически. В некот. можно записывать инд. от имени ОС.

- счетчик команд (адрес выполняемой или следующей команды)

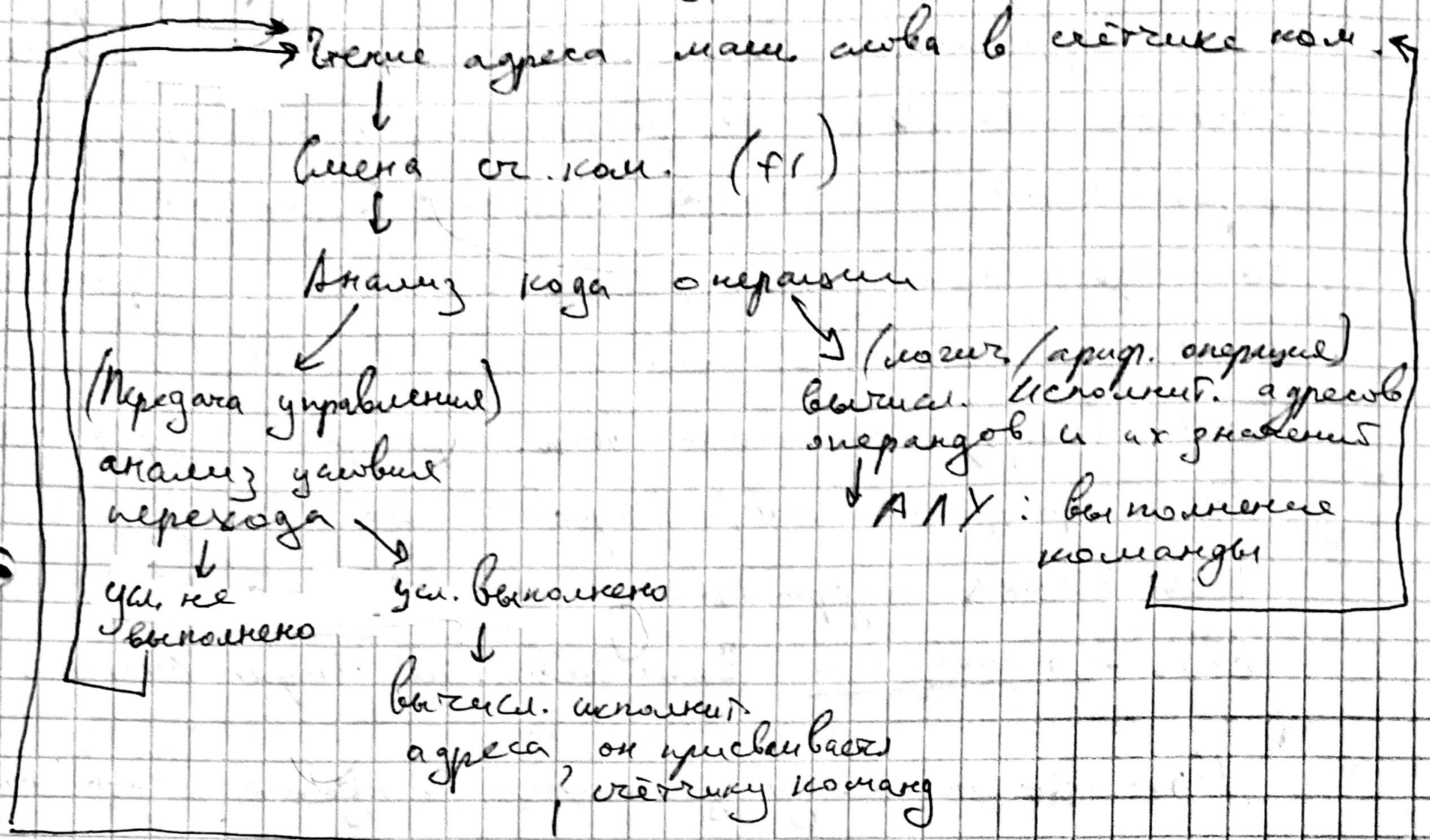
- регистр стека (ОС может записать адрес, кот. будет считаться вершиной стека, но во вр. выполнения программы меняется автоматически)

- буфера м/у ^{ОП и} внутр. устр-вами (обмен данными)

- другие спец. регистры

Схема алгоритма работы процессора:

- 1) система адресации полевая (адрес-^{начало}слова)
- 2) В счётчике команд - текущая команда?



Ошибки всё время!

- должна быть команда, а летят данные (по адресу)
- код операции запрещён
- операция запрещена для выполняемой программы
- переход в несуществующую/недопустимую для прог. память
- попал не на адрес машин. слова (до то середины машин. слова)
- переполнение

Схема ориентирована на положительные результаты. Хотя там еще 43-за обработки ошибок.

4) КЭМ - ^{Высвер из жик. блоков} - находится внутри процессора, реализована так же, как и процессор. КЭМ разделен на блоки фикс. размера. Каждый блок имеет ТЭГ. Взаимодействие КЭМ и ОП осуществляется блоками. В ТЭГ аппаратно-системная игра, отражающая состояние блока: блок свободен или занят; соответствующие содержимому блока области ОЗУ; ...

Игра (напр.): вытеснение Аист., проверка содержимого КЭМ требуемой игрой (проверка по ТЭГам).

Код блока, содержащегося игрой - промах, если - попадание Записывание игры Аистом.

КЭМ почти всегда заполнен $\Rightarrow$ нужно обновление

Промах $\Rightarrow$ нужно найти по Аисту нужные блок в ОП и поместить в КЭМ $\Rightarrow$ обновление содержимого КЭМ - вытеснение. Выбор

блока на вытеснение: случайный / наименее популярный. Что делать с содержимым

выбранного блока? Сильное кэширование:

попадание - модификация блока КЭМ (нужные записи), весь блок сбрасывается (перезаписывается)

в ОП // команда записи

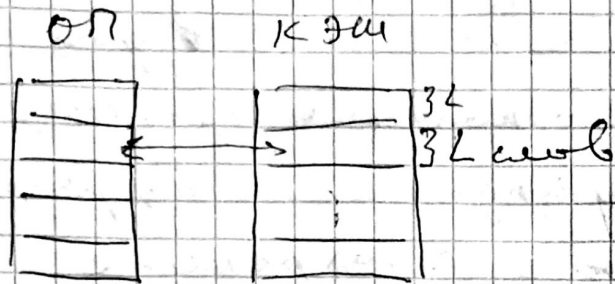
Кэширование с обратной связью: попадание

- модифицирует блок, устанавливается

ТЭГ модификации, переключается в ОП

(если не модиф. - не переключается)

Осн. задача ядра: автоматически сократить обращения к ОП



Аппарат прерываний

Прерывание - аппаратная возможность ядра обрабатывать 1 из нескольких предопределенных событий стандартным образом. Обработка в 2 этапа: 1) автоматически аппаратура выполняет предопр. посыл-то действие по обработке (аппаратная обработка); 2) ядро ОС (ядро в памяти программы, кот. продолжает обработку возникшего события)

Типы событий, вызывающих прерывание:

1) Внутреннее - инициируется схемой контроля работы процессора (деление на 0, переполнение значения конкретного регистра)

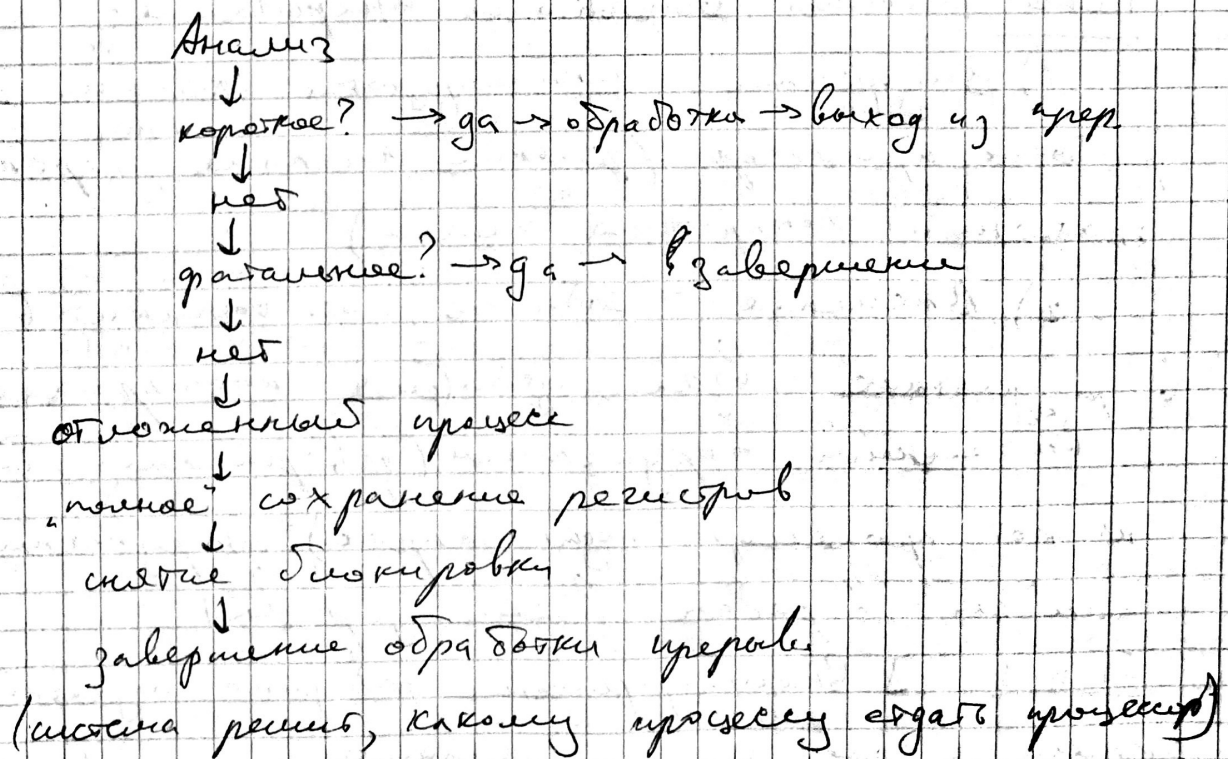
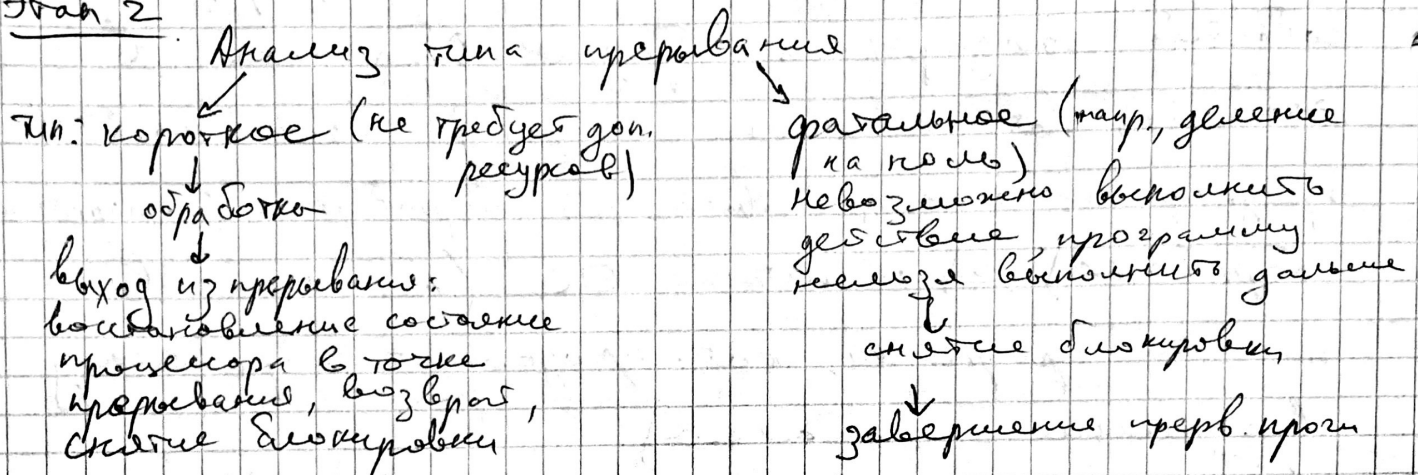
2) Внешнее - событие, кот. возникает в ядре в результате взаимодействия кот. процессора с внешним окружением (завершение обьекта, на кот. процессор подав запрос; обнаружение ошибок в работе внеш. устройства)

Этап 1:] внеш. прерывание $\Rightarrow$ нужно подготовить

исполн. к тому, что надо исполнить прог. и вызвать прог. обработки $\Rightarrow$ завершение текущей команды $\Rightarrow$ $\Rightarrow$ {контекст процесса: нужно сохранить актуальное

состояние текущего процесса } $\Rightarrow$ {сохранение отсчета команд, РОН, состояния процесса} $\Rightarrow$ {при прерывании процессор сбрасывает инд. о состоянии в процесс в регистровый буфер (инд. о контексте процесса); при втором прерывании инд. может потеряться} $\Rightarrow$ $\Rightarrow$ блокировка прерываний $\Rightarrow$ сохранение актуального состояния процессора $\Rightarrow$ переход на предопределенный адрес (к ОС) $\Rightarrow$ программный этап обработки прерывания

Этап 2



Внешние Запоминающие устройства

Типы: / - обмен записями фиксир. размера - блоками
по обмену данными / - обмен записями произвол. размера (магнитная лента)
по доступу / - операциям чтения и записи (жёсткий диск)
↓ - оп-я чтения (CD-ROM, DVD-ROM)

- Последовательного доступа (нужно прочесть все блоки до нужного): магн. лента

- Прямое доступа: магн. диски, барабан

Магн. лента: движение с помощью маркеров (см. Введение 2)

Магн. диск: трата времени на механич. движение

действия Медленного $\Rightarrow$ хочется ускорить $\Rightarrow$

$\Rightarrow$ Магн. барабан: нет траты времени на ввод головки

Модели синхронизации при обмене с внеш. устр.

