

СПИСОК ВОПРОСОВ ПО КУРСУ “ОПЕРАЦИОННЫЕ СИСТЕМЫ” 2020/2021г. г.

1. Этапы развития вычислительной техники и программного обеспечения.

Первое поколение компьютеров

Элементная база	электронно-вакуумные лампы
Временной период	середина 1940-х – вторая половина 1950-х годов

Середина 40-х годов - Пенсильванском университете США была разработана вычислительная машина ENIAC (Electronic Numerical Integrator and Computer), которая считается одной из первых электронных вычислительных машин — ЭВМ.

Особенности

- однопользовательский, персональный режим
- зарождение класса сервисных, управляющих программ
- зарождение языков программирования

Приоритетное направление: военные задачи

Второе поколение компьютеров

Элементная база	полупроводниковые приборы: диоды и транзисторы
Временной период	вторая половина 1950-х – вторая половина 1960-х годов

Особенности

- пакетная обработка заданий
- мультипрограммирование
- языки управления заданиями
- файловые системы
- виртуальные устройства
- операционные системы

Приоритетное направление: управление бизнес-процессами

Третье поколение компьютеров

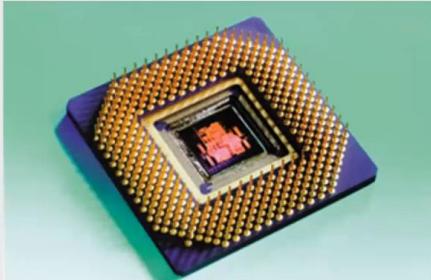
Элементная база	интегральные схемы
Временной период	конец 1960-х – начало 1970-х годов

Особенности

- аппаратная унификация узлов и устройств
- создание семейств компьютеров
- унификация компонентов программного обеспечения

Четвёртое поколение компьютеров

Элементная база



Первый микропроцессор Intel-4004 (1971г.) - 2250 элементов.

Первый универсальный микропроцессор Intel-8080 (1974г.) - 4500 элементов (основа для создания первых ПК).

16-битный микропроцессор Motorola-68000 (1979 г.) - 70 000 элементов.

Первый 32-битный микропроцессор Hewlett Packard (1981 г.) - 450 тыс. элементов.

Четвёртое поколение компьютеров

Элементная база	большие и сверхбольшие интегральные схемы
Временной период	начало 1970-х – настоящее время

Особенности

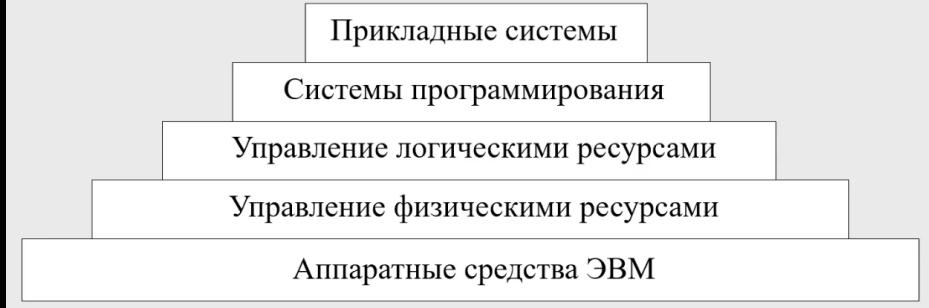
- «дружественность» пользовательских интерфейсов
- сетевые технологии
- безопасность хранения и передачи данных

2. Структура вычислительной системы. Ресурсы ВС - физические ресурсы, виртуальные ресурсы. Уровень операционной системы.

Основы архитектуры вычислительной системы

Вычислительная система — совокупность аппаратных и программных средств, функционирующих в единой системе и предназначенных для решения задач определенного класса.

Структура вычислительной системы:



Аппаратный уровень вычислительной системы

Характеристики физических ресурсов (устройств)

- правила программного использования
- производительность и/или емкость
- степень занятости или используемости

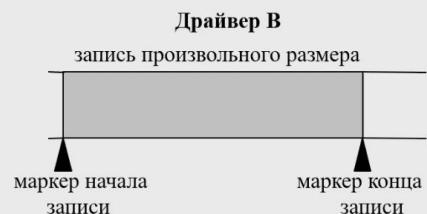
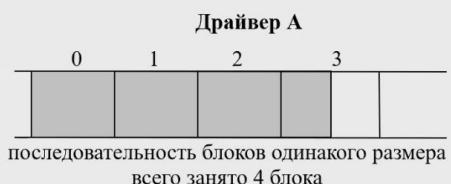
Средства программирования, доступные на аппаратном уровне

- система команд компьютера
- аппаратные интерфейсы программного взаимодействия с физическими ресурсами

Управление физическими ресурсами вычислительной системы

систематизация и стандартизация правил программного использования физических ресурсов
(уровень драйверов физических устройств)

Драйвер физического устройства — программа, основанная на использовании команд управления конкретного физического устройства и предназначенная для организации работы с данным устройством.

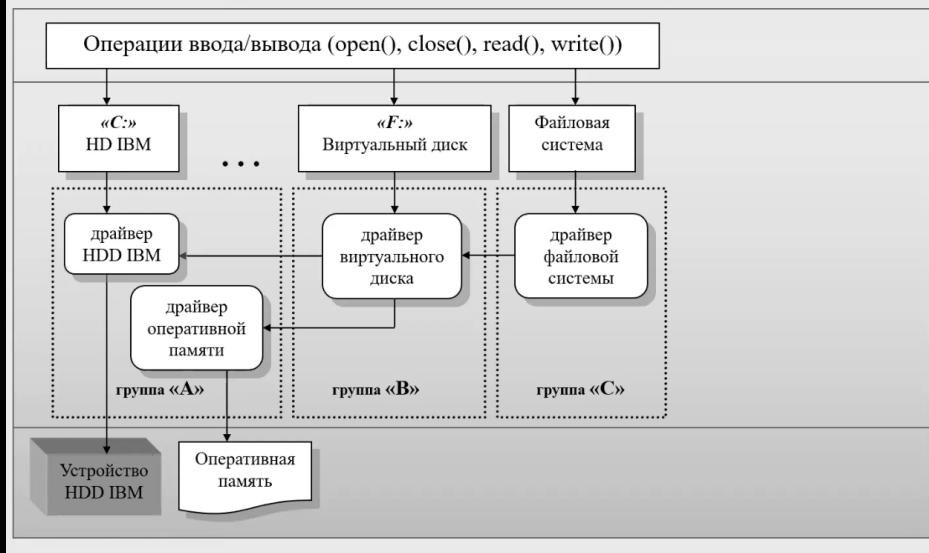


Управление логическими (виртуальными) ресурсами

Логическое (виртуальное) устройство (ресурс) — устройство (ресурс), некоторые эксплуатационные характеристики которого (возможно все) реализованы программным образом.

Драйвер логического (виртуального) ресурса — программа, обеспечивающая существование и использование соответствующего ресурса.

Управление логическими (виртуальными) ресурсами



Управление логическими (виртуальными) ресурсами

Характеристики ресурсов вычислительной системы

Ресурсы вычислительной системы — совокупность всех физических и виртуальных ресурсов.

Одной из характеристик ресурсов вычислительной системы является их **конечность** - возможна конкуренция за обладание ресурсом между его программными потребителями.

Операционная система — это комплекс программ, обеспечивающий управление ресурсами вычислительной системы.



Управление логическими (виртуальными) ресурсами

Средства программирования, доступные на уровнях управления ресурсами ВС

- система команд компьютера
- программные интерфейсы драйверов устройств (как физических, так и виртуальных)



3. Структура вычислительной системы. Ресурсы ВС - физические, виртуальные. Уровень систем программирования.

Системы программирования

Система программирования — это комплекс программ, обеспечивающий поддержание жизненного цикла программы в вычислительной системе.

Жизненный цикл программы в вычислительной системе

1. Проектирование
2. Кодирование
3. Тестирование и отладка
4. Ввод программной системы в эксплуатацию (внедрение) и сопровождение



Системы программирования: история

Начало 50-х годов XX века	Система программирования или система автоматизации программирования включала в себя ассемблер (или автокод) и загрузчик, появление библиотек стандартных программ и макрогенераторов.
Середина 50-х – начало 60-х годов XX века	Появление и распространение языков программирования высокого уровня (Фортран, Алгол-60, Кобол и др.). Формирование концепций модульного программирования.
Середина 60-х годов – начало 90-х XX века	Развитие интерактивных и персональных систем, появление и развитие языков объектно-ориентированного программирования.
90-е XX века – настоящее время	Появление промышленных средств автоматизации проектирования программного обеспечения, CASE-средств (<i>Computer-Aided Software/System Engineering</i>), унифицированного языка моделирования UML.



4. Структура вычислительной системы. Ресурсы ВС - физические ресурсы, виртуальные ресурсы. Уровень прикладных системы.

Прикладные системы

Прикладная система — программная система, ориентированная на решение или автоматизацию решения задач из конкретной предметной области.



Выводы

Пользователь и уровни структурной организации ВС

Прикладные системы	+ набор функциональных средств прикладной системы.
Системы программирования	+ трансляторы языков высокого уровня, библиотеки...
Управление логическими (виртуальными) устройствами	+ интерфейсы драйверов виртуальных устройств.
Управление физическими устройствами	+ интерфейсы драйверов физических ресурсов
Аппаратные средства	Система команд, аппаратные интерфейсы программного управления физическими устройствами



5.??? Структура вычислительной системы. Понятие виртуальной машины.

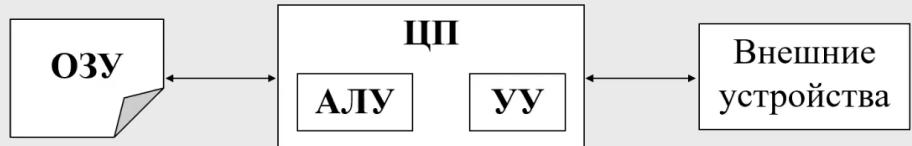
Компьютер фон Неймана

Историческая справка

- Джон фон Нейман (John Von Neumann)
- EDVAC (Electronic Discrete Variable Computer — Электронный Компьютер Дискретных Переменных)
- «Предварительный доклад о компьютере EDVAC» (A First Draft Report on the EDVAC)
- Джон Мочли (John Mauchly) и Джон Преспер Эккерт (John Presper Eckert)
- ENIAC (Electronic Numerical Integrator And Computer)

Компьютер фон Неймана

Структура, основные компоненты компьютера фон Неймана



Принципы построения компьютера фон Неймана

1. Принцип двоичного кодирования
2. Принцип программного управления
3. Принцип хранимой программы

6. Основы архитектуры компьютера. Основные компоненты и характеристики. Структура и функционирование ЦП.

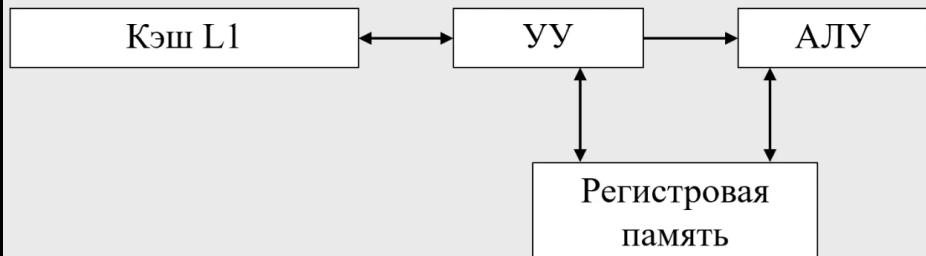
Центральный процессор

Устройство управления (control unit) — координация выполнения команд.

Арифметико-логическое устройство (arithmetic/logic unit) — выполнение команд, арифметической или логической обработки operandов.

Регистровая память (register memory) — совокупность устройств памяти процессора - регистров.

Кэш-память (cache memory) — буферизация работы процессора с оперативной памятью.



Центральный процессор

Регистровая память

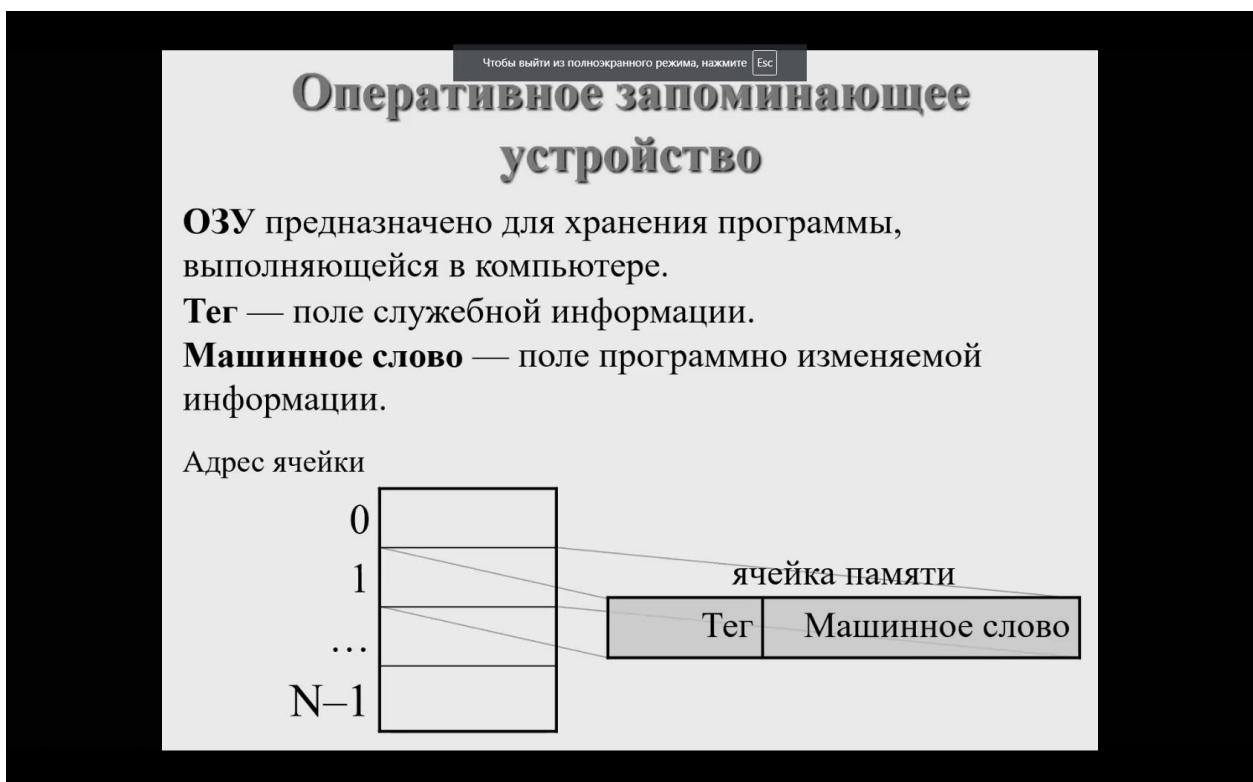
Регистровая память
(регистровый файл)

Регистры общего
назначения (РОН)

- Специальные регистры:
- **счетчик команд** (program counter)
 - **указатель стека** (stack pointer)
 - **слово состояния процессора** (processor status word)
 -



7. Основы архитектуры компьютера. Основные компоненты и характеристики. Оперативное запоминающее устройство. Расслоение памяти.