Синхронная:

Работает процесс $\Rightarrow$ принудительное обращение к ВУ $\Rightarrow$
приостановка процесса, ожидание ответа на запрос
 $\Rightarrow$ завершение обмена с ВУ $\Rightarrow$ возвр. к процессу

Асинхр. (предполагает наличие аппарата прерываний)

Процесс $\Rightarrow$ обращ. к ВУ $\Rightarrow$ процесс ~~не~~ может
продолжаться параллельно с обменом данными $\Rightarrow$
возможна ошибка при обмене $\Rightarrow$ внешнее прерывание

Иерархия устр-в хранения информации

Аппаратная поддержка ОС и систем программирования

Мультипрограмм. режим - возможность организации
переключения выполнения с 1 программы на др.

Пока выполняется одна прога, остальные ждут, когда процессор начнет/продолжит выполнять их команды.

Результат не должен зависеть от порядка выполнения процессов.

Риски: - 1 процесс может зайти в память другого
- управление физическими ресурсами: каждая прога обращается к 1 физ. устр-ву $\Rightarrow$ должно быть ограничение доступа.

Лекция 5

03.10

Аппарат прерываний - возможность аппаратуры помин реагировать на возникновение в вычисл. сист. 1 из набора событий, кот. определяются при разработке ядра.

Прерывание - событие, вызывающее предопредел. посылку действий: должно обеспечить возможность обработки и возвращения $\Rightarrow$ аппаратура обеспечивает корректность: 1) сохраняет хар-ки в спец. внутр. память (хар-ки опис. текущ. точку: счетчик команд, регистры работы процессора, напр. регистр результатов, регистры общего назначения; ROMов много $\Rightarrow$ разработчик определяет, сколько регистров достаточно сохранить и какие скорее всего будут использоваться) 2) наличие состояния блокировки прерываний в процессоре 3) вкл. программный режим

А. прер. позволяет решать много задач: 1) обработка асинхрон, 2) реализация асинхронного обмена

Мультипрогр. режим - в вычисл. сист. обрабатываются 2

и более процесса, кот. могут находиться в 3 состояниях: 1) процессу выделен ресурс процессора ($\rightarrow$ выполняется)

2) ожидание завершения облета, 3) процессы готовы к исполнению, идут, когда ОС дает им ресурс процессора, т.е. факт выполнения

Однопроцессный режим $\rightarrow$ все ошибки - вина программиста.

Мультипрогр. режим $\rightarrow$ необход. обеспечить корректность работы:

1) Защита памяти: от несанкц. доступа других процессов (аппаратное св-во колен) $\Rightarrow$ аппарат защиты памяти

2) Нужно, чтобы для разных процессов были доступны разные команды. Напр., ОС доступны все команды, а процессам не доступны команды управления физич. устройствами $\Rightarrow$ спец. режим ОС

3) Запуск процессора: использует ресурс процессора сам на себя $\Rightarrow$ аппарат прерываний, в т.ч. прерывание по таймеру: передача управления ОС каждый отрезок времени. Справедливое распределение ресурса процессора м/у всеми процессами.

Проблема: использование ОЗУ

Программа должна быть загружена по опред. адресу в буфер программы, ожидающих начала обработки. Прога привязана к диапазону ^{используемых} адресов - плохо, неудобно.

Программы завершаются $\Rightarrow$ освобождается память буфера, но её невозможно использовать $\Rightarrow$ недоступный ресурс (буфер)

Соответствующие адреса, используемые в программе, области ОЗУ, в к-х будет размещена данная прога.

Перемещаемость: если свободна ячейка (фрагмент), в к-х поместится прога $\Rightarrow$ прога перемещается.

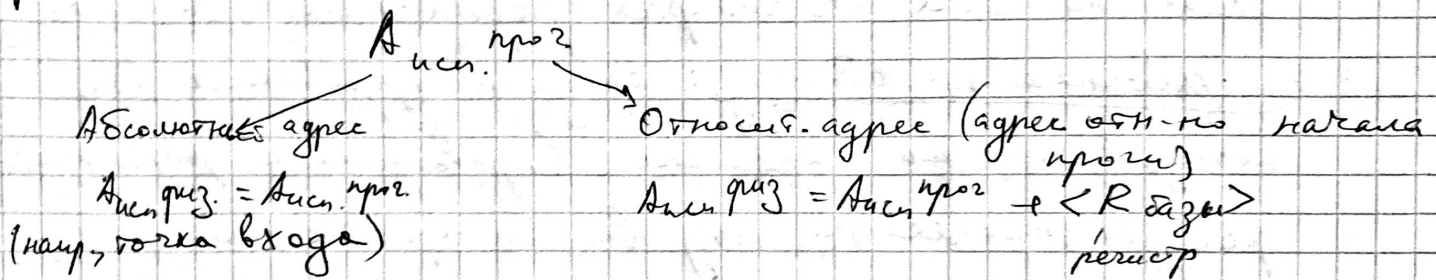
Проблема фрагментации: достаточно места для процесса в сумме, но нет 1 фрагмента, куда он поместится бы целиком

Виртуальное адресное пространство - им оперирует процесс; физическое адр. пр-во - память, к-я будет использоваться при его выполнении

проблема фрагментации и перемещаемости $\Rightarrow$ виртуал. память

Аппарат вирт. памяти - аппарат. средства компа, обеспечивают соответствие программ. адресов, используемых в progr., с адресами физич. памяти, в к-ой размещена программа во вр. выполнения

Базирование адресов - решение проблемы перемещаемости прогн по ОЗУ.



$\Rightarrow$ перемещаемость: нужно только загрузить адрес начала в R_{базы}.

Страничная память

Всё адресное пр-во разделяется на блоки фиксир. размера (степеней 2) - страницы.

Структура адреса:

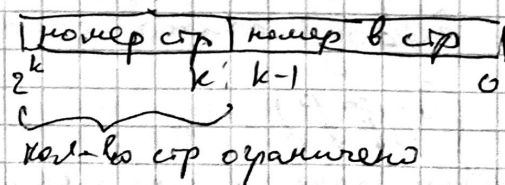
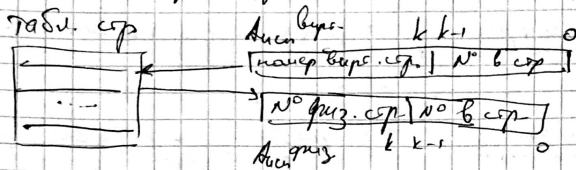


Таблица страниц процессора. Её размер - кол-во вирт. страниц, доступных процессору.

Преобразование вирт. адреса в физический - замена номера вирт. стр на соответ. номер физич. стр.

Вирт. Апп. ограничен размером машин. слова. Он помещается в регистр адреса, как. лб больше машин. слова (!). Вырезается номер вирт. стр и индексирного по табл. стр. $\Rightarrow$ номер физич. стр., записываем на место вирт. номера. Номер физич. стр. может превосходить макс. номер вирт. стр. $\Rightarrow$ объём физич. стр. больше объёма вирт. стр. (потенциально)

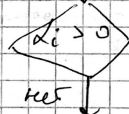


Модельный пример вирт. страничной памяти

Есть табл. из m стр., в кажд. код d_i (≥ 0)
 $d_i > 0 \Rightarrow$ адрес физич. стр., $d_i = 0 \Rightarrow$ содерж. не совб. физич. стр. (содб. i -й вирт. стр.)

0	d_0
1	d_1
...	...
i	d_i
...	...
$m-1$	d_{m-1}

Адрес вирт. [н° вирт. стр. | н° стр.]



да i -й вирт. стр. совб. физич. стр. с номером d_i $\rightarrow$ получ. физ. адр., передать работ.

получ. вирт. адрес не размещён в ОЗУ

прерывание и защита памяти

стр. ещё не загружена в ОЗУ.
 (стр. откачена в целях оптимального использования ОЗУ).

подкачки нужной стр., вычисл. физ. адр., передать работ. процесс

обращение в "чужую" память

STOP

Решение проблемы - фрагментация
 - исключение обращения в "чужую" память

Табл. стр. содержит интр. о преобраз. вирт. стр. в физич. для одного процесса

Не нужно держать все процесс в ОП, достаточно несколько страниц.

Минусы: - Большие накладные расходы (никто не организует такой огромный буфер в ОП)

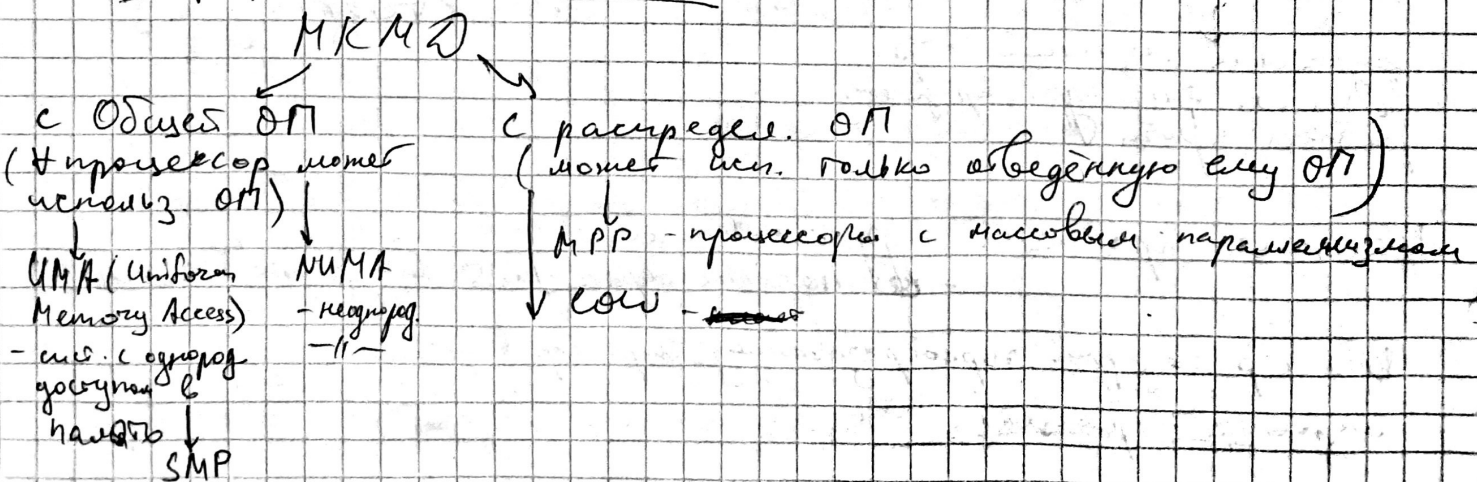
- другие накладные расходы при смене процессов
- если в ср занят 1 байт, то вся ср занята (не существ. минус, т.к. реально ср в памяти мало $\Rightarrow$ фрагментация внутри ср - не принципиальный вопрос).

Классификация архитектур многопроцессорных ассоциаций

Автор: Майк Флинн - рассмотрение с 2 позиций: 1) поток инструкций (команд), 2) поток данных (операндов) - связан с потоком команд. Команд поток и ДБ одноканальны или многоканальны $\Rightarrow$ Классификация:

- ОКОД: однок. поток команд, однок. поток данных (исполн с ед. ЦП)
- ОКМД (матричная обработка данных, напр. сложение векторов).
- МКОД (вынужденная категория)
- МКМД (мн-во процессоров одновр. выполняет разные под-ти команд над общими данными)

Иерархия МКМД систем:



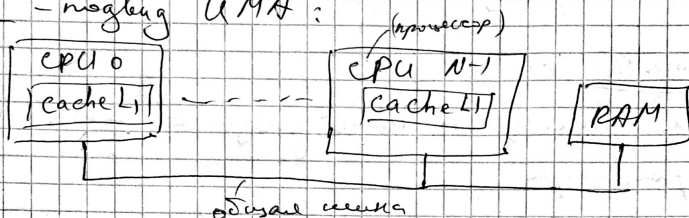
Системы с общим ОП : ОЗУ, 4 процессорных ян-т имеет доступ к 4-м блокам общего ОЗУ, т.е. 4 адрес-мБ исполняемым для 4 процессора.

Виды : • UMA - Uniform Memory Access
- SMP - Symmetrical Multiprocessing
• NUMA - Non-Uniform Memory Access

UMA : унитаризир. доступ к памяти

Все процессоры равноценны в смысле доступа к памяти

SMP - модифиц. UMA :



Приведение кэш-памяти к состоянию при чтении/записи

Операции Локальный кэш Кэш других процессоров

Чтение при чтении Запись памяти → кэш — (операция видна?)

Получение при чтении Использование кэша — (операция не видна)

Чтение при записи Запись в кэш Совм. запись в кэш удаляется

Получение при з. Запись памяти → кэш Совм. з. в кэш удал.

SMP : ⊕ Простота реализации

⊖ Задержки при доступе к памяти

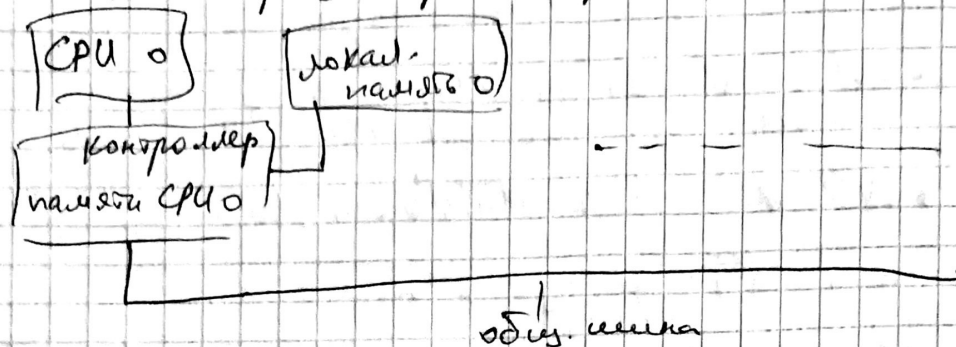
⊖ Общая шина — "узкое место" (являя централизирующей)

⊖ Проблема синхронизации кэша (реш. кэш-память с отслеживанием)

⊖ Огр. кол-во процессорных ян-тов (⊗ ⇒)

ЦУМА : общ. шина с узлами

Узел : процессор, контроллер, локал. память



Проблема синхронизации кэша

Сист. с распр. ОП (*)

Локал. процессор не имеет доступа к локал. ОЗУ
другого процессора

(*) - объединение компьютерных узлов: процессор и ОЗУ
узкоспециализированы.

Высокая эффективность при решении опред. класса задач
Высокая стоимость.

COW-кластеры (Cluster of workstations)

Кластер - объединение вычислит. узлов посредством компьютерной
сети для распределенной обработки данных

Кластеры: - высокопроизводит. система

- кластеры надежности (сохранение работоспособности
при снижении производительности)

- ⊕ простота такой архитектуры
- ⊕ относит. универсальность: массовость и расширяемость
прогр. обеспечения
- ⊕ развитая поддержка
- ⊕ топология эл-тов, перегрузка
- ⊕ Отвод тепла (для больших систем - работ. станц.)

Многомастные и многопроцессорные ассоциации

Терминальный комплекс - многомаст. ас-я, предназначен для организации массового доступа удалённых и локал. пользователей к ресурсам некоей вычисл. сист.

Краниальное устр-во - напр., принтер. МБ
локальные: подключаются непосредственно к вычисл. сист.; удалённые: подкл. к выч. сист. н/з некоей коммуникационной среде (ком. среда - напр., телефонная сеть)

Терм. устр-во - компьютерное $\Rightarrow$ для связи с аналоговыми ком. средой нужно устр-во, переводящее цифровой сигнал в аналоговый - модем.

Мультиплексор - устр-во, позволяющее соединить 1 канал выч. системы с N каналами терм. устр-в.
(МБ мультиплексор $N \times N$)

Схемы связи / каналы

Компьютерный канал - выстраивается на время связи

Пример: помехи при звонке $\Rightarrow$ перезваниваем! система строит новый маршрут соединения, скорее всего без задейств. узлов

Выделенный канал: всегда готов, дежуримированное соединение

- директ. связь объектов

⊖ деградация системы; система массового обслуживания

⊕ простота

Канал точка-точка - соединение 2 абонентов

Многоточечный канал (мультиплексор)

Направления развития связи

- Симплексные каналы - связь в одну сторону (моноканал, передача, мегафон)
- Дуплексные - обмен информацией в обе стороны
- Полудуплексные - в кажд. момент времени возможность передачи информации только в одном направлении

Лекция 6

10.10

Локальное устр-во либо подключено к компьютеру напрямую ($\approx 1/3$ разъем), $\approx 1/3$ мультиплексор (несколько устройств)

Терминальное устройство либо удаленное: схема связи такова, что м/у устр-вом и системой есть коммуникационная среда

Устр-во получает и отправляет цифровой сигнал - послед-во 0 и 1. Чтобы подключить к аналоговой сети, нужен был переводчик - модем (сбрас модем) - переводчик с любой сети на любую, знает м/у цифр. и аналоговой

Телефонная коммуникац. среда коммутируемая - при каждом разговоре строится новый маршрут.

Видеосвязный канал - постоянный. Напр., провод от телефона до станции, от неё к др. телефону. Большая нагрузка $\Rightarrow$ отказ связи.

Компьютерные сети

- объединение ЭВМ, взаимодействие. $\approx 1/3$ некая коммуникац. среда.

Ком. среда - каналы и средства передачи данных; линии связи и оборудование.

АТС - получает запрос на построение маршрута, прокладывает линию дальше

Составляющие комп. сети:

- Хосты - осн. или абонентские компьютеры
- Коммуникационные или вспомогательные компьютеры - строят маршрут (шлюзы, маршрутизаторы. --)

Связь осуществляется посредством обмена сообщениями - логическим целым порциями данных, чья длина произвол. размер.

Взаимодействие происх. в теч. некоего времени - срок связи. Порядок обмена и интенсивность сообщений не регламентируются.

Модели построения комп. сети:

• Сеть коммутации каналов

Обеспечивается выделение абонентам канала на весь период сеанса связи осуществляется обмен сообщениями

⊕ детерминированное качество связи (знаем хар-ки пропускной способности и т.д.)

⊕ требования к коммуникационному оборудованию минимальны

⊖ требование изысканий сети

⊖ ожидание: мы не можем построить маршрут, пока все линии заняты. Время ожидания недетерминировано

⊕ сбав/отказ - повтор передачи информации

⊕ после установления соединения сеть в сост. готовности

⊖ т.к. ^{накладн.} расходы по передаче данных

• Сеть коммутации сообщений

Взаимодействие - посл-ть обмена сообщениями м/д от одного комп. устр-ва к другому

Если ком. устр-во занято - сообщ. идёт $\Rightarrow$ буферизация сообщений

⊖ необход-ть наличия средств буферизации неопредел. как характеристик (размер сообщения при вызове, кол-во сообщ. в буфере, пропускная способность канала $\Rightarrow$ размер буфера ??? нужно определить)

опред. размер буфера

- интенсивность сообщ.
- скорость передачи данных (приход и уход из буфера)
- частота ошибок при передаче
- хар-ки размеров сообщ.

⊕ отсутствие занятости канала на коротких промежутках времени

Вывод: коммутация сообщ. не лучше ком. каналов

• Сеть коммутации пакетов

Пакет - порция информации определённого размера. ^{Сообщ. разбивается на пакеты.} В пакет добавляется служебная информация: размер сообщ., какому сообщ. принадлежит пакет, номер пакета в сообщ., адреса откуда и куда передавать, время отправки пакета и т.п.

По сети перемещаются пакеты. $\Rightarrow$

⊕ можно однозначно определить хар-ки ком. устр-ва (напр., буферов)

⊕ ошибка $\Rightarrow$ повторим не всё сообщ., а пакет

⊖ значительные накладные расходы (доп. информация в заголовках, кот. не нужна пользователю; расходы на формирование (небольш.) и приём (больш.) пакетов, т.к. маршруты для разн. пакетов могут быть разными, мб сброс, проверка целостности пакетов,

разное время передачи пакетов — сколько ждать? Когда пакет, то пакет потерян? Разные таблицы).

В можно определить размер буфера (ситуация с переполненным буфером всё равно не!))

Появление пакетной коммутации — 1961. Автор: Леонард Клейнрок (MIT). Первая публикация (1961).

DARPA: проект ARPANET, 1967 — военный проект, появление пакетной сети (!)

В 1970-е: унифицирование стандартов взаимодействия

Модель ISO/OSI организации взаимодействия в сети (2)

Разделение на уровни:

1) Физич. уровень — обеспечение передачи неструктур. данных с помощью 0 и 1. Унифицирует стандартизацию сигналов

2) Канальный — декларируется правило передачи данных по физич. линии; правило доступности линии, синхронизация, обнаружения и исправления ошибок. МД организована адресация. Передача данных.

3) Сетевой — обеспечивает решение задач взаимодействия сетей. Сетевое взаимодействие. Задачи: именование сетевых устр-в (чтобы они были доступны др. сетям), маршрутизация, обеспеч. связи м/у взаимодей. устр-вами. Обеспечивается буферизация и перемещение пакетов

4) Транспортный — взаимодействие м/у взаимодей. программами. Принимается решение о виде взаимодействия (с установлением⁽¹⁾ / без виртуал. канала⁽²⁾)

(1) — контроль за доставкой и получением всех пакетов и

обработкой пакетов. (2) $\rightarrow$ после отладки пакета
транспортный уровень не выполняет никаких действий

⊗ Стандарт для разработки детерминир. расширения
комм. сетей

5) Семантический - обеспечивает реализацию семант. связи.

Формируется семант., принимается подтверждение о
подлинности и правах. Уровень связан с транспортным,
зависит от установленных верт. каналов

6) Представительный - решает проблемы кодировки

Унификация представления информации

7) Прикладной - формализует сетевые прикладные
задачи. Формализация и декларация правил взаимо-
действия м/у сетевыми системами.

Взаимодействие ^{в сетевых узлах} может осуществляться с помощью

протокола - формальное описание сообщений и
правил, по кот. сетевые устр-ва осуществляют
обмен информацией; правила взаимод. одноименных (!)
уровней. Протокол регламентирует правила и формат
данных.

Каждый уровень может взаимодействовать
с нижестоящим уровнем по правилам, кот. наз.
интерфейсом.

Нижест. ур. предоставляет вышест. му некий
набор операций - служба (сервис)

В наст. вр. лидирует семейство протоколов TCP/IP
(на самом деле - из разработки ARPANET)

Уровни:

- 1) Ур. доступа к сети - оур. физ. среду
- 2) Метасетевой - оур. правил маршрутизации, секционирования
- 3) Транспортный - обеспеч. взаимодействие сетевых

программ, стратегию передачи

- 4) Ур. прикладных программ

- 1) протоколы обеспечивают систему требований для передачи данных другим устр-вам сети

Пример: протокол Ethernet. Единая шина. Широковещательная. Широкая кабели, подключение компьютеров.

Для сетевых устр-в обеспечивается множественный доступ, с контролем и обнаружением коллизий (напр. 2 устр-ва обрабат. & свобод. шина, тогда каждое свобод. каб. пропускает, кто-то пройдет первым).

Ethernet имеет уникальные идентификаторы (укажд. устр-ва ^{уникал. числ} ~~своб. номер~~).

- 2) TCP-транспортный протокол, IP-метасетевой. IP обеспечивает уникальное именование сетевых устр-в. Изначально был 32-битный. Сейчас используют 64-битный.

Протокол 32-бит. адрес: разбивается на классы

Класс А:

0	сет	хост: 3 байта
---	-----	---------------

номер сети (1 байт)

префиксное байты - номер сети

суффиксное - номер устройства (3 байта)

Класс В: 2 преф. байта - номер сети. Преф. байты: 10

110	сет	хост
-----	-----	------

Порядок таких сетей 2^{14}

2 сур. байта

Класс C

1110 1000 1000

сущ. байт - номер сетевого устр-ва в этой сети
3 преп. байта

Стек протоколов - ~~реализация~~ на кон.-во уровней
сетевого взаимодействия, реализованные на конкретном
сетевом устр-ве

Мноуз - устр-во, передающее пакеты м/у разлн. сетями.
~~Стек протоколов~~
V (сет. из метасетевого ур. и ур. доступа к сети.