Оперативное запоминающее устройство

Использование содержимого поля служебной информации

1. Контроль за целостностью данных

При записи слова в память контрольная сумма бит = 9 (1001b) \Rightarrow тег = 1.

1 0 1 0 1 0 1 0 1 0 1 0 1 1 1 0 1	Ошибки нет
---	------------

При чтении машинного слова 16 бит вычисляется тег
сумма бит = 9 (1001b) и сверяется со значением тега.

1 0 1 0 1 0 1 0 1 0 1 0 1 1 0 0 1	Ошибка
---	--------

При чтении машинного слова 16 бит
вычисляется сумма бит = 8 (1000b), а тег = 1 \Rightarrow Сбой в работе ОЗУ

1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1	Ошибка не видна
---	-----------------

При чтении машинного слова 16 бит
вычисляется сумма бит = 7 (111b), и тег = 1 \Rightarrow Ошибка будет не выявлена

Пример контроля за целостностью данных по четности

Оперативное запоминающее устройство

Использование содержимого поля служебной информации

2. Контроль доступа к командам/данными

3. Контроль доступа к машинным типам данных

Оперативное запоминающее устройство

Производительность оперативной памяти — скорость доступа процессора к данным, размещенным в ОЗУ.

- **Время доступа (access time, t_{access})** — время между запросом на чтение слова из оперативной памяти и получением содержимого этого слова.
- **Длительность цикла памяти (cycle time, t_{cycle})** — минимальное время между началом текущего и последующего обращения к памяти.

$$(t_{cycle} > t_{access})$$

Оперативное запоминающее устройство

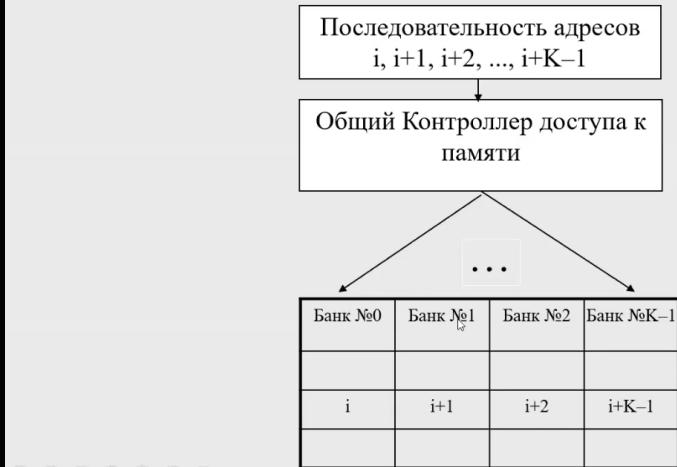
Расслоение памяти

К независимых банков памяти, где $K = 2^L$.



Оперативное запоминающее устройство

Расслоение памяти



Оперативное запоминающее устройство

Расслоение памяти

Сравнение времени доступа

Последовательное чтение машинных слов по адресам:	ОЗУ без расслоения	ОЗУ с расслоением
A	$\sim t_{cycle}$	$\sim t_{access}$
A+1	$\sim t_{cycle}$	$\sim t_{access}$
A+2	$\sim t_{cycle}$	$\sim t_{access}$
Суммарное время:	$\sim 3*t_{cycle}$	$\sim 3*t_{access}$ $\sim t_{access}$

Схема работы



8. Основы архитектуры компьютера. Основные компоненты и характеристики. Кэширование ОЗУ.

Центральный процессор

Использование кэш-памяти (cache memory)

- Сокращается количество обращений к ОЗУ
- Существенно увеличивается скорость доступа к памяти в случае использования ОЗУ с «расслоением»
- Усложнение логики процессора



Центральный процессор

Кэш-память (cache memory) первого уровня (L1)

1. Обмен данными между кэшем и оперативной памятью осуществляется **блоками** фиксированного размера
2. **Адресный тег** блока — содержит служебную информацию о блоке (соответствие области ОЗУ, свободен/занят блок, ...)
3. Нахождение данных в кэше — **попадание (hit)**. Если искомых данных нет в кэше, то фиксируется **промах (cache miss)**
4. При возникновении промаха происходит обновление содержимого кэша — **вытеснение**. Стратегии вытеснения:
 - случайный выбор блока
 - вытеснение наименее «популярного» блока (LRU — Least-Recently Used)
5. Вытеснение кэша данных:
 - **сквозное кэширование** (write-through caching)
 - **кэширование с обратной связью** (write-back cache) — тег модификации (dirty bit)

9. Основы архитектуры компьютера. Аппарат прерываний. Последовательность действий в вычислительной системе при обработке прерываний.

Аппарат прерываний

Прерывание — событие в компьютере, при возникновении которого в процессоре происходит предопределенная последовательность действий.

Типы прерываний

- **Внутренние** — инициируются схемами контроля работы процессора
- **Внешние** — события, возникающие в компьютере в результате взаимодействия центрального процессора с внешними устройствами

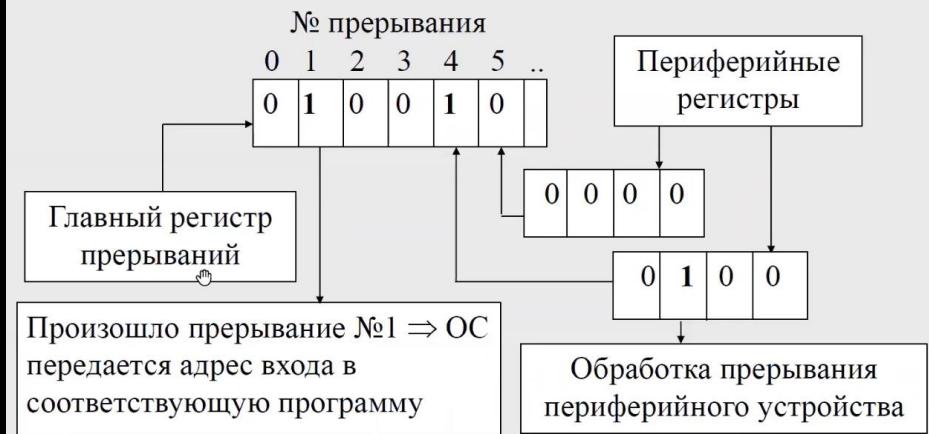
Аппарат прерываний

Этап аппаратной обработки прерываний



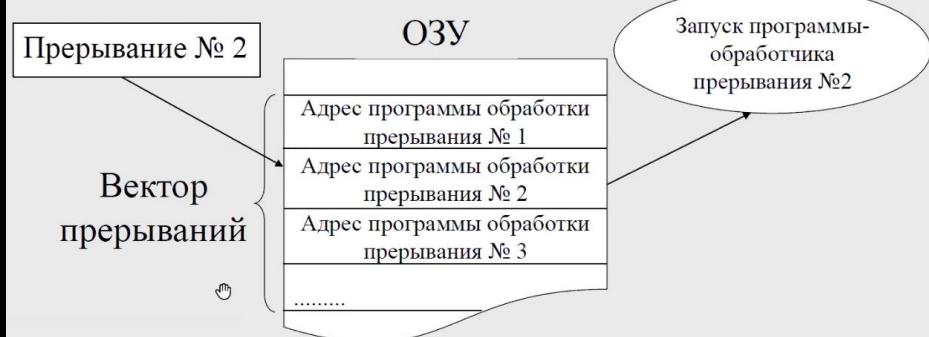
Аппарат прерываний

Модель организации прерываний с использованием «регистра прерываний»



Аппарат прерываний

Модель организации прерываний с использованием «вектора прерываний»



Аппарат прерываний

Модель организации прерываний с использованием регистра «слово состояния процессора»



Аппарат прерываний

Модели организации прерываний

1. С помощью регистра прерываний

Имеется специальный регистр прерываний, в определённые биты которого при возникновении прерываний заносятся «1».

2. С помощью вектора прерываний

В фиксированной области памяти находится вектор адресов обработчиков прерываний.

3. С помощью слова состояния процессора

Имеется специальный регистр, в котором хранится состояние процессора. При возникновении прерывания в определённую часть данного регистра заносится код прерывания.



10. Основы архитектуры компьютера. Внешние устройства. Организация управления и потоков данных при обмене с внешними устройствами.



Внешние устройства

Внешние запоминающие устройства (ВЗУ)

Обмен данными:

- записями фиксированного размера — **блоками**
- записями произвольного размера

Доступ к данным:

- операции чтения и записи (жесткий диск, CD-RW)
- только операции чтения (CD-ROM, DVD-ROM, ...)

Последовательного доступа:

- Магнитная лента

Прямого доступа:

- Магнитные диски
- Магнитный барабан
- Магнито-электронные ВЗУ прямого доступа

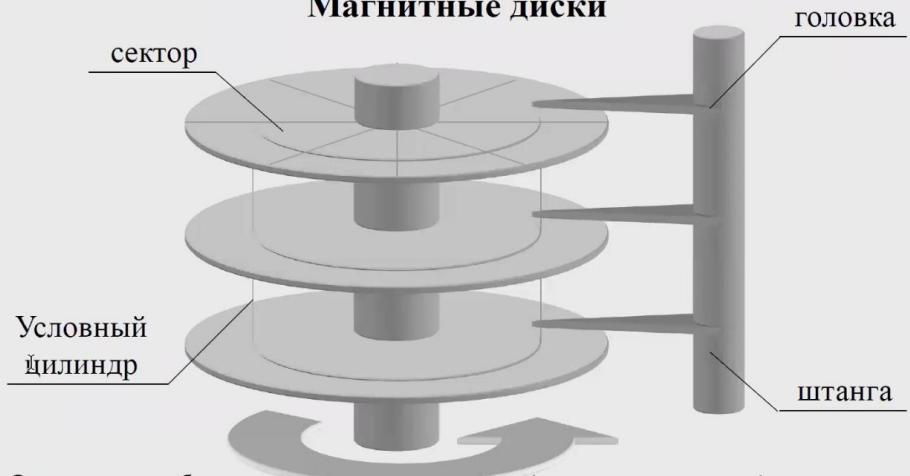
Устройство последовательного доступа

Магнитная лента



Устройство прямого доступа

Магнитные диски

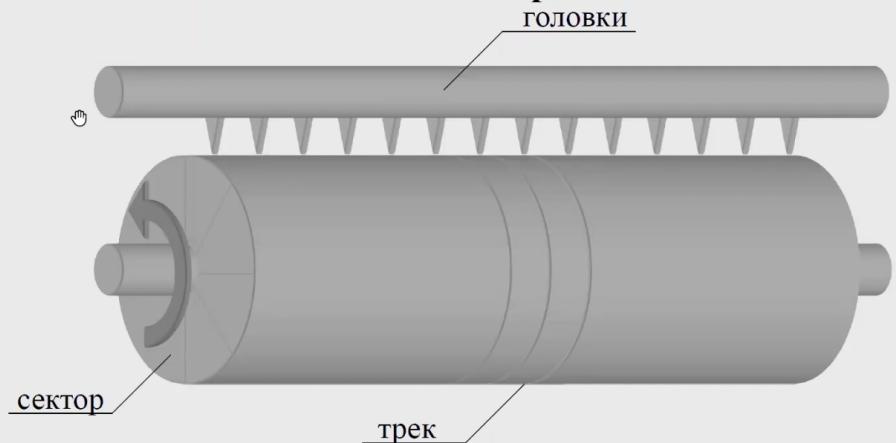


Операции, необходимые для начала чтения (позиционирования)

1. Установка головки на требуемую дорожку
2. Поворот для совмещения головки с началом сектора

Устройство прямого доступа

Магнитный барабан



Операции, необходимые для начала чтения (позиционирования)

1. Поворот для совмещения головки с началом сектора

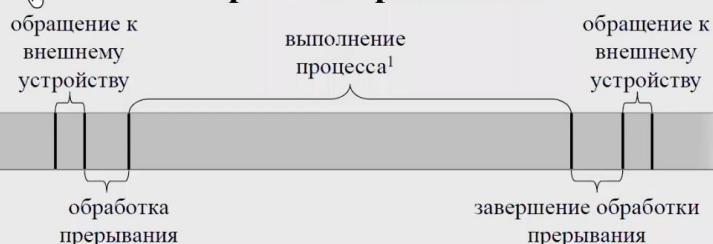
Модели синхронизации при обмене с внешними устройствами

Синхронная организация обмена

обращение к внешнему устройству приостановка выполнения программы, ожидание завершения обмена завершение обмена с ВУ



Асинхронная организация обмена



¹ Примечание: процесс выполняется до возникновения следующего прерывания

11. Основы архитектуры компьютера. Иерархия памяти.

Иерархия устройств хранения информации

ЦП:

РОН

КЭШ L1

КЭШ L2

ОЗУ

ВЗУ прямого доступа с внутренней кэш буферизацией (оперативный доступ к данным)

ВЗУ прямого доступа без кэш буферизации (оперативный доступ к данным)

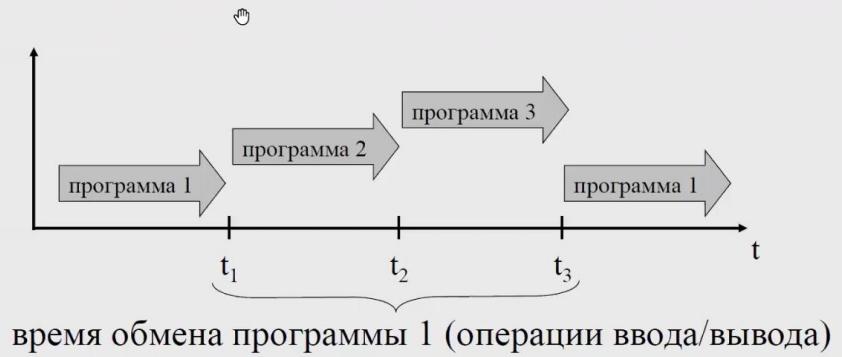
ВЗУ долговременного хранения данных (архивы, резервные копии...)

Увеличение ёмкости
Увеличение времени доступа
Уменьшение скорости чтения/записи
Увеличение времени хранения информации

12. Аппаратная поддержка ОС. Мультипрограммный режим.

Аппаратная поддержка ОС и систем программирования

Мультипрограммный режим — режим, при котором возможна организация переключения выполнения с одной программы на другую.