Имеет не менее 2 адаптеров (принадлежат разным
сетям) $\Rightarrow$ мноуз принадлежит одновременно 2 и более
локал. сетям

Хост ^{интерн} $\rightarrow$ сеть А $\rightarrow$ мноуз $\rightarrow$ сеть В

3) TCP - протокол, обеспе. взаимодей- установленная
соед. с вирт. сетью, доставку данных и обнаруж. и
исправл. ошибок (протокол контроля передачи)

UDP - отправляет пакеты с данными, не контролирует
их доставку (протокол пользовательских данных)
(User Datagram Protocol)

4) Форматизация сетевых, декларация
форматов фр-т и прикладных прог

Основы архитектуры ОС

ОС - комплекс программ, обеспечивающий контроль
за существованием, распределением и использованием
ресурсов Вир. сист.

Существование ресурса - существование драйвера
этого устройства.

Распределение ресурсов — ОС принимает решение о ^{порядке} предоставлении ресурсов (несколько обращений к одному ресурсу)
ОС знает, какими ресурсами должен обладать создаваемый процесс

- Предварительная декларация ресурсов — выделение ресурсов заранее.
- Выделение ресурсов по запросу (обычно использ. окс)
- Модели выд. ресурсов

Процесс — совокупность маш. команд и данных, обрабатываемая

в ОС и обладает правами на владение нек-ми наборами ресурсов

Исключения — эксклюзивное использование (ЦП, ОП не экск. ис.)

— совместное использование (ресурс выделяется для 2 и более процессов)

Требования к ОС

- Надежность — минимизация внутр. сгл., кот. могут приводить к сбоям функционирования

- Защита — инт. и ресурсов от несанкц. доступа

- Эффективность

Должны быть определены критерии эффективности!

Любая ОС нацелена на решение задач в конкретной предметной области $\Rightarrow$ критерии

- Предсказуемость

ОС должна быть предсказуема в смысле сбоя и трасс-матричных обстоятельств, должна быть известна пос-те действий в этих случаях.

Известны заранее проблемы и последствия различных действий.

Структура ОС

Ядро - (ключевой эл-т) резидентная (постоянно находится в ОП) часть ОС, работающая в режиме супервизора (доступны все команды процессора). Обычно ядро работает в режиме физической адресации (вирт. и физ. адреса совпадают, не происходит преобразования). ОС подгружает динамические драйверы устр-в (резидентные / нерезидентные, работают в пользовательском / привилегированном режиме). Подключается устр-во → система ищет нужный драйвер в базе, → подгружает его (динамически подгружаемые драйверы)

Напр: драйвер сканера не нужно постоянно держать в ОП

Системный вызов - интерфейс, посредством кот. можно осуществлять обращение к ОС за предоставлением той или иной усл. др-и. Реализация некоей др-и, кот. хочется, чтобы выполняла ОС.

Библиотечная др-я живёт на ресурсах вызываемого процесса

Максиня 4

Ядро может выполнять любые команды процессора.

Переключение с одного процесса на др. тяжело, т.к. требует перенастройки виртуальной памяти.

Ф-и ядра зависят от ОС

Монопольные ядра - от одного др-а

Микроядра - много.

Ядро работает в физич. памяти

Интерфейсы системных вызовов

Динамически загружаемые драйверы физ. и вирт. устр. в

структура ОС

Ядро ОС

Аппаратура

Примеры структуры ОС

Монолитное ядро - 1 программный модуль, в рамках кот. реализована вся функциональность

⊖ Оптимизация = перекомпиляция всего ядра

⊕ Эфрективность (меньше накладн. расходов)

Многоуровневая архитектура - логически разбивается

р-и ядра на дискр. кол-во уровней

Запросы выполняются г/з уровни

Напр., уровень реализует взаимодей. с конкрет. драйв. сист.; с внеш. устр-вами.

⊕ Легко оптимизировать отдельные уровни (интерфейсы должны совпадать!) ⇒ технологичность (легко модифицировать)

⊖ Понижается эфрективность

⚠ Чем больше структуризация в программной системе, тем больше накладных расходов

Пример - программный модуль. + накладн. расх: вызовы и возвраты, сохранение контекста

Микроядрная архитектура

Микроядро - часть ОС, содержит минимального

реализацию функционала (?)

Микроядро - унифицир. интерфейс, кот. может обеспечить произвол. кол-во драйверов, кот. реализуют опред. р-и ^{подключение}

Функционал строится на подмножествах драйверов

- ⊕ Легко изменить функционал, он разнообразен
- ⊖ Накладные расходы: цепочка запросов по драйверам

Пример: Mach

⊕ Микроядро маленькое (и в нём можно комбинировать функциональность, не зависящую от аппаратуры (?) → мобильность)
Перенос с одной архитектуры на другую проще — нужно реализовать только это маленькое ядро

Логические ср-и ОС

- Управление процессами
- Управление ОП
- Планирование (оптимизация обработки поступающих запросов: организации очередей и стратегий обработки)
- Управление устройствами и драйверами систем (это — внеш. устр., а это — драйв. сист., как это организовывать ⇒ стратегии)
- Сетевое взаимодействие (напр., телефон с сотами)
- Безопасность (! ключевая ф-я в наше время)
(контроль запросов интр-и)

Типы ОС

↓
Пакетная Система разделения времени ОС реального времени

Разделение определяется стратегией распределения времени м/у процессами (распределение ресурсов ЦП)

Пакетная ОС

Критерий эффективности — соотношение вр. работы
~~процессора~~ к вр. работы процессора, максимизация
этого значения (→ 1)

Переключения выполняемых процессов:

- выполнения процесса завершено
- возникло прерывание (напр., требуется время на выполнение др. процесса, или нужен этому)
- система решила, что процесс "зависел"

Цель: минимизация переключений процессов

⇒ сокращение накладных расходов

Аналогия: кэш - минимизация обращений к медленной системе

Критерий эф. кэ: ~~минимизация~~ мин. наклад. расходы

Системы разделения времени

Квант времени ЦП - некот. промежуток времени работы ЦП. Время, кот. может использоваться 1 процесс

Переключение выпол. процессов:

- истечение выделенного кванта времени
- выпол. заверш.
- прерывание

// закидывание - завершение опр. кванта времени

Можно иметь квант для разных процессов

Крит. эф.: т.ч. квант времени (?)

Кодирование системы планирования процессов

Смесь процессов

Интерактивные
(нажал кнопку - сразу получил результат)
В квант. момент впр. опр.

Безотладка
приоритет и квант впр.
неделимые процессы
(напр. прогноз погоды)
 P_2, st_1

Основная часть
неделимые процессы
приоритет и квант
 P_3, st_3

Есть наиболее процесс, где приоритет выше, выполняемый процесс прервется ради него

Пример использования разделений на такие группы:

Сумм: $P_3 > P_2 > P_1$ $P_1 > P_2 > P_3$ $P_2 > P_3 > P_1$

Выполнение процессов в удобное время (работает/не работает)

st_3 - диск. доливание (5 мин, 10 мин, час)

st_2 - в зависимости от процесса

st_1 - доли секунды (интерактивная работа)

ОС реального времени

- организация сист. такова, что при ее функционировании гарантируется обработка событий из некоего предопределенного за время, не превышающего некоего предельного значения

Управление процессами

Процесс - совокупность команд и данных, которая обрабатывается в рамках вычисл. сист. и обладает правами на владение некоем набором ресурсов

Ресурсы → принадлежат только одному процессу

Разделяемые (2 и более процессов) (напр. общ. память)

Выделение ресурсов процессу:

- Предварительная декларация (до начала выполнения)
- Динамическое пополнение списка принадлежностей процессу ресурсов (по запросу; напр. malloc)

Жизненный цикл процесса

- одна из основных задач ОС

Предполагается, что в любой ОС любой процесс может находиться в некоем состоянии: формирование, выполнение, ожидание, завершение. Перемещение по этим состояниям — жизн. цикл.

Граф состояний процесса — ориентир. граф переходов

Модельная ОС

1) Бuffer Ввода процессов (БВП) — пр-во, в к-е размещаются и хранятся сформированные процессы с момента их образования до начала выполнения. Обычная пакетная ОС

2) Бuffer обрабатываемых процессов (БОП) — Бuffer для размещения процессов, находящихся в мультипроцессорной обработке.

Модель пакетной однопроцессорной системы

Граф состояний: 0. Ожидание начала обработки → 1. Обработка ЦП → 2. Заверш.

Мультипроцессорная система



Типы процессов

- "Полновесные" — выполняются внутри защищённых участков ОП
- "Легковесные" (нити) — работа в мультипроцессорной среде на ресурсах полновесного процесса (каждый виртуальный этот нить). Мультипроцессорная среда имеет защиту памяти в рамках полновесного процесса (пользователь может испортить окружение только самому себе), нить одного процесса защищена от

нитей другого.

Примеры нитей: поиск максимума

Процесс вкл в себя:

- 1) исполняемый код
- 2) собств. адрес пр-во - совокупность вирт. адресов, кот. может использовать процесс
- 3) ресурсы системы (назначены ОС)
- 4) хотя бы 1 выполняемая нить

Контекст процесса - совокупность данных, характеризующих актуальное состояние процесса

- Пользовательская составляющая - текущ. сост. программы (совокупность маш. команд и данных, размещ. в ОЗУ)
- Системно-аппаратная сост.-я - информация ОС, которую она сохраняет при выполнении процесса (PID, PPID, регистры и т.д.)

Процесс в Unix

Объект, зарегистрированный в таблице процессов ОС (1)

Объект, порождённый системным вызовом fork() (2)

(1): табл. можно задать так много процессов $\Rightarrow$ размер таблицы

Много процессов - ресурс вык. сист.

Табл. позиционная: сост. из записей фикс. размера, нумерованных с нуля

2/3 запись можно получить доступ к контексту процесса

Контекст процесса

Пользовательская сост.-я ("тело процесса") (*)

Аппаратная сост. (**)

Системная сост. (***)

(*)
 Сегмент кода Сегмент данных

- Маш. команды
- Константные const
- Статические данные (переменные)
- Разделяемая память
- Блок (фрейм) параметров в ф-ях, автоматические переменные, динамическая память

Unix система позволяет оптимизировать ^{инструменты} процессы

(**) Процесс в состоянии выполнения (регистры и используемые таблицы и т.д.)

(***) то, что формирует и сохраняет система

для выполняемого процесса:

- идентификатор родительского процесса

- текущий счетчик

- приоритет

- реальный и эффективный идентификаторы

пользователя - владельца: процесс формируется от имени пользователя, работающего в сист. При запуске процесса формируется атрибут - реальный идентификатор.

(кто запустил процесс). Хозяин файла $\Rightarrow$ эф. ид-р.

($\Rightarrow$ процесс имеет права хозяина $\Rightarrow$ может работать с/з системными данными)

(2). Обращение к системному вызову $\Rightarrow$ ОС формирует новый процесс - копию процесса, из кот. произойдет обр. к систем. вызову, и ~~возвращается~~

При удачном завершении возвращается:

• Основному процессу - 0

• Родительскому - PID порожденного

Неудач. завершение - -1 (не удалось сформировать процесс)

егого - переменная, в кот. записаны кад. описки
составляющие контентного наследия при вызове fork().

- Окружение
- Файлы, открытые в родит. процессе (!)
- Способы обработки сигналов
- Разрешение переустановки эфрр. идент-ра пользователя
- Разделяемые ресурсы род. процесса
- Текун. рабочий и домаш. каталоги
- ...

— // — не наследуемые:

- PID, PPID
- Сигналы, идущие добавки в родит. пр.
- Время посылки опедативного сигнала
- Блокировки файлов, установка род. пр-м.

Не выполняется exec:

- кат. исполняемого файла
- файл не исполняемый
- кат. прав
- ...

Завершение процесса:

- сист. вызов `exit()`: можно передать код зав. процесса: 0 - хорошо, 1 - ошибка
- пишется в составе `_main()`, выхад за закрывающую операторную скобку ответственного блока ф-н
- получение сигнала

Лекция 8

24.10

Процесс — объект, зарегистрированный в табл. процессов
Табл. пр. — вытаскивает ресурс ОС $\Rightarrow$ находится в ОП $\Rightarrow$
нужно место $\Rightarrow$ нужно вытаскивать тех же
процессы (размер табл.)