Базовая аппаратная поддержка мультипрограммного режима

1. Аппарат защиты памяти
2. **Специальный режим операционной системы**
(привилегированный режим или режим супервизора)
3. Аппарат прерываний (как минимум, **прерывание по таймеру**)

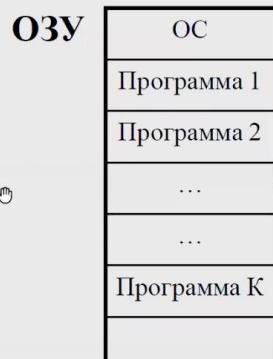
Аппаратная поддержка программных систем и мультипрограммного режима

Проблемы:

- Вложенные обращения к подпрограммам
- Накладные расходы при смене обрабатываемой программы
- Перемещаемость программы по ОЗУ
- Фрагментация памяти



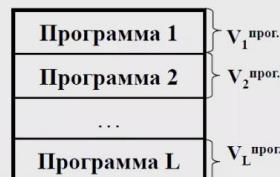
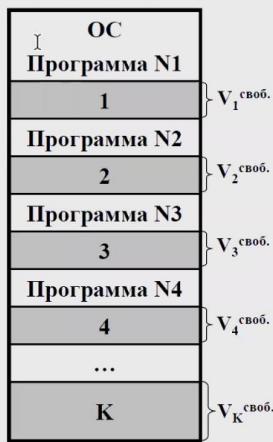
Перемещаемость программы по ОЗУ



Буфер
программ,
ожидающих
начала
обработки

Соответствие адресов, используемых в программе,
области ОЗУ, в которой будет размещена данная
программа

Фрагментация памяти



Буфер программ,
ожидающих начала
обработки

Проблема фрагментации: $V_i^{\text{прог.}} > V_j^{\text{своб.}}$
для $\forall i, j$ (несмотря на то, что $\exists \{i\}$):
 $\sum_{j=1}^K V_j^{\text{своб.}} > V_i^{\text{прог.}}$) \Rightarrow деградация системы

Несмотря на то, что имеется
достаточное количество места в памяти,
разместить ни одну из программ не
удаётся.

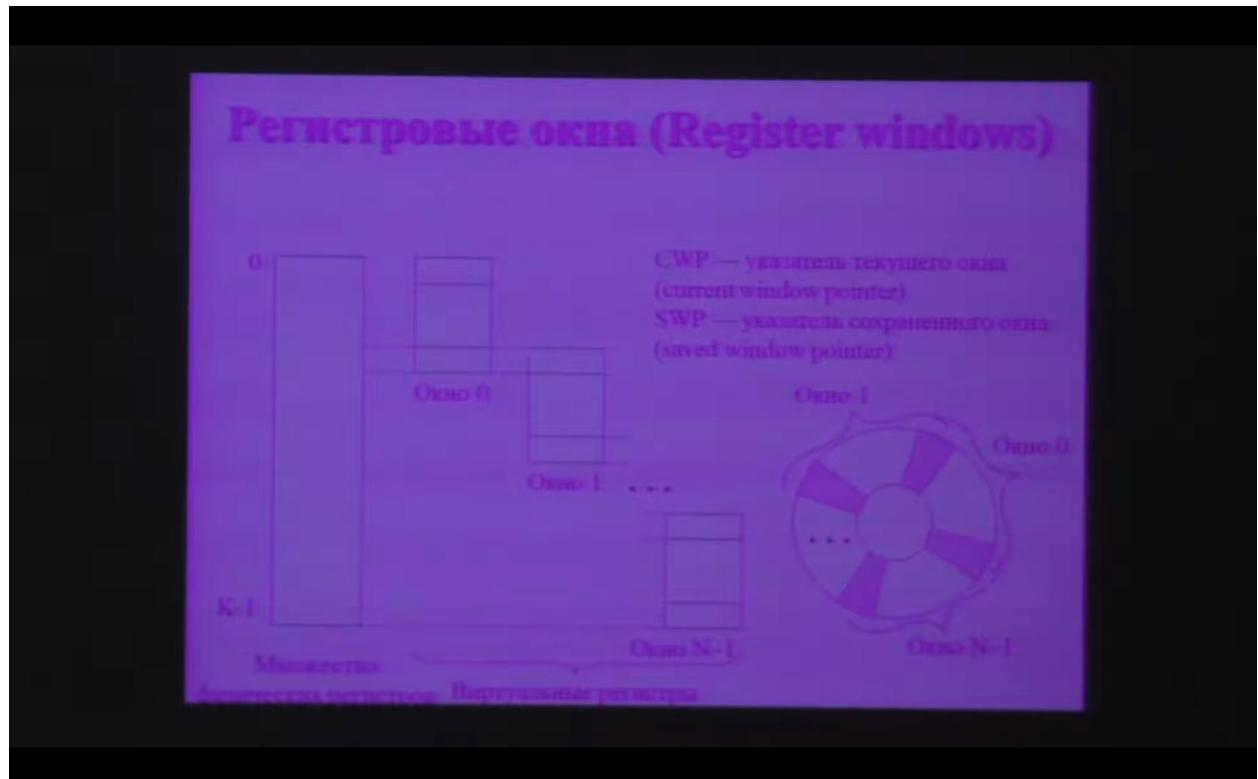
13???. Аппаратная поддержка ОС и систем программирования.. Организация регистровой памяти ЦП.

Проблема программ из большого кол-ва модулей и подпрограмм – накладные расходы при входе и выходе из подпрограмм. В общем случае в модуле используются все регистры \Rightarrow регистровые окна.

- В процессоре физически есть К регистров общего назначения,

- через систему команд в программе доступна только часть этих регистров.
- Аппарат регистровых окон позволяет реализовывать команды, которые при входе в подпрограмму перемещают окно доступных регистров из отображения в одни реальные регистры в другие регистры => фактически доступны виртуальные регистры => виртуализация регистров
- CWP – указатель текущего окна
- SWP – указатель сохраненного окна

(Авторские права принадлежат Аните)



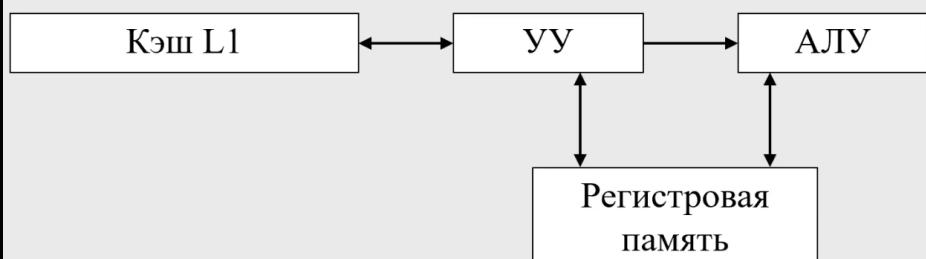
Центральный процессор

Устройство управления (control unit) — координация выполнения команд.

Арифметико-логическое устройство (arithmetic/logic unit) — выполнение команд, арифметической или логической обработки операндов.

Регистровая память (register memory) — совокупность устройств памяти процессора - регистров.

Кэш-память (cache memory) — буферизация работы процессора с оперативной памятью.



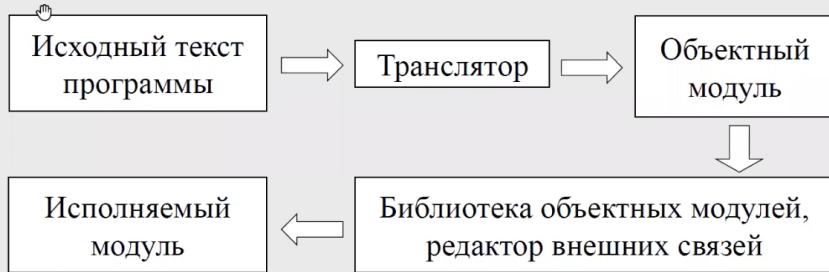
14. Аппаратная поддержка ОС. Виртуальная оперативная память.

Виртуальная память. Базирование

Аппарат виртуальной памяти — аппаратные средства компьютера, обеспечивающие преобразование (установление соответствия) программных адресов, используемых в программе в адреса физической памяти, в которой размещена программа во время выполнения.

Базирование адресов — реализация одной из моделей аппарата виртуальной памяти.

Виртуальная память. Базирование

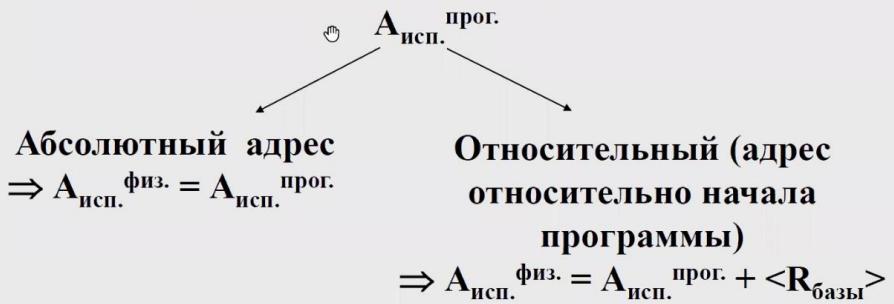


В исполняемом модуле используется программная (логическая или виртуальная) адресация

Проблема – установление соответствия между программной адресацией и физической памятью

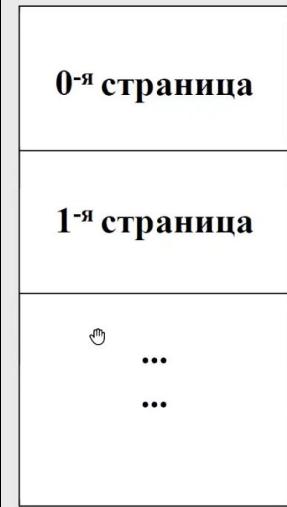
Виртуальная память. Базирование

Базирование адресов — решение проблемы перемещаемости программы по ОЗУ.



15. Аппаратная поддержка ОС. Пример организации страничной виртуальной памяти.

Виртуальная память. Страницчная организация памяти



Страницы - блоки фиксированного размера. Размер страницы — 2^k

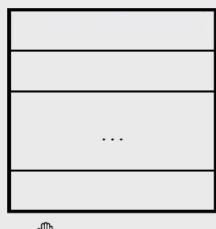
Структура адреса



Количество страниц ограничено размером поля «номер страницы»

Виртуальная память. Страницчная организация памяти

Таблица страниц процессора



$A_{\text{исп.}}^{\text{вирт.}} \quad \begin{matrix} k & k-1 & \dots & 0 \end{matrix}$

Номер виртуальной страницы | Номер в странице

$A_{\text{исп.}}^{\text{физ.}} \quad \begin{matrix} k & k-1 & \dots & 0 \end{matrix}$

Номер физической страницы | Номер в странице

Преобразование виртуального адреса в физический — замена номера виртуальной страницы на соответствующий номер физической страницы

Виртуальная память. Страницчная организация памяти

Модельный пример организации страницочной виртуальной памяти

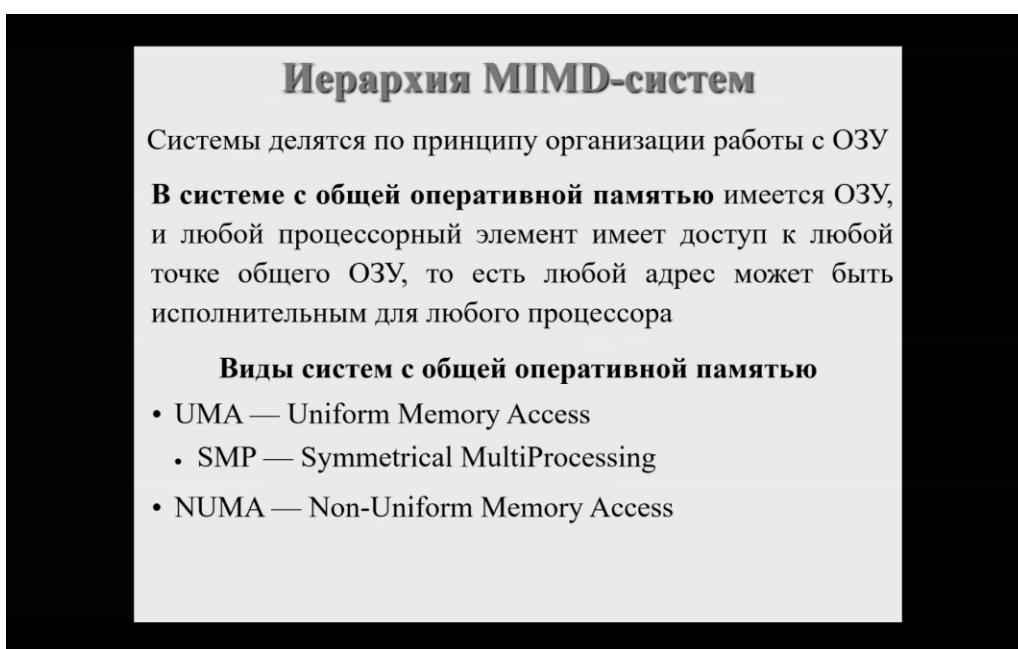
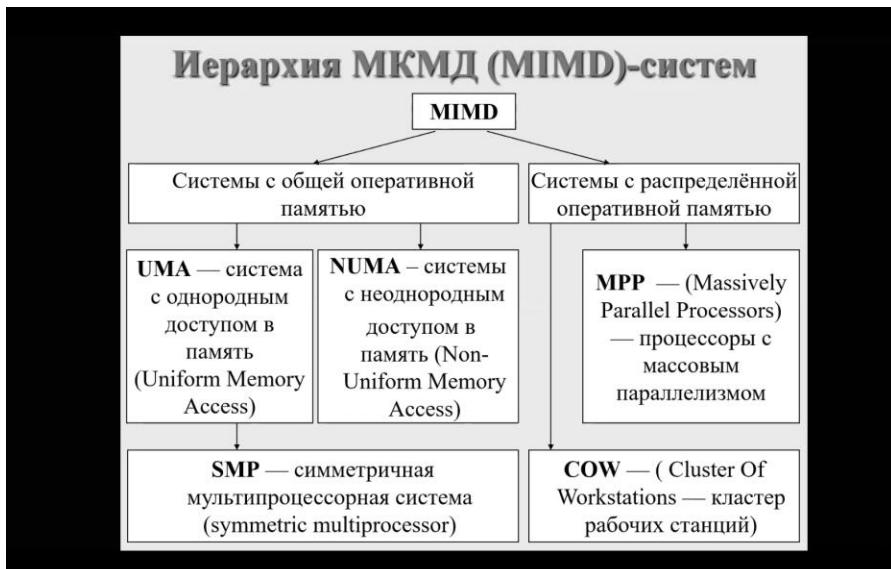


16. Многомашинные, многопроцессорные ассоциации. Классификация. Примеры.