Процесс однозначно идентифицируется по
номеру (от 0 до max) N^o — объект (2), единица с
меткой структуры
`fork()` — отказ, если процесс создаст нелзя; если

ничего нет, то создается копия процесса, оба с
неодного PIDом и некоем атрибутом

Родительский процесс завершается $\Rightarrow$ все сыновья
усыновляются специальным процессом с PID = 1

Нет динамики: не можем оптимизировать алгоритм
семейство системных вызовов `exec()`

— замена тела текущего процесса на тело, к-е
может быть сформировано исходя из файла,
переданного π/z `exec`

Ошибки: можем передать некорректно сформированный файл;
или файла, не являющегося исполняемым (Unix-сист
опред. по содержанию, эти ли файлы исполняемы)
файл не подходит для процесса, к-е обратился
к `exec()` $\Rightarrow$ `exec()` возвращ. -1, в итоге — потому

При успешном выполнении управление оказывается
на точке входа нового тела $\Rightarrow$ не возвращается

Заверш. процесса

— сист. выз. `exit(code)`; `code` — код завершения
(определим, почему завершаем процесс, какими-то
алгоритмами). Переводит процесс в стадию

завершение: освобождение памяти, усыновление потоков

- выход из закрывающей скобки объемлющего блока ф-и

- сигнал

wait

- системный вызов в ожидании событий в одном из сыновних процессов. Возв. завершение.

Возвращает PID сыновнего процесса, у коб. произошел событие.

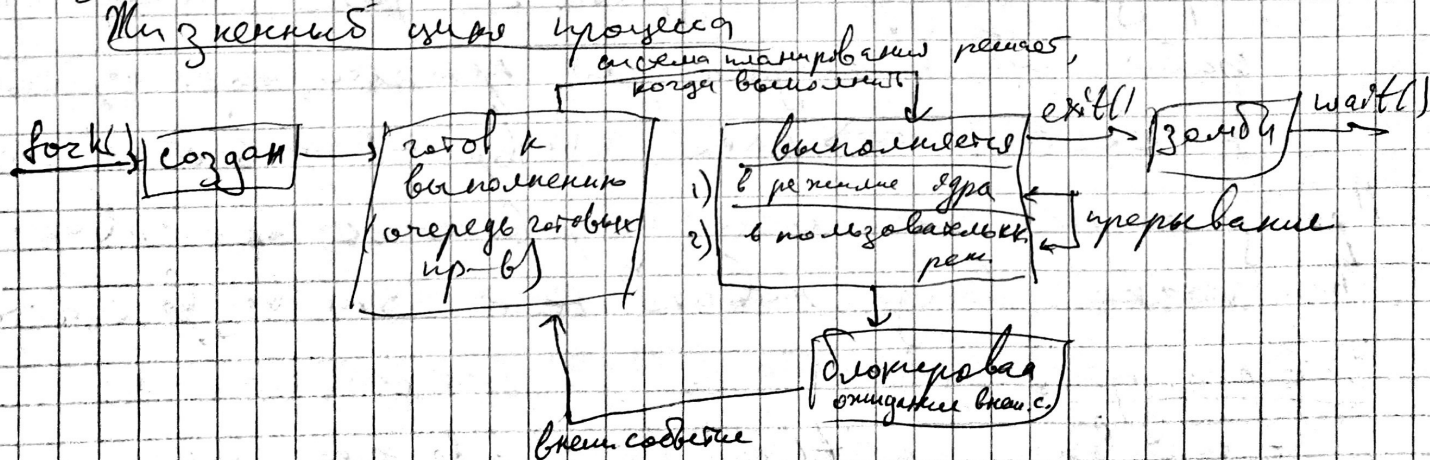
`pid_t wait (int *status);`

status возвр.: в старшей части - пользовательский код завершения (из `exit(code)`), в младшей - системный код завершения процесса.

При обращении к функции `wait` заблокирует текущий процесс, если есть сыновние процессы и ни у одного еще не произошло событие (завершение); из завершившихся выбирается один и информация передается

из `wait`

Жизненный цикл процесса



⇒ Начальная загрузка

Включение ⇒ управление передаётся в точку
аппаратного управления ⇒ загрузка ядра (и
прога, позволяющего выбор программы ОС) ⇒
передача управления на точку управления
загруженной ОС.

- чтение базового блока системного устройства
аппаратным загрузчиком

- поиск и считывание в память графика / и др.
(ОС)

- запуск на исполнение

Если не удастся загрузить ОС, можно
запустить с альтернативного системного устр-ва

⇒ Начальная инициализация компонентов системы: установка и настройка

⇒ Инициализация структур данных: графовые дескрипторы и т.д.

⇒ Инициализация процесса №0: он не имеет кодового сегмента, т.е. в течение всего времени работы системы (ОС?)

⇒ Создание процесса №1 (напомню не стандартным образом)

- копируется пр. №0

- созд. область кода процесса №1

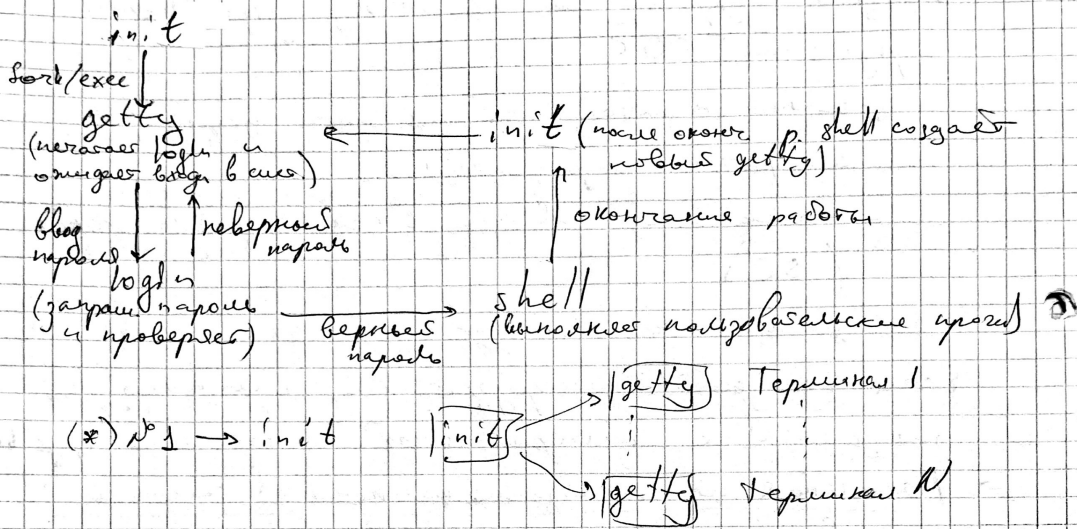
- копирование в область кода процесса №1 прог.,
реализующей сист. выз. `exec()`, коб. необх.

для выполнения прог. /etc/rc/rc6
(стандарт. загрузка)

⇒ замена команд пр. №1 команд из файла /etc/crontab
(запуска exec(1))

⇒ подки интерпретатора команд и системный консоль

⇒ созд. многопользовательской среды (*)



init может завершить инициализацию процессов

Взаимодействие процессов

Параллельные процессы — выполнение/обработка которых хотя бы частично пересекается по времени

→ Независимые процессы : используют независимые ресурсы

→ Взаимодействующие : 2 или более проц. имеют доступ к одному ресурсу. Влияют друг на друга

Разделяемые ресурсы — доступны нескольким процессам

Критичные ресурсы — разд. р-рн, к которым должен быть доступ в текущ. момент только одному процессу

Критическая секция (interval) — часть прогн, в рамках коб. осуществляется работа с критич. ресурсом

С т. зр. взаимодействия процессов нужна корректность:
Предвзятые мифы прогн-я: Рез. выполнение процессов не должно ~~точно~~ зависеть от порядка переключения выполнения м/у процессами.

Зависит $\Rightarrow$ гонки.

Взаимное исключение — способ работы с разделяемым ресурсом, при котором в тот момент, когда один из процессов работает с разделяемым ресурсом, все остальные не имеют к нему доступа

⊖ Тупик: некорркт. организация доступа к разделяем. ресурсу $\Rightarrow$ взаимоблокировка.

⊖ Блокирование: доступ одного из процессов к разделяемому ресурсу не обеспечивается из-за активности более приоритетных процессов

Способы реализации взаимного исключения:

позволяют работать с разделяемым ресурсом

- Запрещение прерываний и спец. инструкции
- Алгоритм Петерсена
- Активное ожидание
- Семифоры Делекора
- Мониторы Хоара
- Обмен сообщениями

Семафоры Дедкорта

- формальная модель синхронизации, предложенная Дедкортам
Семафоры с целочисленными значениями S

Атомарные операции (невозможны прерываю)

Операции, опер. над семафорами:

- $Down(S)$ - опустить семафор, если значение семафора ≥ 0
блокировка, если нельзя выполнить
- $Up(S)$ - поднять семафор: увеличивается значение семафора на 1. Если с семафором были связаны заблокир. процессы, один из них разблокируется и завершает команду $Down$ (высвобождает)

Бинарные семафоры, позволяют организовывать взаимное исключение

Нужна гарантированная возможность работы

с разделяемыми переменными, с общим именем

Монитор Хоара

- более высокоуровневая конструкция (Трикоур); совокупность процедур и структур данных, объединенных в специализир. программный модуль

- Структура данных доступна только для входящих в монитор процедур
- Процессы "входят" в монитор при вызове одной из его процедур
- Внутри монитора в $\forall$ моменты времени ≤ 1 процесс

Пример: мидт

Объект свободных ячеек

- 1 из самых распространенных механизмов реализации взаим. исключ.

Реализация: взаимодействие (передача данных) и синхронизация.

Ф-и: send, receive

Синхронизация: ф-и не блокирующими и неблок.

Адресация: прямая (ID процесса) или косвенная (номерный список, очередь сообщений, получатели из списка по запросу (1-ое в очереди); если список пуст, ^{образованная} процесс блокируется)

MPI - интерфейс передачи сообщений

Классические задачи синхронизации процессов
Ход от возможных рунков (взаим. блокировка)
- Обедательные философы

Круглый стол, перед каждым философом тарелка, м/у тарелками вилка, в центре посуда с макаронами. Если философ проголодается, он берёт 2 вилки (разом с сосед.) и берёт макароны. Как организовать этот процесс? (Проблема - слабые из управл. проц.?)

Лекция 9

31.10

Семантика Дейкстры - организация взаимодействия на аппарате ур

Аппарат ур. $\Leftarrow$ атомарность операции

Работ реализует на ОС (псевдоатомарн.)

Обмен сообщениями - самый распространенный механизм организации взаимод. процессов. Может работать почти на любой архитектуре

Send, receive: с блокировкой: процесс, обратившийся к одному из примитивов, будет заблокирован, если не хватает ресурсов (до момента появления ресурсов)

классическая задача синхр. пр-ва

(с помощью семафоров дедекеры)

"Обед. философы"

X простейшее реш.: если одновр. не захватят еда, каждый возьмёт по левой вилке $\Rightarrow$ deadlock! (думай)

✓ Реш. с сем. семафоров: на каждого философа разные процессы

test: $\text{up}(\&S[i])$: если $S[i] = 0 \Rightarrow \text{это - го}$

номер не так; получили вилки $\Rightarrow S[i] = 1$

Узнал. S илис. нулем

put_forks: изменим состояние философа (thinking) $\Rightarrow$

проверка изменения соед. соседей: $\text{test}(\text{LEFT})$,

$\text{test}(\text{RIGHT})$

$S[i] = 0 \Rightarrow \text{down}(\&S[i])$ - блокировка (семафор)

$\Rightarrow$ когда размещает down в критической секции (когда mutex occupied)

"Читатели и писатели"

задача резервирования ресурса

Сначала все же инициализация (хотим купить билет, напр.),

потом резервирование свобод. ресурса $\Rightarrow$ 2 ф-и:

читать и писать

Читательские процессы приоритетнее пишущих

// Прога: $\text{wait}(\&\text{pr. пр-ва})$

rc - разделяемая переменная: счётчик читателей

mutex - контроль доступа к разделу пер

db - семафор базы данных: контроль доступа к данным

2 процесса: читатель и писатель

Новый читатель $\Rightarrow rc++$

if $rc == 1 \Rightarrow$ блокируем доступ к данным: $\text{down}(db)$

Корректная обработка завершения чтения $2/3$
критич. секция (mutex)

Writer: входит в крит. секцию, только когда $dw=1$,
 $dw=0 \Rightarrow$ заблокирован

Список парикмахера

задача реализации клиент-сервера с ограничением
на длину очереди

Парикмахерская. 1 парикмахер. Колонка ожидающих
с n креслами: если свободного кресла нет,
посетитель уходит (отказ по ошибке).

barber: парикмахер. 1 - ^(истинный) готов работать, 0 - спит

Процесс barber: изнач. будет заблок, т.к.