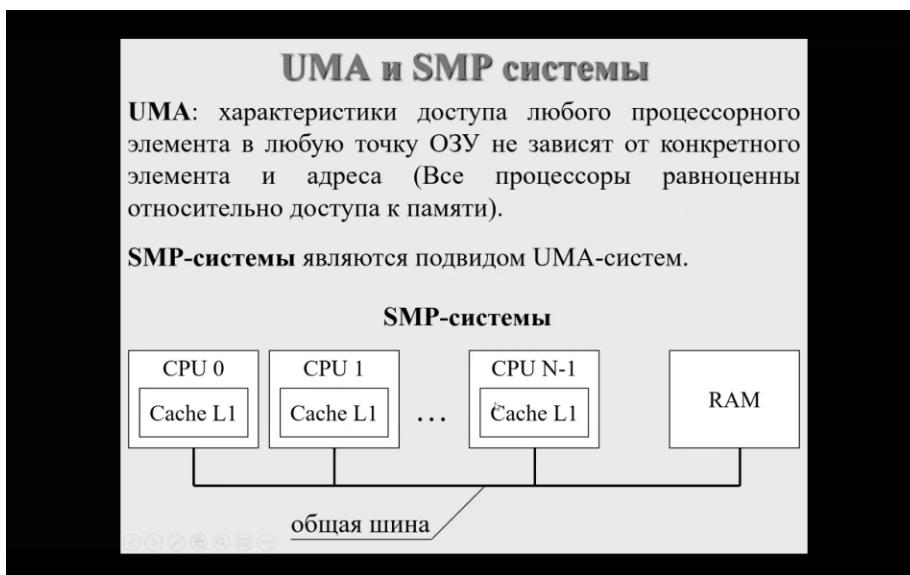
Классификация архитектур многопроцессорных ассоциаций

Автор классификации: Майкл Флинн (M. Flynn)

- Поток инструкций (команд)
- Поток данных
- **ОКОД (SISD — Single Instruction, Single Data stream)** — компьютер с единственным ЦП
- **ОКМД (SIMD — Single Instruction, Multiple Data stream)** — матричная обработка данных
- **МКОД (MISD — Multiple Instruction, Single Data stream) — ?**
- **МКМД (MIMD — Multiple Instruction, Multiple Data stream)** — множество процессоров одновременно выполняют различные последовательности команд над своими данными



UMA



SMP-системы

Синхронизация кэша.

Поведение кэш-памяти с отслеживанием при чтении/записи

Операции	Локальный кэш	Кэш других процессоров
Промах при чтении	Запись из памяти в кэш	Ничего
Попадание при чтении	Использование кэша	Ничего (операция «не видна»)
Промах при записи	Запись в память	Соответствующая запись в кэше удаляется
Попадание при записи	Запись в память и кэш	Соответствующая запись в кэше удаляется

SMP-системы

Преимущества SMP

- Простота реализации

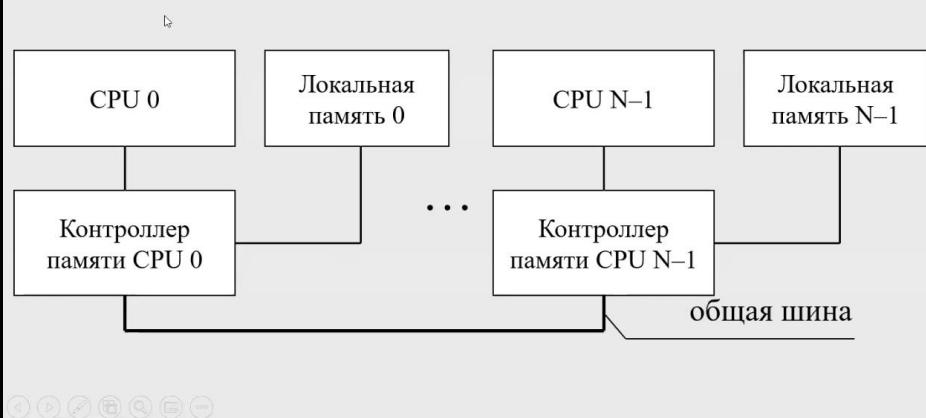
Недостатки SMP

- Задержки при доступе к памяти
- Система с явной централизацией — общая шина является «узким местом»
- Проблема синхронизации кэша (решением является кэш-память с отслеживанием)
- Ограничение на количество процессорных элементов (как следствие централизации)

NUMA

NUMA-системы

NUMA: Процессорные элементы работают на общем адресном пространстве, но характеристики доступа процессора к ОЗУ зависят от того, куда он обращается.



NUMA-системы

Преимущества NUMA

- Степень параллелизма выше, чем в SMP

Недостатки NUMA

- Централизация (ограничение ресурсом шины)
- Использование когерентных кэшей загружает шину служебной информацией

Недостатки ccNUMA

- Загрузка общей шины служебной информацией

Иерархия MIMD-систем

Системы с распределённой оперативной памятью представляются как объединение компьютерных узлов, каждый из которых состоит из процессора и ОЗУ, непосредственный доступ к которой имеет только «свой» процессорный элемент. Класс наиболее перспективных систем.

Виды систем с распределённой оперативной памятью

- MPP — Massively Parallel Processors
- COW — Cluster Of Workstations

MPP

MPP-системы

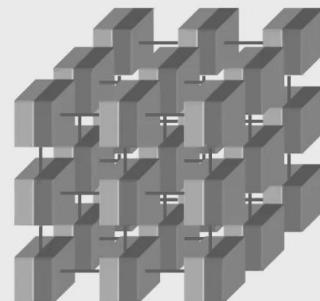
MPP — Специализированные дорогостоящие ВС. Эти компьютеры могут выстраиваться, процессорные элементы могут объединяться в различные топологии: макроконвейер, n-мерный гиперкуб и др.

Примеры топологий MPP-систем

Макроконвейер:



3-мерный гиперкуб:



Процессорный элемент с локальной памятью



Межэлементные коммуникации, определяющие топологию

MPP-системы

Преимущества MPP

- Высокая эффективность при решении определённого класса задач

Недостатки MPP

- Высокая стоимость
- Узкая специализация

COW

COW-кластеры

- Кластер как вычислительный узел (высокопроизводительная система)
- Кластеры, которые обеспечивают надёжность (сохранение работоспособности при возможном снижении производительности)

COW- кластеры

Преимущества COW

- «прозрачность» архитектуры
- относительная «универсальность» - возможность применения для решения широкого круга задач

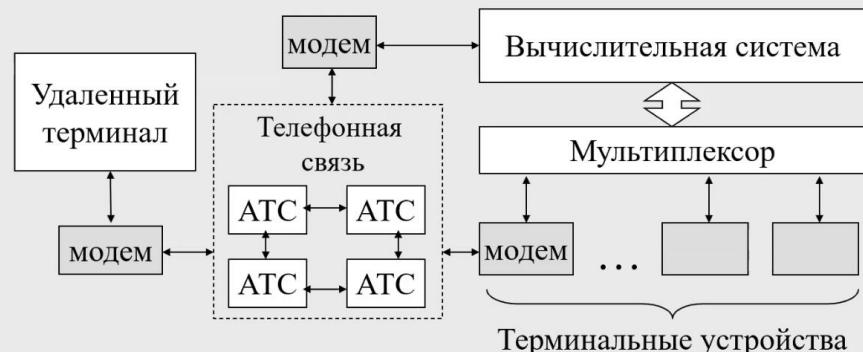
Проблемы COW

- Топология

17. Многомашинные, многопроцессорные ассоциации. Терминальные комплексы. Компьютерные сети.

Терминальные комплексы

Терминальный комплекс — многомашинная ассоциация, предназначенная для организации массового доступа удаленных и локальных пользователей к ресурсам некоторой вычислительной системы.



Линии связи / каналы

Организация канала

- **Коммутируемый канал**
Физически коммутируемая линия может иметь различные топологии
- **Выделенный канал**
Обеспечивает доступ к системе на постоянной основе.
Преимущества: отсутствие отказа, детерминированное качество соединения

Организация канала

- **Канал точка-точка**
Один абонент общается с одним абонентом (подключение к удалённому терминалу без мультиплексирования)
- **Многоточечный канал**
Подключение терминала осуществляется через локальный мультиплексор

Направление движения информации

- Симплексные каналы
- Дуплексные каналы
- Полудуплексные каналы

Компьютерные сети

Компьютерная сеть — объединение компьютеров (вычислительных систем), взаимодействующих через коммуникационную среду.

Коммуникационная среда — каналы и средства передачи данных.



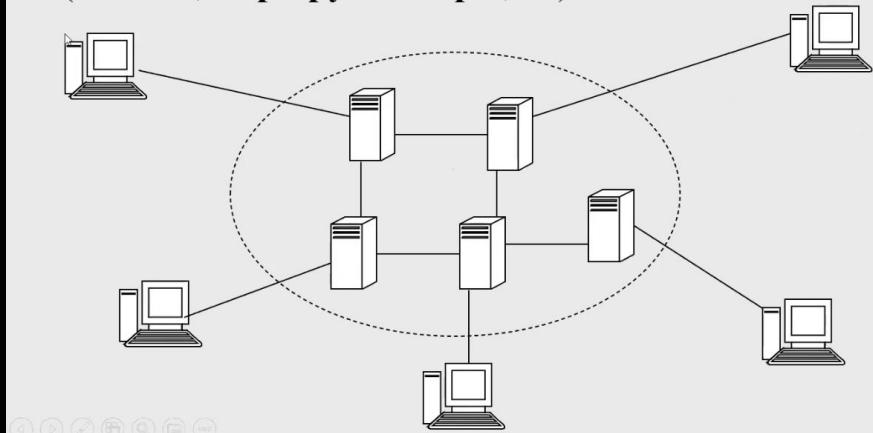
Свойства компьютерных сетей

1. Большое число связанных узлов, обеспечивающих решение определённых задач
2. Возможность распределения обработки информации
3. Расширяемость сети (сеть должна обеспечивать развитие сети по протяжённости, по расширению пропускной способности каналов, по составу и производительности узлов, входящих в состав сети)
4. Применение симметричных интерфейсов обмена информации внутри сети (возможность произвольного распределения функций внутри сети)

Компьютерные сети

Составляющие компьютерной сети

- Абонентские или основные компьютеры — **хосты**
- Коммуникационные или вспомогательные компьютеры (**шлюзы, маршрутизаторы, ...**)



Компьютерные сети

Модели построения компьютерной сети:

- Сеть коммутации каналов
- Сеть коммутации сообщений
- Сеть коммутации пакетов

Сообщение — логически целостная порция данных, имеющая произвольный размер).

Взаимодействие абонентов осуществляется сеансами связи. **Сеанс связи** состоит из обмена сообщениями между абонентами. Начало/завершение сеанса связи.

≡ Оси Лекция #8-9



Сеть коммутации каналов

Сеть коммутации каналов обеспечивает выделение коммуникаций абонентам на весь сеанс связи.

Преимущества

- После установления соединения сеть находится в состоянии готовности
- Требования к коммуникационному оборудованию минимальны
- Минимизируются накладные расходы по передаче данных
- Детерминированная пропускная способность

Недостатки

- Требование избыточности сети
- Период ожидания соединения (канала) недетерминирован
- Неэффективное использование выделенного канала
- В случае сбоя или отказа повторная передача информации

Сеть коммутации сообщений

Взаимодействие представляется в виде последовательности обменов сообщениями.

Преимущества

- Отсутствие занятости канала на недетерминированный промежуток времени

Недостатки

- Сообщения могут быть произвольного размера - необходимость наличия средств буферизации неопределённых характеристик
- Необходимость в специализированном коммуникационном оборудовании и ПО
- Повтор передачи всего сообщения в случае сбоя

Сеть коммутации пакетов

Каждое сообщение разбивается на блоки фиксированного размера - **пакеты**. Структура пакета (заголовок, данные)

Преимущества

- Так как известна топология сети и характеристики её элементов, то возможно определение требований в коммутационных узлах
⇒ возможна оценка размера буфера и времени доставки пакетов

Недостатки

- Увеличение трафика из-за наличия заголовочной информации
- Проблема сборки пакетов

18. Операционные системы. Основные компоненты и логические функции. Базовые понятия: ядро, процесс, ресурс, системные вызовы. Структурная организация ОС.

Базовые понятия

Операционная система — комплекс программ, обеспечивающий контроль за существованием, распределением и использованием ресурсов ВС.

Процесс — совокупность машинных команд и данных, исполняющаяся в рамках ВС и обладающая правами на владение некоторым набором ресурсов.

Требования к ОС

- **Надежность**

Количество ошибок должно быть минимизировано

- **Защита**

Предусмотрение защиты информации и ресурсов от несанкционированного доступа

- **Эффективность**

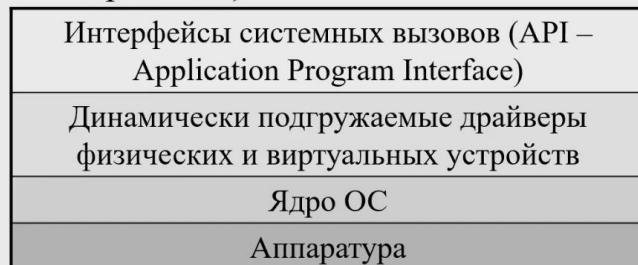
Удовлетворение критериям эффективности

- **Предсказуемость**

Известны заранее проблемы и последствия различных действий, устойчивость к форс-мажору

Структура ОС

Ядро (Kernel) — резидентная часть ОС, работающая в режиме супервизора (обычно работает в режиме физической адресации).



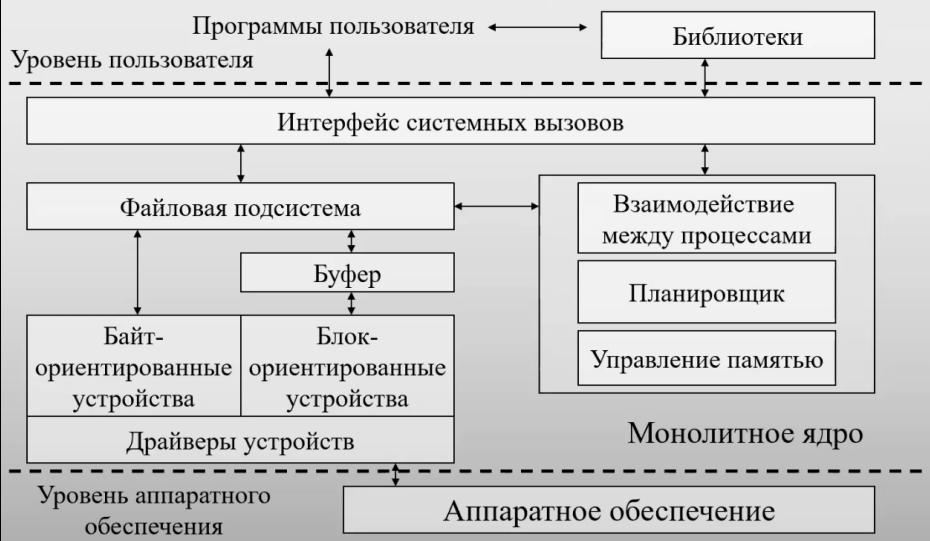
Динамически подгружаемые драйверы устройств:

- резидентные / нерезидентные
- работают в пользовательском / привилегированном режиме

Системный вызов — обращение к ОС за предоставление той или иной функции (возможности, услуги, сервиса).