посетителей нет ($customers=0$) $\Rightarrow$ ложь $\Rightarrow$ блок.

waiting - кол-во ожидающих

Взаимодействие процессов

В рамках одной локал. ЭВМ

В рамках сети (процессы на разных
компьютерах)

модель взаимод-я
произвольных
процессов
(не обязат. разоб.
связи)

Важна дано опр.
модель именования
процессов (процессы
1-1/3 P4 D6).

↓
сокеты

родственников
процессов
(наименование
дескриптора)

ориентированная цепочка процессов, есть разделяемые
ресурсы; каждый процесс имеет возможность
поиска взаимод-я. - симметрическая
модель

→ главный и подчиненные процессы;
у гл. обычно больше дебола по объ.
и подг; пример: обладчик и
обслуживаемая прот. - модель гл.-подчиненн.

Сокеты - могут работать и в рамках локал. ЭВМ, и сети.

Сигналы

- средство Unix'a, позволяющее передавать сообщения от ~~ка~~ любого процесса или ОС
- средство асинхронного уведомления процесса о наступлении некоего события в системе.

`<signal.h>` - файл с константами, опис. событий

Отправка сигнала: другой процесс; уведомление о событии в вычисл. системе

Похоже на аппарат прерываний

- Реакция на сигнал:
- обработка по умолчанию (дефолтная) (как правило, завершение процесса с кодом завершения, совп. сигналу)
 - вызов ф-и обработки. Ф-я будет вызвана асинхронно в рамках процесса, которому придёт сигнал; вернёт управление в точку прихода сигнала
 - игнорирование сигнала

Не все сигналы можно обрабатывать / игнорировать, см. в `signal.h`

Завершение процесса:

`SIGINT (12)` - можно перехватывать и игнорировать

`SIGKILL (13)` - нельзя перехв. и игнор.

```
#include <sys/types.h>
```

```
#include <signal.h>
```

```
int kill (pid_t pid, int sig);
```

идентификатор
процесса,
к которому посл. сигнал

номер посылаемого сигнала

Возвр: 0 - удача, -1 - ошибка

Системный вызов установки обработчика сигналов.
#include <signal.h>

void (*signal (int sig, void (*disp) (int))) (int)

номер сигнала,
для кот. установка реакции
ссылка на ф-ию
(опр. пользователем)
или 1 из констант:
SIG_DFL - обработка по умолчанию
SIG_IGN - игнор

Возврат: указатель на предыдущий обработчик данного сигнала

Пример (tzl + C (см. алг))

Пример "Будильник"

alarm(t); - сист. вызов, выдает наполняющее $\frac{1}{3}$ t
(время)

Аппарат сигналов относится к стандартным средствам взаимодействия процессов.

Неименованные каналы.

- Ресурсы, кот. могут создаваться посредством обращения к сист. вызову pipe. Параметр - указатель на массив из указателей на 2 эл-та.

#include <unistd.h>

int pipe (int *pipes);

При наличии ресурса система формирует неименованные каналы: системный ресурс с кот. ассоциируется 2 файловых дескриптора ($\frac{1}{3}$ 1-го чтение, 2 - запись). Размер ресурса предопределен.

$\frac{1}{3}$ систем. вызовы: read, write, close и т.д.
(см. концепцию файлов в Unix - все представляется файлами)

pipes[0] - чтение (г-р)

pipes[1] - дескриптор записи

Чтение и запись - блокируемые операции

Напр., запись блокируется и идёт, пока не освободится место для оставшейся информации. Это происходит, если открыт хотя бы 1 дескриптор на чтение, иначе - сообщ. об ошибке

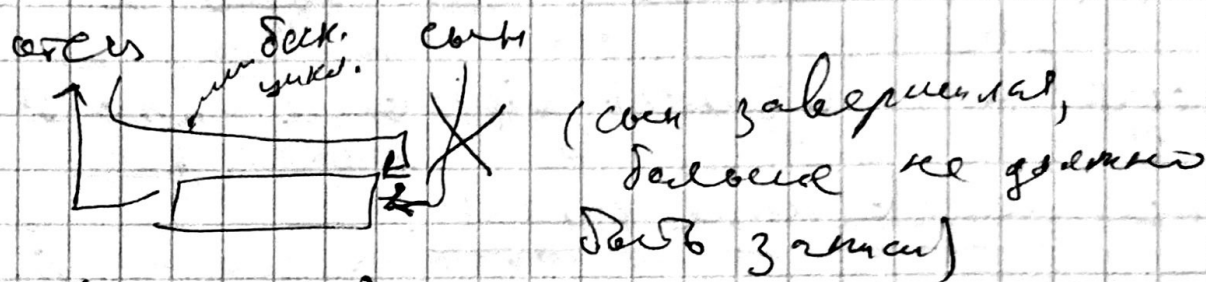
Чтение: придёт меньшая порция данных, чем ожидается
→ Блокировка (пока есть хотя бы 1 д-р на запись)

Нет записи ⇒ заверш., код EOF

(!) pipe имеет смысл, если он передаётся по наследству сыновним процессам

Связь родств. процессов: 1 читает, др. пишет (не обяз. прямые родственники, напр. дед и внук)

Закрывать ненужные дескрипторы! иначе лб деск. чики:



Пример: конвертер (ан. язык)

(!) посмотреть сист. вызовы dup; dup2:

2 параметра - файлов. дескр.-ры: N1 закрывается, N2 ассоциируется с открытым файлом, с кот. связан N1

N1 NAME1 / d1
NAME2 / d2 ⇒ dup2(d1, d2)
NAME1 ⇒ d2

fd = 0 / 1 - стандарт. вв / выв

Лекция 10

7.11

Взаимодействие процессов в рамках компютер. сети (многоадресная)

Комп. сеть: лб имеет, лб устройство (напр. мастер) (локал.)

Пример: самолёт, машина управляет комп. сетью (локально)

Сигнал - программное прерывание

Пример. Совместное использование сигналов с каналами.

"Пинг-понг"

2 процесса могут и читать, и писать в канал.
Процессы передают друг другу сигнал ("шарик"). Счётчик — знак, передаваемое ± 1 именнов. канал (средство синхронизации). ("игра" должна закончиться в определённый момент $\Rightarrow$ кол-во передач сигнала).

MAX_CNT — max кол-во передач

cnt — счётчик (счётчик переим.) // нужно было указать инициализацию нулём!

target_pid — целевой pid

ld[2] — массив, опис. канал

read(ld[0], &cnt, sizeof(int)); — чтение актуал.

знак cnt

kill(target_pid, SIGUSR1); — передача сигнала

Первый процесс завершится сам

Если родителю $\Rightarrow$ передаёт сигнал сыну чтобы завершить его

while (wait(&status) == -1); — выйдём, когда процесс завершится (сигнал $\Rightarrow$ сиг. вызов $\Rightarrow$ -1)

send-ресурс с блокировкой: read в процессе — сыну заблужд., если родителю еще ничего не написал

Именованные каналы

Именов. канал — разновидность графа в граф. мес.

// FIFO * 1/2

int mkfifo(char *pathname, mode_t mode);

имя создаваемого канала

права доступа + режимы стр.

Имен. каналы могут существовать без прикрепл. канала

Не можем менять чтение/запись (режим)

Симметрич. взаимодействие: канал, или канал (кажд. процесс имеет равные возможности)

Взаимодействие "главный - подчиненный"

Ассиметрич. вз.: возможности одного процесса принципиально больше по объёму

```
#include <sys/ptrace.h>
```

посмотреть!

```
int ptrace (int cmd, int pid, int addr, int data)
```

код выполняемой команды подчинён. процесс-канал (с кэш. хэбл. осущ. действий) адрес в адр. пространстве (адреса) инфра

- Позволяет ~~на~~ братившемуся процессу передать 4 параметра
cmd: группы команд:

- чтение (из сегмента кода, из сегмента данных - адресов. память, из контекста процесса - чтение регистров подчинённого процесса: какой регистр - поле адреса)

// Нельзя организовать коррект. рекурсию

- запись (изменение сегм. кода (нормально для "м.-подг."), данных, ~~про~~ контекста процесса)

! Чтение и запись могут выполняться только над основанными процессами

- управление (продолжить выполнение с адреса / с заданного адреса, "шаговый режим": по команде продолжить подг. процесс выкал. 1 команду, завершить процесс; трассировка)

Единств. команда, доступ подг.-му: трассировка
разрешено.

Разрешение облади остатке после сменен

Контрольные точки - в ас. проекции φ и λ при основании.

Лекция 11

14. 11

Глав. процесс может осуществлять управление кодом подчиненного: запускать (rtase - сист. вызов), останавливать (сигналом - сист. вызов kill; дождается кода завершения от подчин. процесса). Если подг. запущен, он может только посылать сигнал - остальные действия только в случае остановки. Остановка означает, что ОС забрала часть контекста процесса (регистры) $\Rightarrow$ изменение регистров. Записали значение в регистр подг. процесса $\Rightarrow$ реально его значение изменилось как только снова запустили процесс!

Система тематического взаимодействия ТРС

Создатель

1) Служит разделение ресурсов, кот. могут быть доступны 2 и более процессам. Они живут до: 1) ко-то удалит ресурс 2) завершения сеанса ОС. Могут $\neq$ без процесса, использ. их. Группы ресурсов:

```
#include <sys/types.h>
#include <sys/ipc.h>
key_t ftok (char * filename, char prog)
```

Как организовывать именование? $\Rightarrow$ канд. создаваемых
ресурс ассоциируется с некой категорией. По каталогу:

а) создать ресурс, б) подключиться к нему

Права создания = права создания

Генерация уникальных ключей: сист. вызов `ftok`

(создаётся на основе параметров, времени и т.д.)

`<ResName> get (key, ..., flags)` - сист. вызов: либо
(подпись, без. с ресурсом?)

подключение к существующему ресурсу, либо создание
и подключение

Flags: - `IPC_PRIVATE` - создание нового ресурса (если
уже есть ресурс с таким ключом → сист. вызов получит
отказ), доступного, только породившему процессу.
Взаимодейств. с родств. процессами (аналог кеш-мемов.
каналов)

- `IPC_CREAT` - создать новый или подклю. к уже

- `IPC_EXCL + IPC_CREAT` - создание нового

Ⓢ Посмотреть подробности в мануалах

Сист. вызов - ctl

`<ResName> ctl`

можно управлять ресурсом: удалить, копир., установить
несовместим. значение

Очереди сообщений

Принцип FIFO

Может рассматриваться как: - сквозная (очередь сообщений
всех типов)

- совокупность подочереди разных типов

Сист. вызов для очереди: создание/доступ к очереди сообщ.

```
#include <sys/types.h>
        <sys/ipc.h>
        <sys/message.h> ](*)
```

```
int msgget(key_t key, int msgflag);
```


В арг. функция возвр. номерок указ-гексиратор орегед,
кегдари - -1.

msg flag - грам, управлененне поведеним возрора

Отправка сообщ:

^(*)
#include <sys/msg.h>

int msgsnd (int msqid, const void *msgp, size_t msgsz,
int msgflg);
гексиратор орегед, указавен на размер буфера
наизв. в раз. msgget()

Получение сообщ:

int msgrcv (int msqid, void *msgp, size_t msgsz,
long msgtyp, int msgflg)

// ат. ататер (!) : граммог. процесор 2

Управление орегедом сообщ: int msgctl(---)

ат. процесор (ататер)

IPC: Разделение имен

^(*)
#include <sys/shm.h>

int shmget (key_t key, int size, int shmflg)
номер гет
докуме размер
область имени грам. управ. поведением
возрора

Возвр: гексиратор области имени (воздан ресурс) ат -1.

char *shmat (int shmid, char *shmaddr, int shmflg)

агресс поименование (указавен = агр. виртуал.
наимено) - агр. нок. с кат. нок. нок. нок. нок.
раздел. наимено (> 0 или = 0)

Возвр: агр., нок. с кат. будет отобратався при сведении
разделения имени / -1.

Открытие ат. разд. имени

int shmctl (char *shmaddr, int cmd)

0 - успех, 1 - неудача

int shmctl(...) - управление

IPC: массив сегментов

^(*)
#include <sys/sem.h>

int semget (key_t key, int nsems, int sem_flg)

идентиф.
ресурса

кол-во
сегментов

- создание

Условно атомарные операции

ли. ошибки

Пример: есть раздв. память. 1 процесс записывает туда
слова, 2 читает и интерпретирует.

Необх. синхронизация

Файловые системы

Структурная организация файлов:

- 1) Файл как послед-ть байтов (массив. модель)
- 2) послед-ть записей переменной длины
- 3) послед-ть записей постоянной длины
- 4) Иерархическая организация (дерево)

Каждый файл имеет набор характеристик, описыв. состояние

файла - атрибуты:

- имя

- права доступа (каким пользователям/категориям польз-б
можно выполнять те или иные действия)

- идентификация

- тип файла (как нужно интерпретировать его содержимое)
файл-каталог как организован файл
- ссылка и как с ним можно работать
- socket
- утилиты

- размер записи/блока (обычно файлы на блоко-ориентирован.