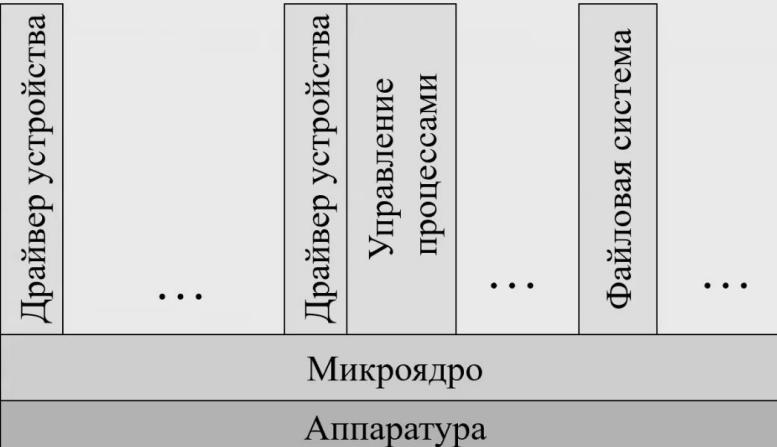
Структура ОС

Пример структурной организации классической системы UNIX



Структура ОС

Микроядерная архитектура



Логические функции ОС

- Управление процессами
- Управление ОП
- Планирование
- Управление устройствами и ФС
- Сетевое взаимодействие
- Безопасность

19. Операционные системы. Пакетная ОС, ОС разделения времени, ОС реального времени.

Пакетная ОС

Переключение выполнения процессов происходит:

- выполнение процесса завершено
- возникло прерывание
- зацикливания процесса

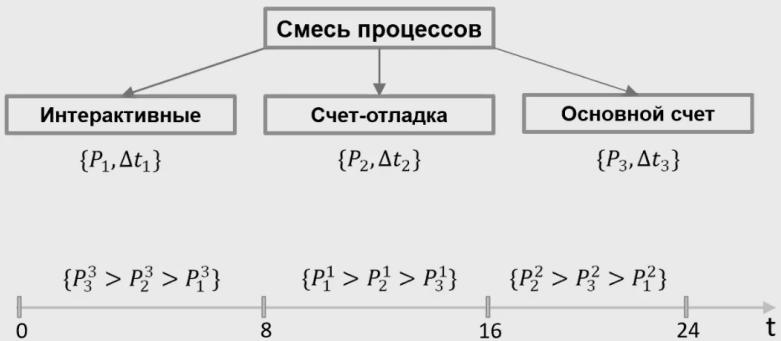
Системы разделения времени

Квант времени ЦП — некоторый фиксированный ОС промежуток времени работы ЦП.

Переключение выполнения процессов происходит:

- исчерпался выделенный квант времени
- выполнение процесса завершено
- возникло прерывание
- зацикливания процесса

Модель организации планирования процессов

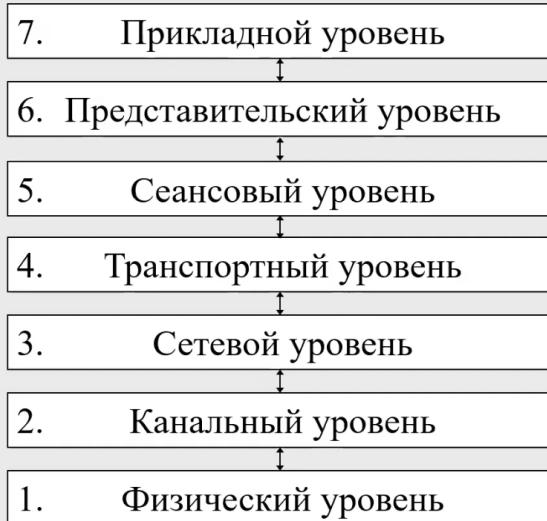


ОС реального времени

Системы реального времени являются специализированными системами, в которых все функции планирования ориентированы на обработку фиксированного набора событий за время, не превосходящее некоторого предельного значения.

20. Организация сетевого взаимодействия. Эталонная модель ISO/OSI. Протокол, интерфейс. Стек протоколов. Логическое взаимодействие сетевых устройств.

Модель ISO/OSI организации взаимодействия в сети



Основные понятия

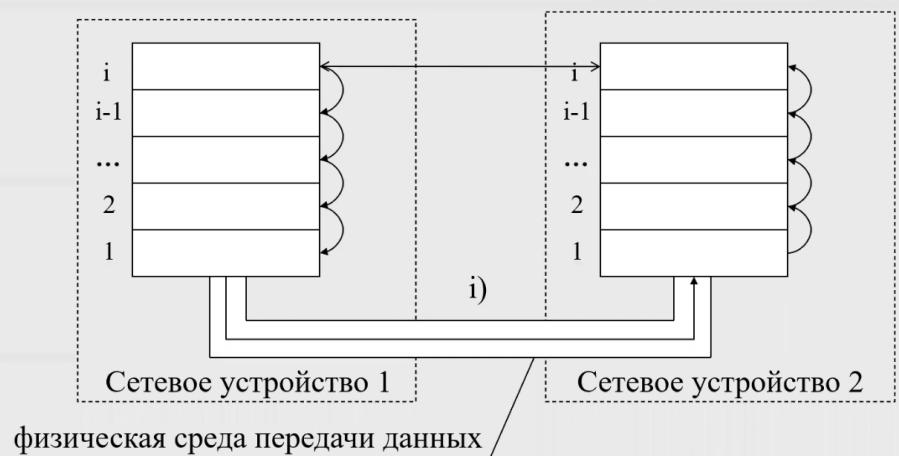
Протокол (правила взаимодействия одноименных уровней) — формальное описание сообщений и правил, по которым сетевые устройства (вычислительные системы) осуществляют обмен информацией. Правила взаимодействия одноимённых (одноранговых) уровней сети.

Интерфейс — правила взаимодействия вышестоящего уровня с нижестоящим.

Служба (сервис) — набор операций, предоставляемых нижестоящим уровнем вышестоящему.

Стек протоколов — перечень разноуровневых протоколов, реализованных в системе.

Логическое взаимодействие сетевых устройств по i-ому протоколу



21. Организация сетевого взаимодействия. Семейство протоколов TCP/IP, соответствие модели ISO/OSI. Взаимодействие между уровнями протоколов семейства TCP/IP. IP адресация.

Семейство протоколов TCP/IP



Соответствие модели ISO/OSI модели семейства протоколов TCP/IP

Уровень модели TCP/IP	Уровень модели ISO/OSI
1. Уровень доступа к сети Специфицирует доступ к физической сети.	Канальный уровень Физический уровень
2. Межсетевой уровень В отличие от сетевого уровня модели OSI, не устанавливает соединений с другими машинами.	Сетевой уровень

Соответствие модели ISO/OSI модели семейства протоколов TCP/IP

Уровень модели TCP/IP	Уровень модели ISO/OSI
3. Транспортный уровень Обеспечивает доставку данных от компьютера к компьютеру, обеспечивает средства для поддержки логических соединений между прикладными программами. В отличие от транспортного уровня модели OSI, в функции транспортного уровня TCP/IP не всегда входит контроль за ошибками и их коррекция. TCP/IP предоставляет два разных сервиса передачи данных на этом уровне. Протокол TCP, UDP.	Сеансовый уровень Транспортный уровень

Соответствие модели ISO/OSI модели семейства протоколов TCP/IP

Уровень модели TCP/IP	Уровень модели ISO/OSI
4. Уровень прикладных программ Состоит из прикладных программ и процессов, использующих сеть и доступных пользователю. В отличие от модели OSI, прикладные программы сами стандартизуют представление данных.	Уровень прикладных программ Уровень представления данных

Взаимодействие между уровнями протоколов TCP/IP



Уровень доступа к сети

На уровне доступа к сети протоколы обеспечивают систему средствами для передачи данных другим устройствам в сети

Пример

Протокол Ethernet — разработка исследовательской компании Xerox (1976 год). Единая шина — широковещательная сеть. Для сетевых устройств обеспечивается множественный доступ, с контролем и обнаружением конфликтов (Carrier Sense Multiple Access with Collision Detection — CSMA/CD)

Межсетевой уровень. Протокол IP

- Функции протокола IP
 - формирование дейтаграмм
 - поддержание системы адресации
 - обмен данными между транспортным уровнем и уровнем доступа к сети
 - организация маршрутизации дейтаграмм
 - разбиение и обратная сборка дейтаграмм
- IP — протокол БЕЗ логического установления соединения
- Протокол IP НЕ обеспечивает обнаружение и исправление ошибок

Система адресации протокола IP

32 бита



Класс А



Класс В



Класс С



Класс D



Класс Е



Дейтаграммы

Дейтаграмма — пакет протокола IP.

Шлюз — устройство, передающее пакеты между различными сетями.

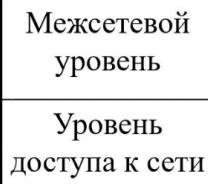
Маршрутизация — процесс выбора шлюза или маршрутизатора.

Маршрутизация дейтаграмм

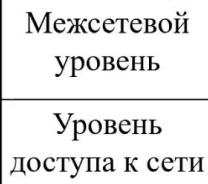
Хост А1



Шлюз G1



Шлюз G2



Хост А2



Сеть А

Сеть В

Сеть С

Транспортный уровень

- **Протокол контроля передачи (TCP, Transmission Control Protocol)** обеспечивает надежную доставку данных с обнаружением и исправлением ошибок и с установлением логического соединения.
- **Протокол пользовательских дейтаграмм (UDP, User Datagram Protocol)** отправляет пакеты с данными, не контролируя их доставку.

TCP обеспечивает последовательную передачу пакетов, контроль доставки пакетов, отработку сбоев.

Уровень прикладных программ

На основе TCP базируются прикладные протоколы, которые обеспечивают либо доступ и работу с заведомо корректной информацией, которая осуществляется в сети Интернет.

- Протоколы, опирающиеся на TCP
 - TELNET (Network Terminal Protocol)
 - FTP (File Transfer Protocol)
 - SMTP (Simple Mail Transfer Protocol)
- Протоколы, опирающиеся на UDP
 - DNS (Domain Name Service)
 - RIP (Routing Information Protocol)
 - NFS (Network File System)

22. Управление процессами. Определение процесса, типы. Жизненный цикл, состояния процесса. Свопинг. Модели жизненного цикла процесса. Контекст процесса.

Определение процесса. Основные понятия

Процесс — совокупность машинных команд и данных, которая обрабатывается в рамках вычислительной системы и обладает правами на владение некоторым набором *ресурсов*.

Ресурсы могут принадлежать только одному процессу, либо ресурсы могут разделяться между процессами — **разделяемые ресурсы**.

Выделение ресурсов процессу

- предварительная декларация — до начала выполнения
- динамическое пополнение списка принадлежащих процессу ресурсов

Количество допустимых процессов в системе — *ресурс ВС*.



Жизненный цикл процесса

Основной из задач ОС является поддержание жизненного цикла процесса.

Типовые этапы обработки процесса в системе

Жизненный цикл процесса в системе:

- Образование (порождение) процесса
- Обработка (выполнение) процесса
- Ожидание (по тем или иным причинам) постановки на выполнение
- Завершение процесса

Модельная ОС

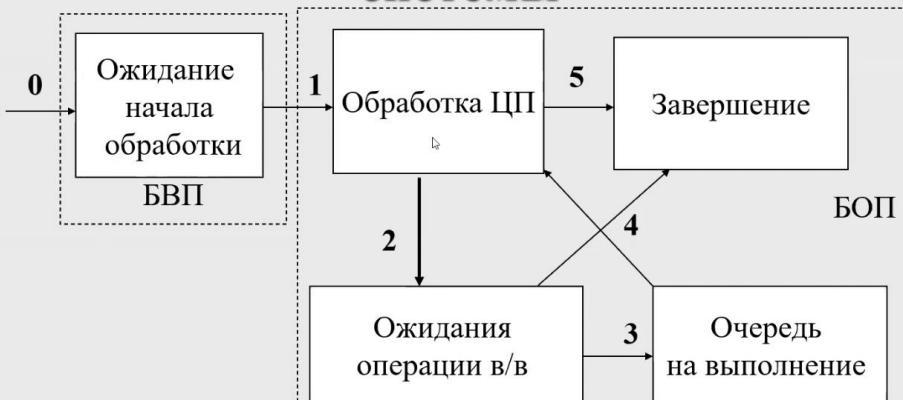
1. **Буфер ввода процессов (БВП)** — пространство, в котором размещаются и хранятся сформированные процессы с момента их образования, до момента начала выполнения.
2. **Буфер обрабатываемых процессов (БОП)** — буфер для размещения процессов, находящихся в системе в мультипрограммной обработке.

Модель пакетной однопроцессной системы

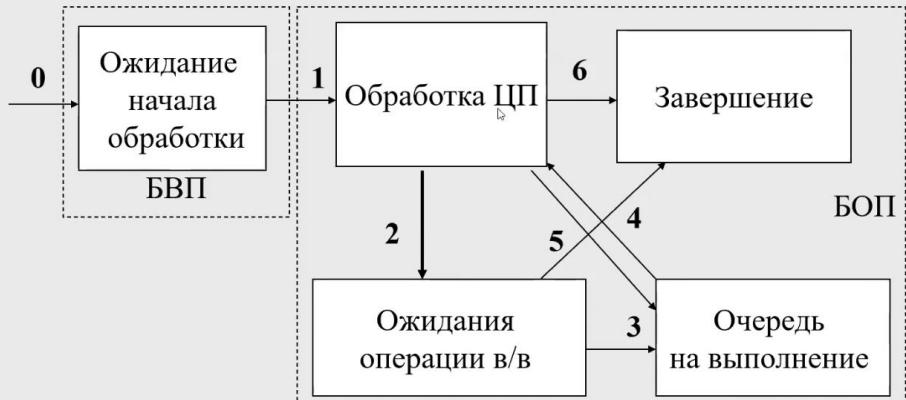


0. Поступление процесса в очередь на начало обработки ЦП (процесс попадает в БВП)
1. Начало обработки процесса на ЦП (из БВП в БОП)
2. Завершение выполнения процесса, освобождение системных ресурсов.