устройствах: блок-записи (диск, лента)

- размер файла

- указатель чтения/записи (относ. какой позиции можно читать/писать: указатель лб статическим - напр. при открытии всегда на начале, динамическим - изменяется)

- статистика: время создания, послед. модификации, послед. обращение; предельный размер файла и т. п.

Основные операции работы с файлами

Начало: открытие файла (регистрация в системе возможности работы процесса с содержимым файла)

// Оптимизация обращений к реал. устр-ву: оно упрощается

⇒ минимизируем обращения



Работа с содержимым и атрибутами файла (создается структура данных, опис. текущ. состояние; создается файловый дескриптор (явл. фактом открытия файла); сеанс-структура данных)

// Файл. дескриптор - системная структура данных, содержащая информацию об актуальном состоянии открытого файла

open, close, read, write, delete, seek (позиционирование указателя на за/запись), rename, read/write-attributes (чт./модиф. атрибутов)



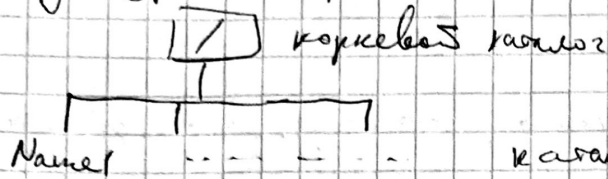
Завершение. Закрывание файла - инд-я системе о завершении работы процесса с открытым файлом

Каталог

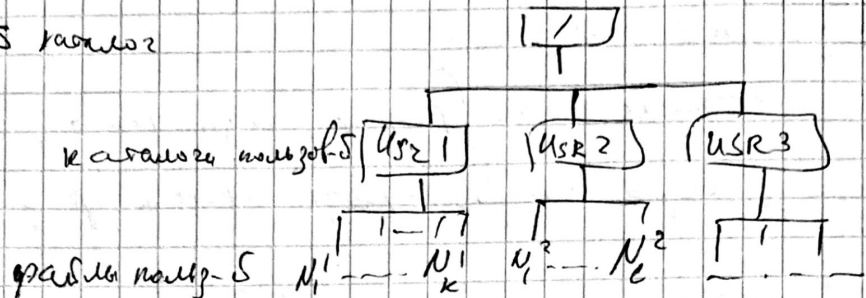
- спец. вид файлов; компонент файловой системы, содержащий информацию о содержимых в файл. системе файлах

Модели

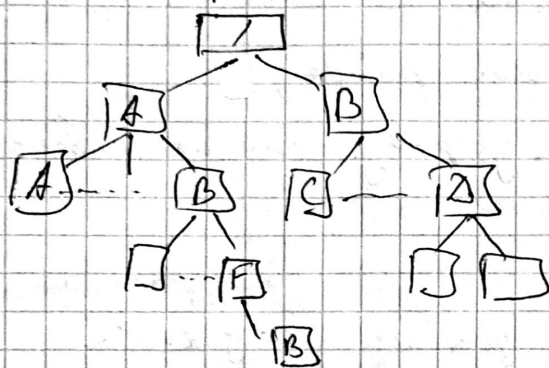
Одноуровневая ф. сист.



Двухуровневая ф. с.



Иерархическая (привычная)



Имя файла! - полное (от корневой кат.)

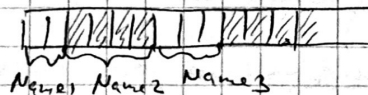
- относительное (от текущей кат.)

Домашний каталог - тот, к-й становится текущим при входе пользователя в сист. номер (∃ файл passwd, в к-ом каждому пользователю соотв. домашн. каталог)

Структура "системного" диска - см. ниже (файловые системы)

Модели реализации файлов

• Непрерывные



+ простая реализация

+ высокая производительность

- фрагментация свободного пространства (место есть, но файл разместить нельзя - нет дост. свобод. места подряд)

- сложно изменить размер файла

• Имеющие организацию внешнего поиска

Кажд. содержательный блок файла содержит pointer на след. блок

+ простая организация каталога (имя файла и N стартового блока)

+ простой последовательный доступ

+ простота реализации

+ отсутствие фрагментации свобод. прост-ва (за счёт блочной фрагментации)

- сложность организации прямого доступа

- фрагментация файла по диску: 1-й блок в начале, 2-й в конце, 3-й в середине $\Rightarrow$ головка диска скачет $\Rightarrow$ износ (не для электронного диска)

- наличие ссылки в блоке $\Rightarrow$ чтение 2-х блоков при необход-ти чтения данных объёмом 1 блок (фрагментация блока)
(Блок: 1020 байт содержательного текста + 4 байта служебной информации, почитай ного 1024)

Таблица размещения файло. информации

Катг. i -й стр. табл. $\leftrightarrow i$ -й блок

в катг. стр. - след. блока

+ эффективный прямой доступ

+ упрощ. а-т проблем связ. списка

- табл. должна размещаться в ОП

0	8
1	5
2	
3	1
4	
5	0

3-1-5

FAT
(File Allocation Table)

Лекция 12

21.11

Произвольное кол-во подключаемых устройств

У кажд. устр-ва свой размер блока (физич.)

Виртуальное устр-во - драйвер = программа, реализующая это вирт. устр-во

Файловая система наход. на системном диске

Структура "системного" диска

Блоки

№0	таблица разделов: начало, фин., нота...				
----	---	--	--	--	--

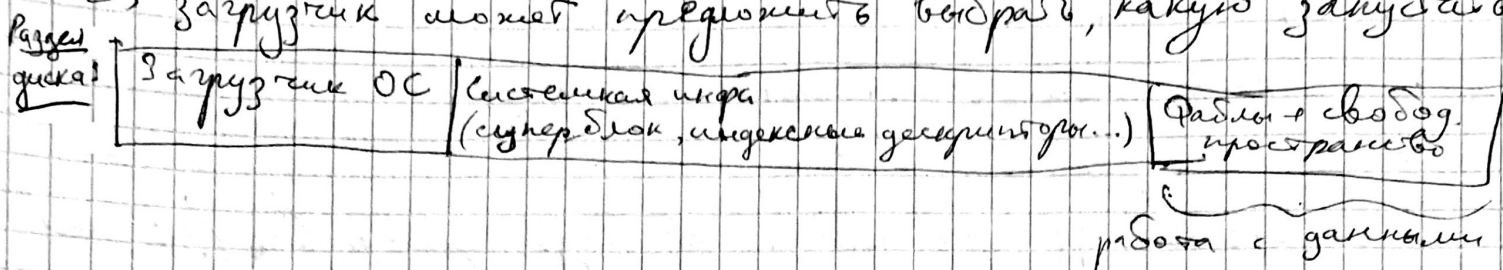
(*) осн. программы загрузки

разделы диска

(*) антивирусная программа запускает эту програ. при включении диска

В каждом разделе лб своя ОС

⇒ Загрузчик может предложить выбрать, какую загрузить



Накладные расходы - имеющие несодержательный обмен
(напр., хотим считать информацию с диска: сам заголовок файла - содержит обмен, всего, что нужно для доступа - несод.)

Табл. размещ. файловых систем

В i -й строке - информация о том, где находится i -ый блок файловой системы

Находится в ОП

Каталог: информация о имени файла и номер блока, в кот. находится первый блок файла (ссылка на голову списка). Далее по табл. FAT перемещаемся по файлу.

Перемещение по FAT-табл. - работа программы ⇒ не накладные расходы! Работа прог. не упрощ. как наклад. расх. - неэффективная реализация для больших дисков, только маленькие + возможность перемещать весь блок для хранения информации

Индексные узлы (дескрипторы)

- системные структуры данных, содержат информацию о размещении

Блок конкрет. файла в файл. системе

- Конечный размер файла ⇒ ограничение размера файла, иерархическая организация индексных узлов

+ нет необход. размещения в ОЗУ информации всей FAT о всех файлах системы, в памяти размещ. атрибуты, связь только с открыт. файлами

Модели организации каталогов

Цель каталога - обеспечить именованность работы с данными

- 1) Запись каталога: фиксир. размера, содержи. имя файла и его атрибуты

Name1	Атрибуты
Name2	---
---	---

- 2) Каталог содержи. имя файла и ссылки на атрибуты, их размер может варьироваться

Name1	Атрибуты
Name2	---
---	---

→ [Атр. 1]
→ [Атр. 2]

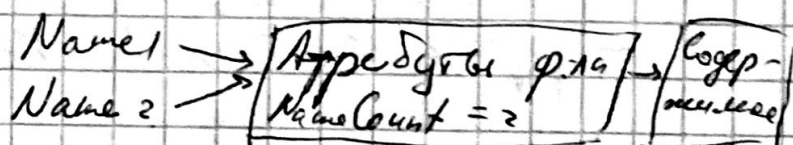
Взаимнооднозначная связь имени и содержимого файла?

- 1) Содержимому файла соотв. единств. имя файла, имени соотв. единств. содержимое

- 2) 1/ каждого содержи. файла мб произвол. кол-во имен (реальность)

2.1) "Жесткая" связь

Равнозначные имена



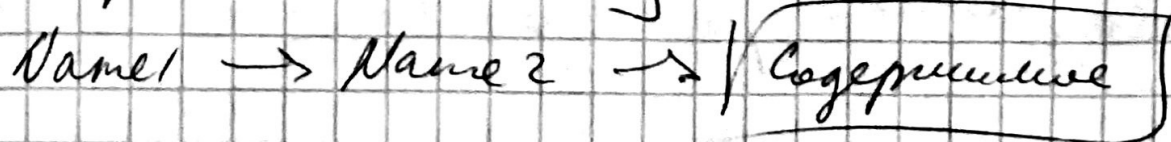
Создаём файл Name1; средствами ОС можем добавить имя файла; в атрибутах мб счётчик имен. При этом того, что содержимое больше не нужно: NameCount=0 ⇒ содержи. удалится из каталога

2.2) "Символическая" (косвенная) связь

Файл - ссылка, на файл, с коб. котым работаем

Удалил файл-ссылку, никак не можем повлиять на содержимое, но оно остаётся

Удалил содержимое, но осталась ссылка ⇒ при открытии - отказ



Размер блока файловой сист. (?)

Большой: + эффективный обмен

- существенная внутренняя фрагментация (не эфф. использование пространства вкеш. памяти)

Маленький: + эфф. ищ. пр-ва вкеш. памяти

- фрагментация по устройству

Проблема: определение оптимального размера

Учет свободных блоков гр. сист.

Вычислять их - трудоёмко $\Rightarrow$ хранить карту

Связь, список свобод. блоков (табл.: перечень всех св. блочков)



В ОЗУ - первый блок списка

+ простая организация

- фрагментация по диску (возв. в процессе работы)

Битовые массивы

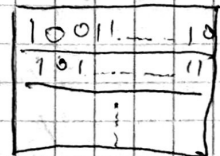
Составл. в соотв. блокам битовый массив

Состояние блока определяется содержанием бита

с соотв. номером: блок свободен $\Rightarrow$ бит = 1, занят $\Rightarrow$ 0

+ можем искать наиболее близкое свобод. пр-во $\Rightarrow$

минимизир. фрагментация диска



Современная ОС многопользовательская $\Rightarrow$ конкуренция, необход. координация доступа к ресурсам

Итоговая ситуация: забита вся память; открыто так возможное кол-во файлов $\Rightarrow$ невозможно работать

Решение: выделить каждому пользователю отдельное пространство. Проблема: 1 не использует пр-во, другому не хватает $\Rightarrow$ не может работать $\Rightarrow$ выделить группе пользователей $\Rightarrow$ тоже проблемы.

Квотирование пр-ва файла системы

1) Жёсткая квота: нельзя пересечь ни при каких обстоятельствах. Превыш. $\Rightarrow$ пользователи блокируются

Квота — кол-во ресурса

2) Гибкая квота: при превышении вкл. счётчик предупреждений (штрафов).

До гибкой квоты: обычная работа

Превыш. гибк квоты: из счётчика пред-б вычит. 1, если $\neq 0 \Rightarrow$ предупреждение: освободить ресурс.

$= 0 \Rightarrow$ превышение жёсткой квоты $\Rightarrow$ блокировка.

Освобождение ресурса $\Rightarrow$ счётчик в начальное значение

Надёжность ф. сист.

Потеря информации: 1) сбой аппаратуры, 2) ошибка программного обеспечения, 3) случайное удаление файлов

Ущерб: 1) потеря системных данных

2) потеря данных конкретного пользователя

$\Rightarrow$ Резервное копирование

1) С некоего момента времени долгая (много файлов) $\Rightarrow$ выборочное копирование: не нужно копировать то, что уже скопировано, объектные файлы (будут получены при запуске); избирательность архивирования по типам файлов

Долго и дорого

2) Инкрементное архивирование: сначала создается полная копия, все последующие вы только обновленные графы. Расписание: копировать раз в день/неделю/мес.
Мастер-копия: дата и время ее создания $\Rightarrow$ копирование всех графов, кот. были созданы/изменены после ее создания.

Маленькие копии изменений: потеря игры $\Rightarrow$ накладываем на мастер-копию последовательных изменений $\Rightarrow$ восстановление.
Мастер-копия - первое копирование

3) Использование компрессии: хранение статей версии + многие типы графов сжимаются + со статьями архивами проще работать - статьи архивы крайне чувствительны к потере игры

4) Проблема архивирования "на ходу": изменение/создание/удаление графов во вр. копирования (компания: система должна работать постоянно $\Rightarrow$ нельзя приостановить)

5) Распределенное хранение резервных копий

Стратегии архивирования

1) Физическая архивация

Копирование блоков (используемых)

2) Логическая арх.

Копир. графов, а не блоков, модифицир. после заданных дат

Проверка целостности ФС

Контроль непротиворечивости блоков ФС

Занятые блоки: графы, сыгравшая игра
Свободен/занят

Контроль занятости: табл занятых блоков (создание таблиц) табл свободных блоков

Число записей канд. табл. = числу блоков ФС

Извест все записи табл. обнуляются

⇓

Анализ табл. свобод. блоков (Список свобод. бл. $\Rightarrow +1$ к свобод. записи табл.)

⇓

Анализ индексных узлов. Для канд. блока, встретившегося в индексном узле, увеличить его счётчик на 1 в табл. зан. бл.

⇓

Сопоставление

Несоответствие $\Rightarrow$ считаем, что блок свободен

1	1	0	1	0	1
---	---	---	---	---	---

 7. зан.
 $\updownarrow$

0	1	0	1	1	0
---	---	---	---	---	---

 7. св.

$\Rightarrow$ потеря блока (в обеих табл. 0 или 1)

Знач. в табл. св. бл. $> 1 \Rightarrow$ дубликат св. блока $\Rightarrow$ пересоздание списка свобод. блоков

0	1	0	1
---	---	---	---

1	0	2	1
---	---	---	---

Знач. в т. зан. бл. $> 1 \Rightarrow$ индексн. дескрипторы не корректны: дубликат занесённого блока

0	2	1	1	0
---	---	---	---	---

1	0	0	1	1
---	---	---	---	---

Решение: 2 копии графов, удаление ориентиров, разнесение дескрип. в 2 копии, пересопределение списка свобод.

блоков, обратное переименование графов и фиксации графов их возможных проблемности

Проверка счётчика имен

1) $L = M$ - коррект.

2) $L > M$

3) $L < M$

$\Rightarrow$ NameCount = L

Имя₁ $\rightarrow$

Алг. Ф-лы
NameCount = M

 $\rightarrow$ Код.
Имя₂ $\rightarrow$
M - кол-во вершин имен в ФС

Лекция 13

Лекция 14

5.12

Простота - одно из важнейших св-в: экономически дешевле,
скорее всего надежнее сложной вещи, проще реализовать

Сравнительная ОП

Таблица сравнения - отображение номеров виртуал. сор "с"
номера физических

Размер табл. сор. - проблема:

1) накладные расходы преобразования адреса, т.к. табл
не на быстрых регистрах $\Rightarrow$ мед. доступ к ОП

2) много обрабатываемых процессов $\Rightarrow$ много копий $\Rightarrow$
какие табл. держать в памяти, какие нет? Все реально,
слишком много, несколько можно

TLB - таблицы - табл. быстрого преобраз. адресов

2 потока интр.-данных (с вычислениями операндов)
- команд (образуются в ходе выполнения)

$\Rightarrow$ Поиском адресов команд и данных не всегда
Компилируются наиб. часто используемая хранятся в регистровой
буфере

в какой-то
статусе (работает
по более медленной
принципу)

TLB - тоже кэш; регистровая табл., т.е. находится
внутри процессора, скорость работы с ней сопоставима
со скоростью работы процессора.

Внутри процессора реализован автоматический параллель-
ный поиск вирт. номера по табл. Найден $\Rightarrow$

берём физ. номер $\Rightarrow$ заменяем на него виртуальный
 $\Rightarrow$ реальный адрес

Не нашли $\Rightarrow$ обращение к ОС (в общ. случае
можно говорить, что происх. прерывание) $\Rightarrow$ начинается
медленный этап, т.к. начинается работа улога

то всели в высказывании. Ищем нужную физич. стр. в таблице стр., кот. наход. в ОП, обновлен TLB Если нет в табл. стр. в ОП, управл. передат. др. процессу

сокращение накладных расходов за счёт локализации адресов

Размер TLB статистически рассчитывается (Решим 1) проблему)

Проблема 2)

Многоуровневые (иерархические) табл. стр.

$\boxed{VP | offset}$ $\rightarrow$ $\boxed{FP | offset}$ $V_{\text{вирт. памяти}} = 2^{32} \text{ байт}$
 $\underbrace{20}_{\text{данный}} \quad \underbrace{12}_{\text{разряды}}$
 $\Rightarrow \boxed{VP1 | VP2 | offset}$ $V_{\text{кэш}} = 2^{12} \text{ байт}$
 $\underbrace{10}_{\text{1-й уровень}} \quad \underbrace{10}_{\text{2-й уровень}} \quad \underbrace{12}_{\text{3-й уровень}}$
 $\Rightarrow 2^{10} \cdot 2^{10} \cdot 2^{12} = 2^{32} = V_{\text{вирт. памяти}}$
 $\Rightarrow$ дан. табл. (т. первого ур.)

Для $VP1$ - внешняя табл. стр., $VP2$ - номер табл. 2-го уровня в кэш (кол-во стр. - 2^{10})

$VP2$ - смещение по табл. 2-го уровня; в этой стр. - физ. номер

$V_{\text{вирт. памяти}} = 2^{32} \text{ байт} \Rightarrow \text{кол-во вирт. стр. } 2^{20}$
 $V_{\text{стр}} = 2^{12}$

Тем это лучше? Не обязательно держать в буфере все табл. 2-го уровня. В буфере держится N стр., они меняются (работает как кэш)

Нужно посчитать оптимальн N

Мб 2-уровневая 3-уровневая, n -уровневая табл. (больше 3 обычно неразумно).

Хэш-табл.

Хэш-ф-я совпа в соотв. VP некое адрес
МБ коллизия! (не взаим. однознач. отображение)

Инвертирован табл. ср.

Архитектура: $\boxed{PID | VP | offset}$

(процесса, которому в ср

i-я ср табл. содержит инд-у: какая табл. какой процесс
находится в этой памяти (т.е. № — физич. № ср.)

④ уходим от проблемы смены таблиц при смене контекстов
// кол-во записей в этой табл. = кол-во ср. физич. памяти

⑤ поиск соответствия

⇒ Алгоритмы поиска, основ на хэшировании, инвер. и т.д.

Замена ср.

Какую ср. выкинуть, чтобы загрузить новую?

Алгоритм NRU (Not Recently Used)

2 аппаратных (битовых) признака: обращение ^R
и извлечение ^M ср. Обнуляются программно по
таблице (все признаки для всех физ. ср.)

Категории ср: 0: $M=0, R=0$
(масс)

1: 1 0

2: 0 1

3: 1 1

⑦ запись: $M=1, R=1$

⑧ разные таблицы

Выбор ср. для удаления из класса с наим. номером

Алгоритм FIFO

Удаляется самая старая ср.

Несправедливо!

⇒ Модификация алгоритма: $R=0 \Rightarrow$ замена ср,

$R=1 \Rightarrow R=0$, обновление времени загрузки ср. в памяти
(т.е. перенос в конец очереди)

Диз. NFU (Not Frequently Used)

счит.; - счетчик обращений $\Rightarrow$ хранить

статистику обращений $count_i + 1 = R_i$

⊖ помним старые обращения

⊕ мб переполнение счетчика при большой активности

$\Rightarrow$ Модификация: знат. $count_i$ сдвига на 1 разряд вправо, знат. R добавляется в крайний левый разряд счетчика (мб потеря старш. разряда - не страшно)

Сегментное распр.

Браун: логические области памяти: стек, куча и т. п.

Отображение на одно адресное пр-во не от мб пересечение

В пр. адр. пр-во представляется в виде неуп. совокупности сегментов.

Кажд. сегмент имеет свою вирт. адресацию от 0 до $N-1$ (размер сегмента N - мб разное)

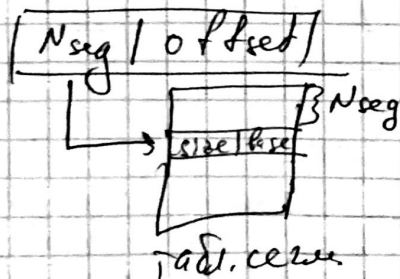
Вирт. адрес $\langle N^{\circ} \text{сегмента, смещение} \rangle$

Необх. табл. сегментов

$\Rightarrow$ строится по аналогии с табл. стр.

i -я стр. - описывает i -й сегмент, содержит

хар-ки сегмента: физич. адрес начала размещения сегмента в памяти и размер сегмента



$offset > size \Rightarrow$ выход за пределы сегмента $\Rightarrow$ прерывание

$\leq \Rightarrow base + offset$ - физич. адр.

⊖ про это

⊖ все сегм. данные хран. в ОП, а они мб большие $\Rightarrow$ трата времени

Безменного - стратегическая задача

Каждое семейство разделилось на стр
// компромисс 2 предыдущих способов организации

N_{seg}	N_{page}	Offset
-----------	------------	--------

Модель современных компьютеров

Лекция 16

19.12

Создание сока

```
int socket (int domain, int type, int protocol);
```

fd = socket (---) - файловый дескриптор, создан

Визуализация процесса $\Rightarrow$ нужно как-то измерять, чтобы

к чему может привести процесс → б/д
int bld (int sockd, struct sockaddr *myaddr, int addrlen)

ist Blut (ist sockel, strukt sockadre ~~mayadlr~~, ist ~~adlr~~)

дескриптор,
т. е. Vol (получен)

член, к-го хотели ассоциировать с этим советом

длина
чисел

Вызывание — создание грабля типа `socket` (при вызывании не требуется указывать такого имени)

Модель взаимодействия с предварительным установлением соединения

тип сожета - виртуальный канал или датаграмма

Был бы аналог — телеграфный разговор

Клиент - серверные системы

Клемент

Сергей

socket() - создание сокета - socket()

Word() - изменение кода;
т.к. устанавливает
виртуал. канал

bedr(!)!

connect() - запрос на установление
соединения (ссылка вызов)
(по listen в сервере - ответ)
(если ответ зашла - ответ)

1. sten() - процес добиване на
сокет готов за обработка
запроси, устанав.
огреда на својствома

wh/e(-) - ^{ини} обработка загроов

send / recu

↓

shutdown()

close()

accept()

закрывает, если не
закрыто, до его за-
крытия,
создается socket, связ
с дескриптор клиента

send/recv

работает почти как
каналов. каналом

shutdown()

close()

если у 5 клиентов, у сервера 6 socketов
fork, exec после accept: т.к. у каждого клиента -
процесса свой ресурс; в оригинале-сервере - закрываем
(необяз.)

int send(--), int recv(--), аналог read/write

Модель без установления предварительного соединения
Безопасный аналог - telnet

проблем. ест. от предыдущих: в каждом сообщ. нужно
писать кому и от кого, т.к. кет сеанса связи
адреса должны существовать!

всегда именуем socket

int sendto(--)

int recvfrom(--)

Завершение работы с socketом
↓
мягкое жесткое

Обобщение курса

И клиент, и вышняя система - системы, т.е. ^{мн.во} компьютеры,
работающих, как единое целое.
Работа зависит от самого слабого компьютера
Вызывает ≠ клиент !

Дисбаланс: "медлен" ОП - "быстрые" ЦП

Уметь показывать, как бороться с этим

Ресурс: осн. св-во - правила программного использования
нет правил - не ресурс

Виртуальность, виртуальный ресурс

Дисбаланс: скорость поступления обработки запросов $\Rightarrow$

$\Rightarrow$ формирование очереди $\Rightarrow$ задача планирования

процессы: взаимодействие, взаимное исключение

① Семантика дедексы - формальная модель
формальное определение

Примеры межпроцессного взаимодействия: IPC, сокеты

Работает система: FFS не обсуждали! ①

Большие накладные расходы к другим блокам
(как можно сгладить?)

Управление внеш. устр: поток управления^(управл. команды), поток
данных (операнды)

Управление памятью: аппаратно-программный компонент

Если есть аппарат границей памяти - её можно управлять

Реал. модель: сегментно-страничная память