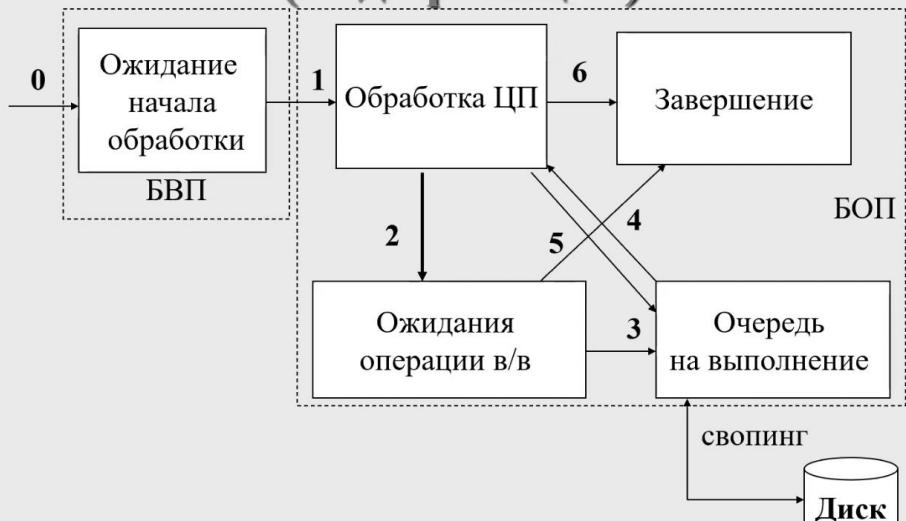
Модель пакетной мультипроцессной системы



Модель ОС с разделением времени



Модель ОС с разделением времени (модификация)



Типы процессов

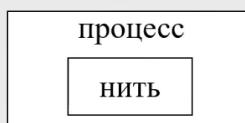
- «полновесные» процессы
- «легковесные» процессы

«Полновесные процессы» — процессы, выполняющиеся внутри защищенных участков оперативной памяти.

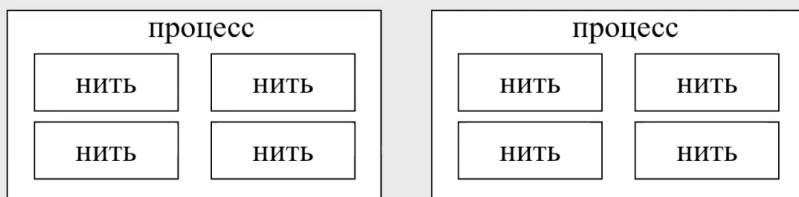
«Легковесные процессы» (нити) — работают в мультипрограммном режиме одновременно с активировавшим их полновесным процессом и используют его виртуальное адресное пространство.

Типы процессов

Однонитевая организация процесса
— «один процесс — одна нить»:



Многонитевая организация процесса:



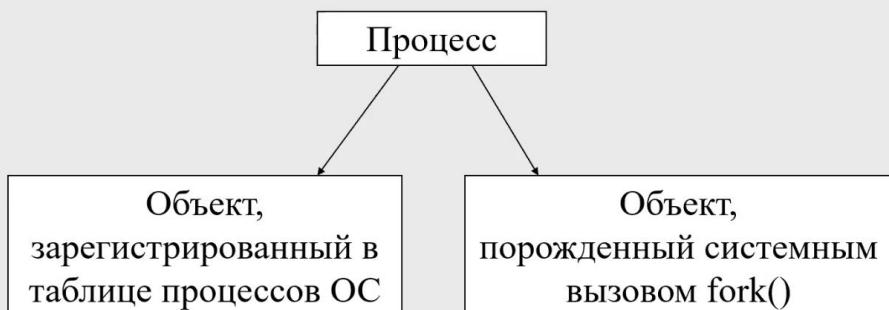
23. Реализация процессов в ОС UNIX. Определение процесса. Контекст, тело процесса. Состояния процесса. Аппарат системных вызовов в ОС UNIX.

Контекст процесса

Контекст процесса — совокупность данных, характеризующих актуальное состояние процесса.

- Пользовательская составляющая — текущее состояние программы (совокупность машинных команд и данных, размещенных в ОЗУ)
- Системно-аппаратная составляющая
 - информация идентификационного характера (PID процесса, PID «родителя»...)
 - информация о содержимом регистров, настройках аппаратных интерфейсов, режимах работы процессора и т.п.
 - информация, необходимая для управления процессом (состояние процесса, приоритет).

Определение процесса Unix



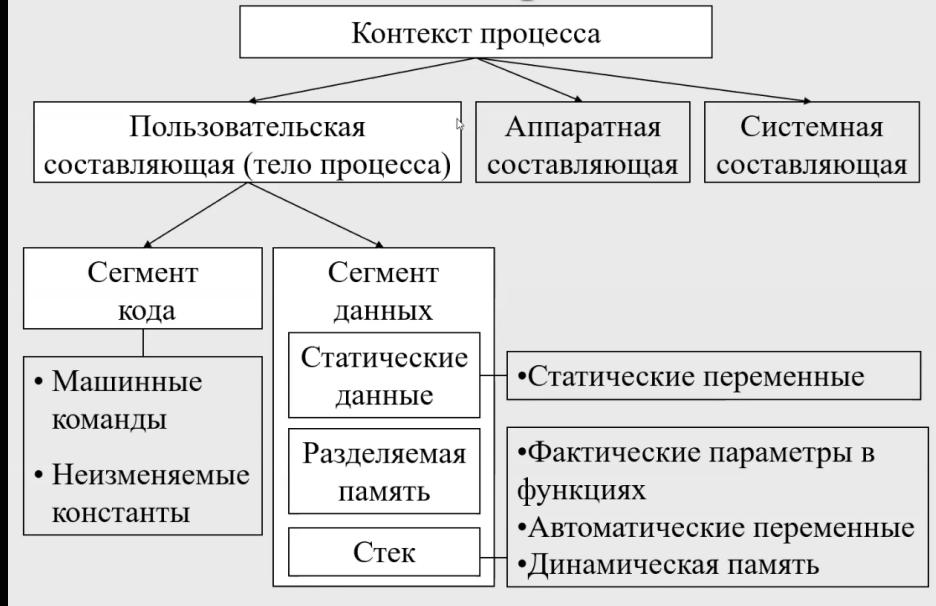
Определение процесса в UNIX

Процесс в UNIX — объект, зарегистрированный в таблице процессов UNIX.

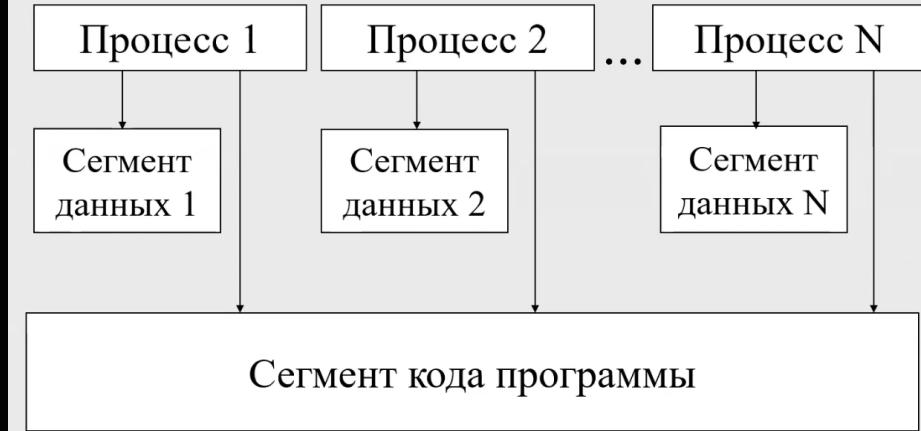
Идентификатор процесса (PID)



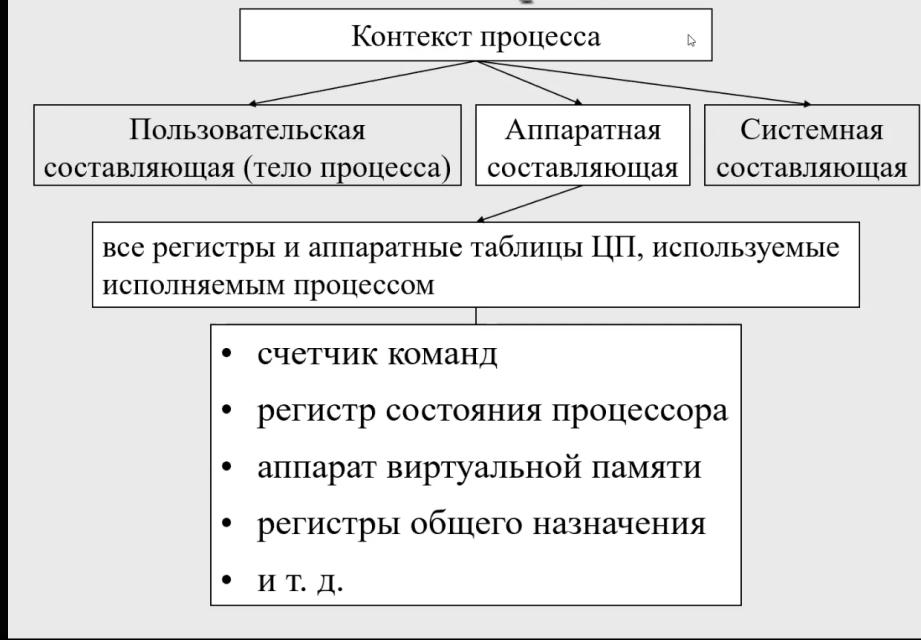
Контекст процесса

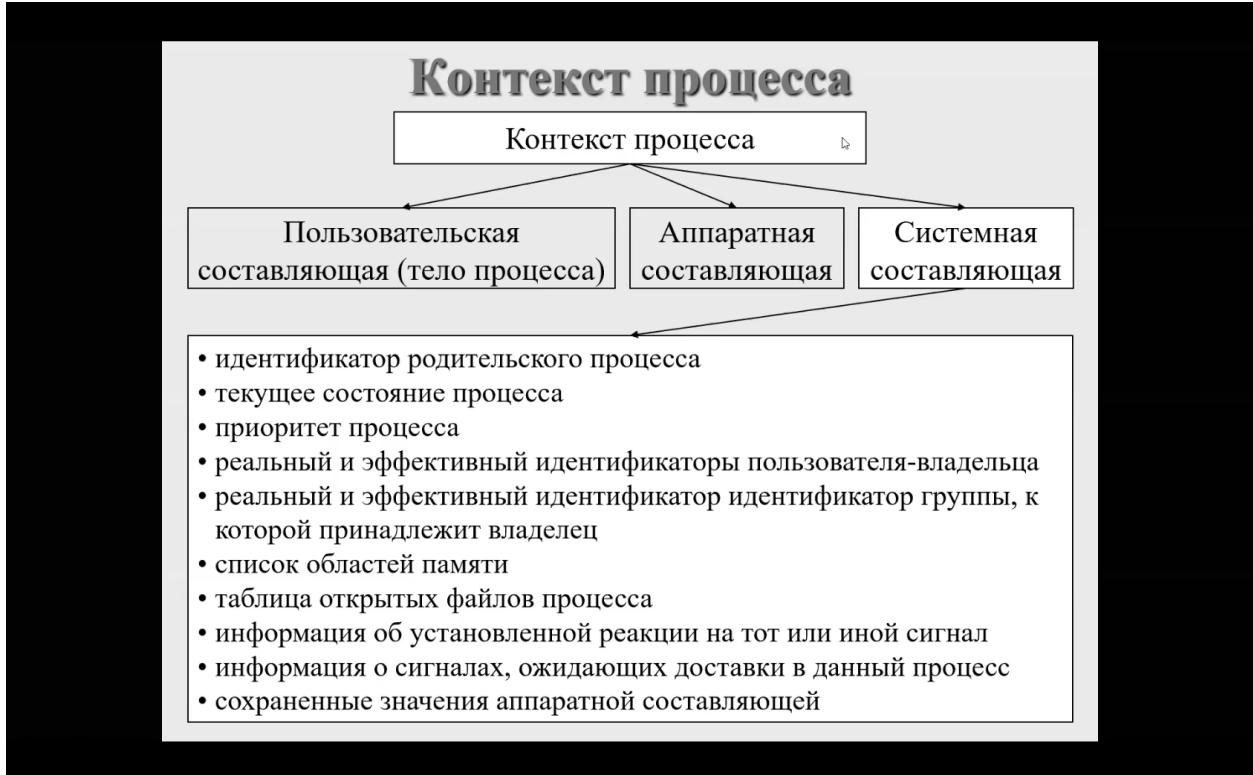


Разделение сегмента кода

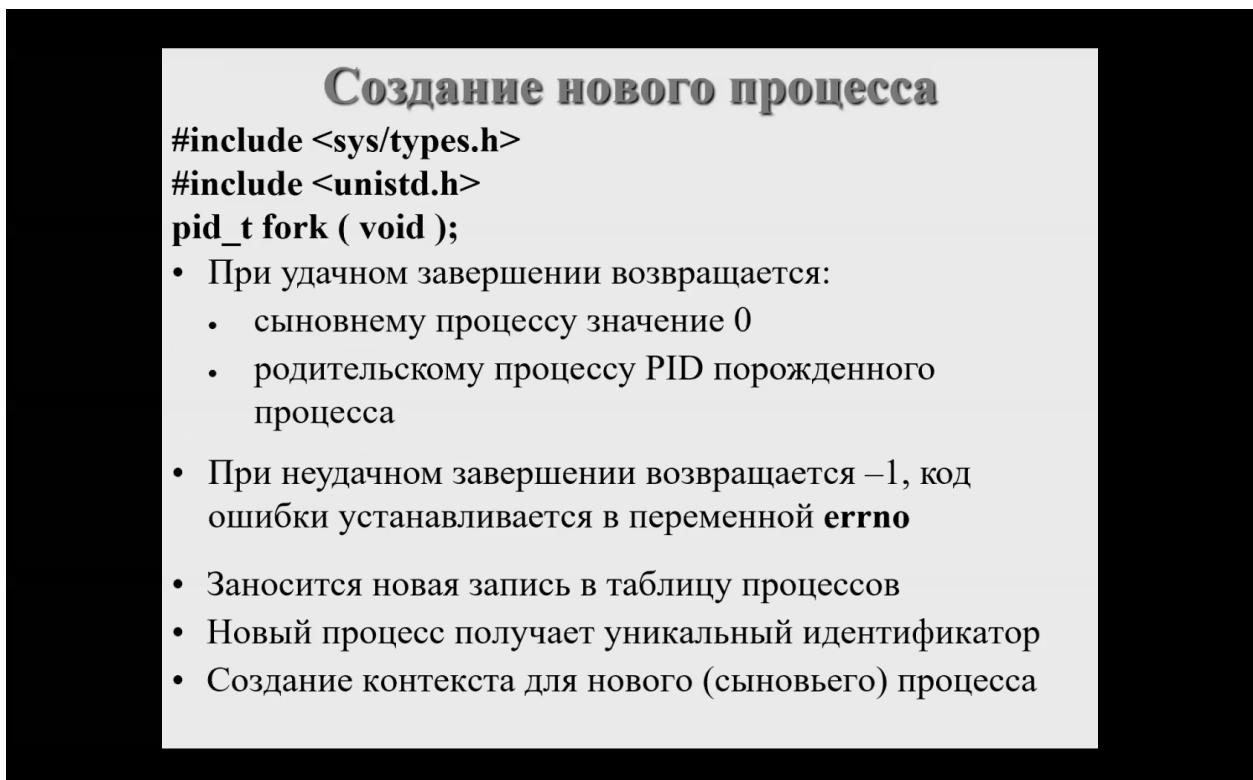


Контекст процесса





24. Реализация процессов в ОС UNIX. Базовые средства управления процессами в ОС UNIX. Загрузка ОС UNIX, формирование нулевого и первого процессов.



Создание нового процесса

Составляющие контекста, наследуемые при вызове
`fork()`

- Окружение
- Файлы, открытые в процессе-отце
- Способы обработки сигналов
- Разрешение переустановки эффективного идентификатора пользователя
- Разделяемые ресурсы процесса-отца
- Текущий рабочий каталог и домашний каталоги
- ...

Семейство системных вызовов `exec()`

```
#include <unistd.h>
int execl (const char *path, char *arg0, ..., char *argn, 0);
```

- **path** — имя файла, содержащего исполняемый код программы
- **arg0** — имя файла, содержащего вызываемую на выполнение программу
- **arg1, ..., argn** — аргументы программы, передаваемые ей при вызове

Возращается:
в случае ошибки

-1

Семейство системных вызовов exec()

Сохраняются:

- Идентификатор процесса
- Идентификатор родительского процесса
- Таблица дескрипторов файлов
- Приоритет и большинство атрибутов

Изменяются:

- Режимы обработки сигналов
- Эффективные идентификаторы владельца и группы
- Файловые дескрипторы (закрытие некоторых файлов)

Завершение процесса

- Системный вызов **_exit()**
- Выполнение оператора **return**, входящего в состав функции **main()** или выход на закрывающую операторную скобку объемлющего блока функции
- Получение сигнала

Завершение процесса

- Освобождается сегмента кода и сегмента данных процесса
- Закрываются все открытые дескрипторы файлов
- Если у процесса имеются потомки, их предком назначается процесс с идентификатором 1
- Освобождается большая часть контекста процесса (кроме статуса завершения и статистики выполнения)
- Процессу-предку посыпается сигнал SIGCHLD

Получение информации о завершении своего потомка

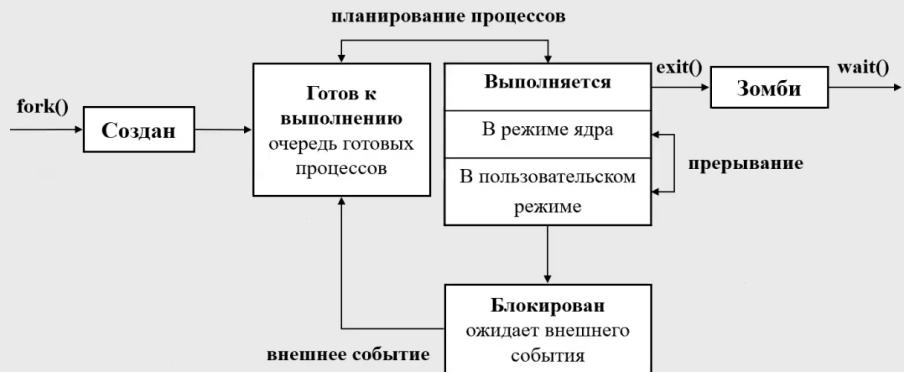
```
#include <sys/types.h>
#include <sys/wait.h>
pid_t wait ( int *status );
```

status по завершению содержит:

- в старшем байте — код завершения процесса-потомка
(пользовательский код завершения процесса)
- в младшем байте — индикатор причины завершения процесса-потомка, устанавливаемый ядром UNIX
(системный код завершения процесса)

Возвращается: PID завершенного процесса или –1 в случае ошибки или прерывания

Жизненный цикл процессов



Начальная загрузка

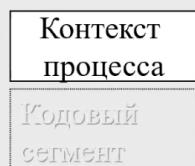
Начальная загрузка — загрузка ядра системы в оперативную память, запуск ядра.

- Чтение нулевого блока системного устройства аппаратным загрузчиком
- Поиск и считывание в память файла /unix
- Запуск на исполнение файла /unix

Инициализация Unix системы

- Начальная инициализация компонентов компьютера (установка часов, инициализация контроллера памяти и пр.)
- Инициализация системных структур данных
- Инициализация процесса с номером “0”:
 - не имеет кодового сегмента
 - существует в течении всего времени работы системы

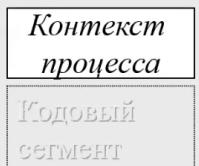
0



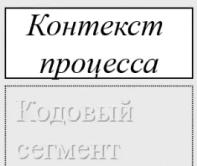
Инициализация системы

- Создание ядром первого процесса
 - Копируется процесс “0” (запись таблицы процессов)
 - Создание области кода процесса “1”
 - Копирование в область кода процесса “1” программы, реализующей системный вызов `exec()`, который необходим для выполнения программы /etc/init

0



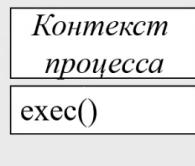
1



1



1



Инициализация системы

- Замена команды процесса “1” кодом из файла /etc/init (запуск exec())
- Подключение интерпретатора команд к системной консоли
- Создание многопользовательской среды

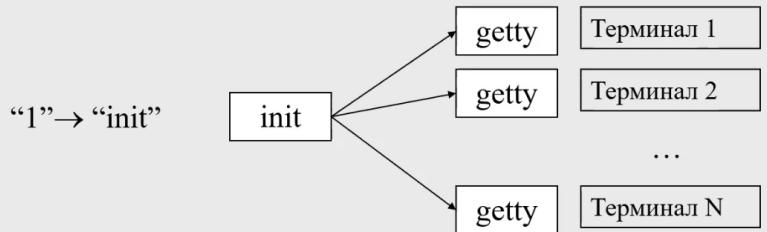
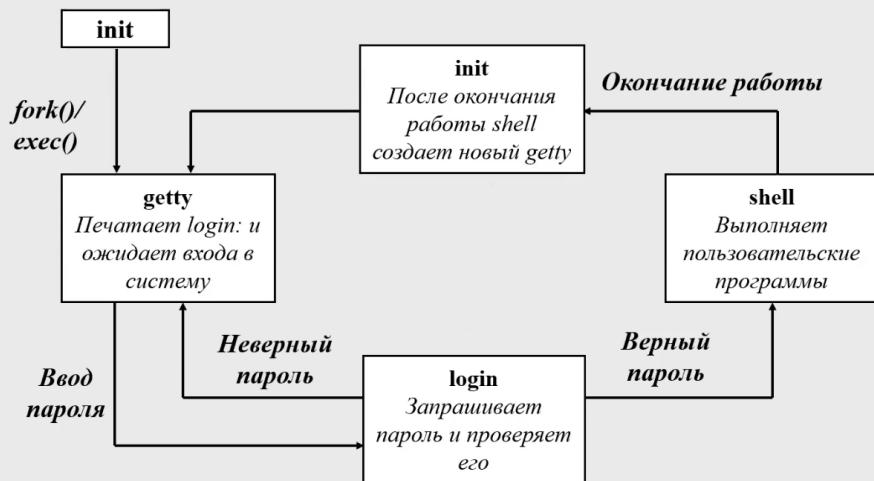


Схема дальнейшей работы системы



25. Взаимодействие процессов. Разделяемые ресурсы. Критические секции. Взаимное исключение. Тупики.

Параллельные процессы

Параллельные процессы — процессы, выполнение (обработка) которых хотя бы частично перекрывается по времени.

- **Независимые процессы** используют независимое множество ресурсов
- **Взаимодействующие процессы** используют ресурсы совместно, и выполнение одного процесса может оказать влияние на результат другого

Разделение ресурсов

Разделение ресурса — совместное использование несколькими процессами ресурса ВС.

Критические ресурсы — разделяемые ресурсы, которые должны быть доступны в текущий момент времени только одному процессу.

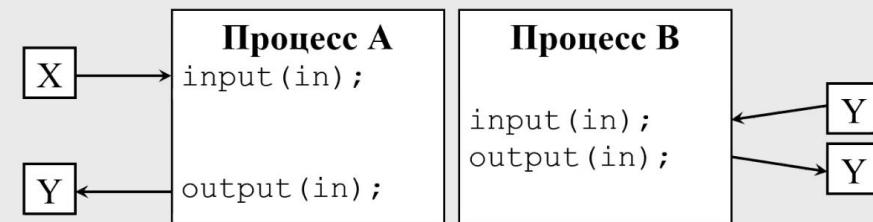
Критическая секция или критический интервал — часть программы (фактически набор операций), в которой осуществляется работа с критическим ресурсом.

Требование мультипрограммирования

Результат выполнения процессов не должен зависеть от порядка переключения выполнения между процессами.

Гонки (race conditions) между процессами.

```
void echo ()  
{  
    char in;  
    input ( in ) ;  
    output ( in ) ;  
}
```



Взаимное исключение

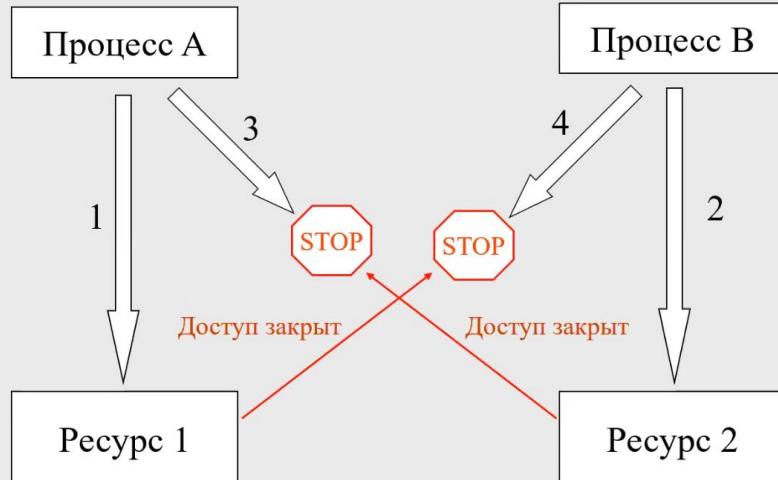
Взаимное исключение — способ работы с разделяемым ресурсом, при котором в тот момент, когда один из процессов работает с разделяемым ресурсом, все остальные процессы не могут иметь к нему доступ.

- Тупики (deadlocks)
- Блокирование (дискриминация)

Тупик — ситуация, при которой из-за некорректной организации доступа и разделения ресурсов происходит взаимоблокировка.

Блокирование — доступ одного из процессов к разделяемому ресурсу не обеспечивается из-за активности других, более приоритетных процессов.

Тупики (Deadlocks)



26. Взаимодействие процессов. Некоторые способы реализации взаимного исключения: семафоры Дейкстры, мониторы, обмен сообщениями.

Способы реализации взаимного исключения

Способы, которые позволяют работать с разделяемыми ресурсами. Существуют как аппаратные, так и алгоритмические модели.

- Запрещение прерываний и специальные инструкции
- Алгоритм Петерсона
- Активное ожидание
- **Семафоры Дейкстры**
- **Мониторы Хоара**
- **Обмен сообщениями**

Семафоры Дейкстры

Семафоры Дейкстры — формальная модель синхронизации, предложенная голландским учёным Эдсгером В. Дейкстрой

Операции, определённые над семафорами

- Down (S) или P (S) – Proberen (проверить)
- Up (S) или V (S) – Verhogen (увеличить)

Использование двоичного семафора для организации взаимного исключения

Двоичный семафор — семафор, максимальное значение которого равно 1.

процесс 1

```
int semaphore;
...
down ( semaphore ) ;
/*критическая секция
процесса 1*/
...
up ( semaphore ) ;
...
```

процесс 2

```
int semaphore;
...
down ( semaphore ) ;
/*критическая секция
процесса 2*/
...
up ( semaphore ) ;
...
```

Мониторы Хоара

Монитор Хоара — совокупность процедур и структур данных, объединенных в программный модуль специального типа.

- Структуры данных монитора доступны только для процедур, входящих в этот монитор
- Процесс «входит» в монитор по вызову одной из его процедур
- В любой момент времени внутри монитора может находиться не более одного процесса

Обмен сообщениями

Синхронизация и передача данных:

- для однопроцессорных систем и систем с общей памятью
- для распределенных систем (когда каждый процессор имеет доступ только к своей памяти)

Функции:

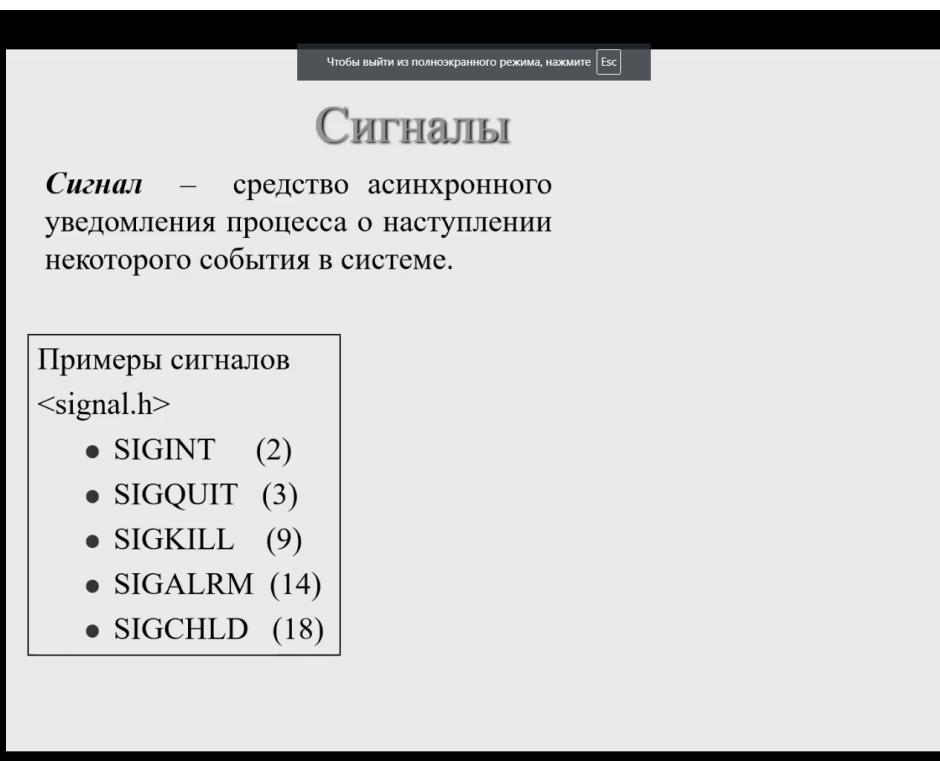
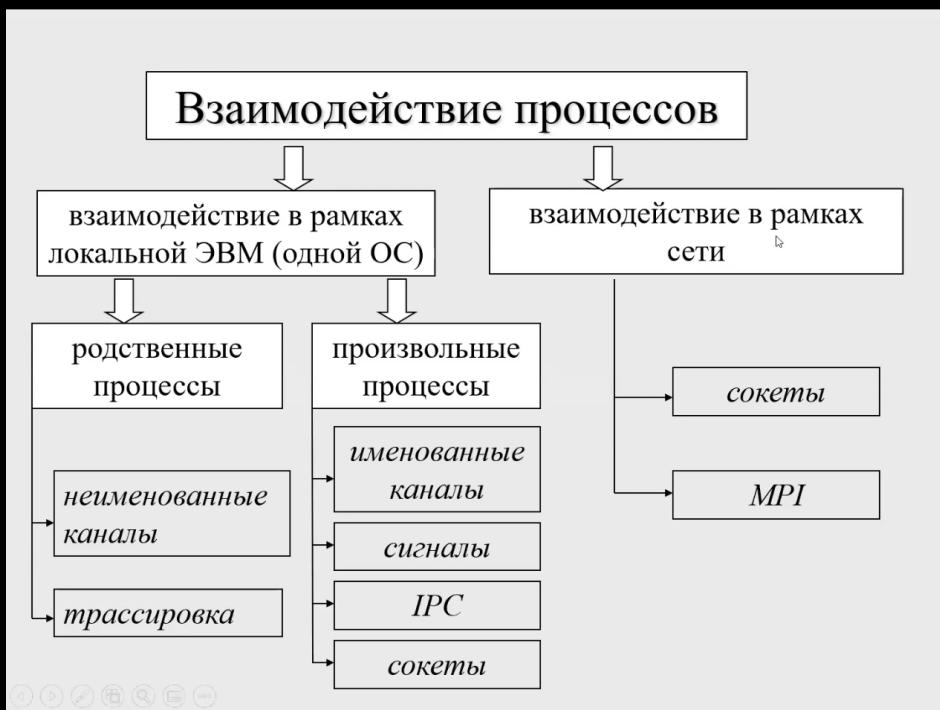
- send (destination, message)
- receive (source, message)

27. Взаимодействие процессов. Классические задачи синхронизации процессов. “Обедающие философы”.

28. Взаимодействие процессов. Классические задачи синхронизации процессов. “Читатели и писатели”.

29. Взаимодействие процессов. Классические задачи синхронизации процессов. «Спящий парикмахер».

30. Базовые средства взаимодействия процессов в ОС UNIX. Сигналы. Примеры программирования.



Работа с сигналами

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

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

pid – идентификатор процесса, которому посыпается сигнал

sig – номер посыпаемого сигнала

При удачном выполнении возвращает 0, в противном случае возвращает -1

Работа с сигналами

```
#include <signal.h>

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

sig – номер сигнала, для которого устанавливается реакция

disp – либо определенная пользователем функция – обработчик сигнала, либо одна из констант:

SIG_DFL – обработка по умолчанию
SIG_IGN - игнорирование

При успешном завершении функция возвращает указатель на предыдущий обработчик данного сигнала.

31. Базовые средства взаимодействия процессов в ОС UNIX. Неименованные каналы. Примеры программирования.

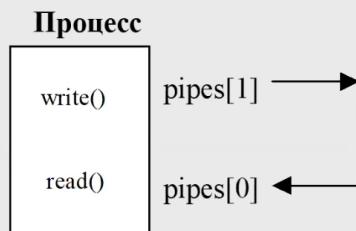
Неименованные каналы. Системный вызов pipe()

```
#include <unistd.h>
```

```
int pipe (int *pipes);
```

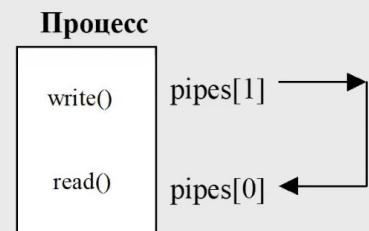
pipes[1] – запись в канал

pipes[0] – чтение из канала



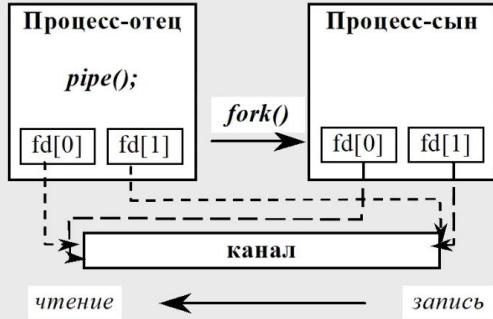
Пример. Использование канала.

```
int main(int argc, char **argv)
{
    char *s="channel";
    char buf[80];
    int pipes[2];
    pipe(pipes);
    write(pipes[1],s,strlen(s)+1);
    read(pipes[0],buf,strlen(s)+1);
    close(pipes[0]);
    close(pipes[1]);
    printf("%s\n",buf);
}
```



Пример. Типовая схема взаимодействия процессов с использованием канала.

```
int main(int argc, char **argv)
{
    int fd[2];
    pipe(fd);
    if(fork()) {
        close(fd[0]);
        write(fd[1], ...);
        ...
        close(fd[1]);
        ...
    }
    else {close(fd[1]);
          while(read(fd[0],...)) {...}
          ...
    }
}
```



```
#include <stdio.h>
int main(int argc, char **argv)
{
    int fd[2];

    pipe(fd);
    if(fork() == 0) { dup2(fd[1],1);
                      close(fd[1]);
                      close(fd[0]);
                      execl("/usr/bin/print","print",0);
    }
    dup2(fd[0],0);
    close(fd[0]);
    close(fd[1]);
    execl("/usr/bin/wc","wc",0);
}
```

Пример. Реализация конвейера.

32. Базовые средства взаимодействия процессов в ОС UNIX. Взаимодействие процессов по схеме "подчиненный-главный". Общая схема трассировки процессов.

Главный - Подчиненный

```
#include <sys/ptrace.h>
int ptrace(int cmd, int pid, int addr, int data);
```

cmd – код выполняемой команды

pid – идентификатор процесса-потомка

addr – некоторый адрес в адресном пространстве процесса-потомка

data – слово информации.

Главный - Подчиненный

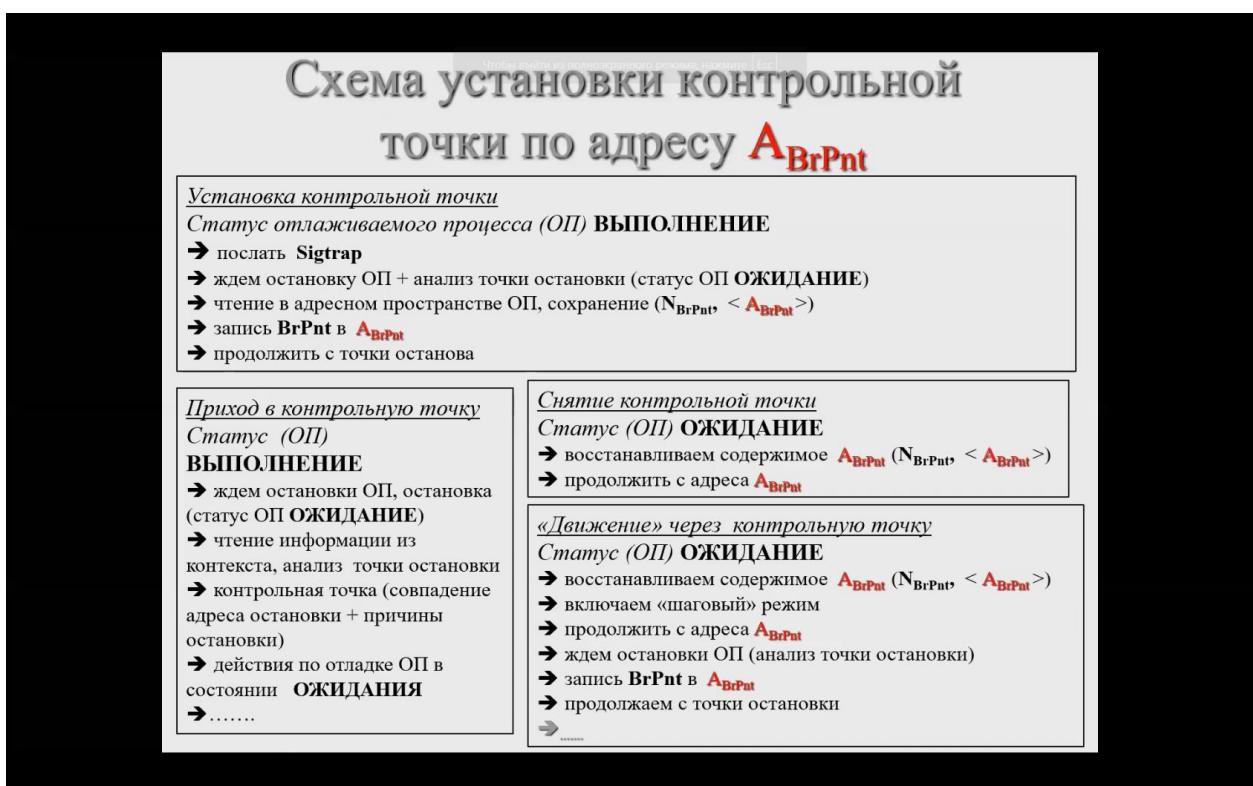
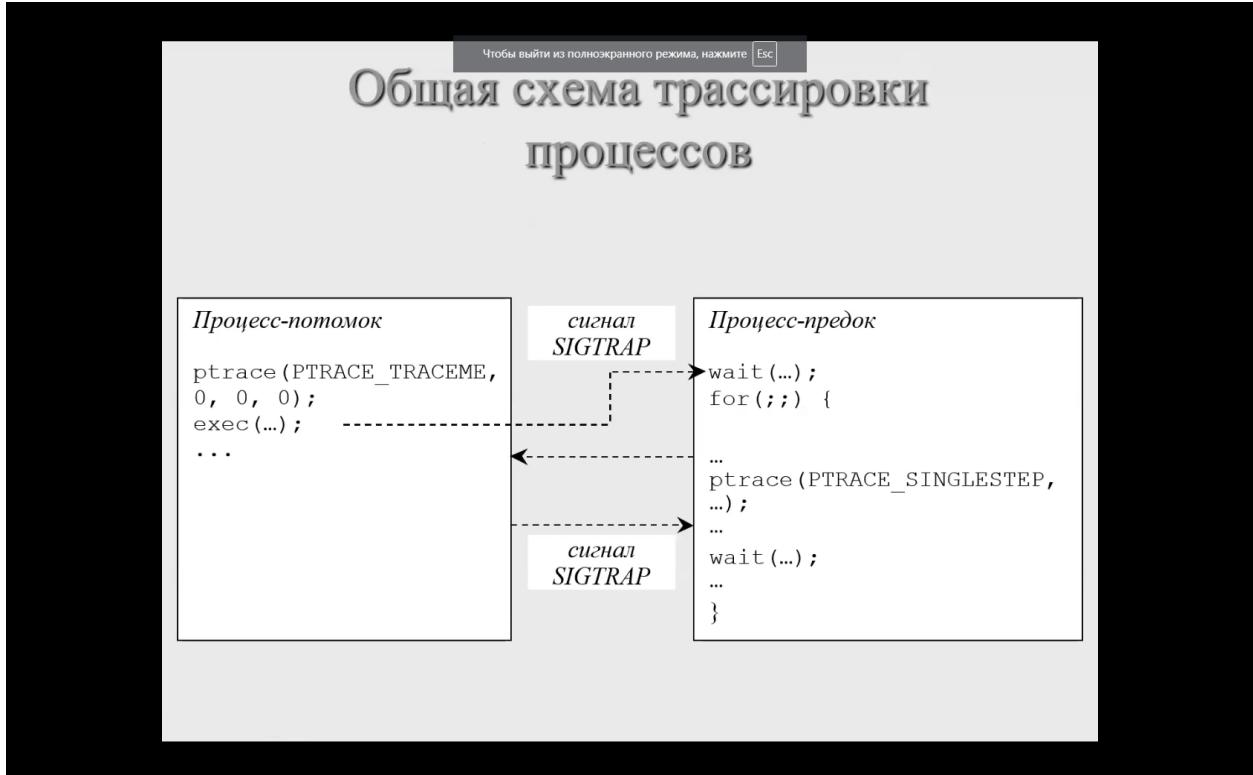
```
int ptrace(int cmd, int pid, int addr, int data);
```

cmd – код команды:

- **группа команд чтения** (сегмент кода, сегмент данных, контекст процесса)

- **группа команд записи** (сегмент кода, сегмент данных, контекст процесса)

- **группа команд управления** (продолжить выполнение, продолжить выполнение с заданного адреса, включить «шаговый режим», завершить процесс, разрешить трассировку)



33. Система межпроцессного взаимодействия ОС UNIX.
Именование разделяемых объектов. Очереди сообщений.
Пример.

34. Система межпроцессного взаимодействия ОС UNIX .
Именование разделяемых объектов. Разделяемая память.
Пример.

35. Система межпроцессного взаимодействия ОС UNIX .
Именование разделяемых объектов. Массив семафоров.
Пример.

36. Сокеты. Типы сокетов. Коммуникационный домен.
Схема работы с сокетами с установлением соединения.

Создание сокета

Необходимые заголовочные файлы и прототип

```
#include <sys/types.h>
#include <sys/socket.h>
int socket ( int domain, int type, int protocol ) ;
```

Параметры

domain — коммуникационный домен:

- AF_UNIX
- AF_INET

type — тип сокета:

- SOCK_STREAM — виртуальный канал
- SOCK_DGRAM — датаграммы

protocol — протокол:

- 0 — автоматический выбор протокола
- IPPROTO_TCP — протокол TCP (AF_INET)
- IPPROTO_UDP — протокол UDP (AF_INET)

Связывание

Необходимые заголовочные файлы и прототип

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

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

```
int bind ( int sockfd, struct sockaddr * myaddr, int  
addrlen );
```

Параметры

sockfd — дескриптор сокета

myaddr — указатель на структуру, содержащую адрес
сокета

Структура
адреса для
домена
AF_UNIX

```
#include <sys/un.h>  
struct sockaddr_un {  
    short sun_family ;           /* == AF_UNIX */  
    char sun_path [ 108 ] ;  
};
```

Связывание

Необходимые заголовочные файлы и прототип

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

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

```
int bind ( int sockfd, struct sockaddr * myaddr, int  
addrlen );
```

Параметры

sockfd — дескриптор сокета

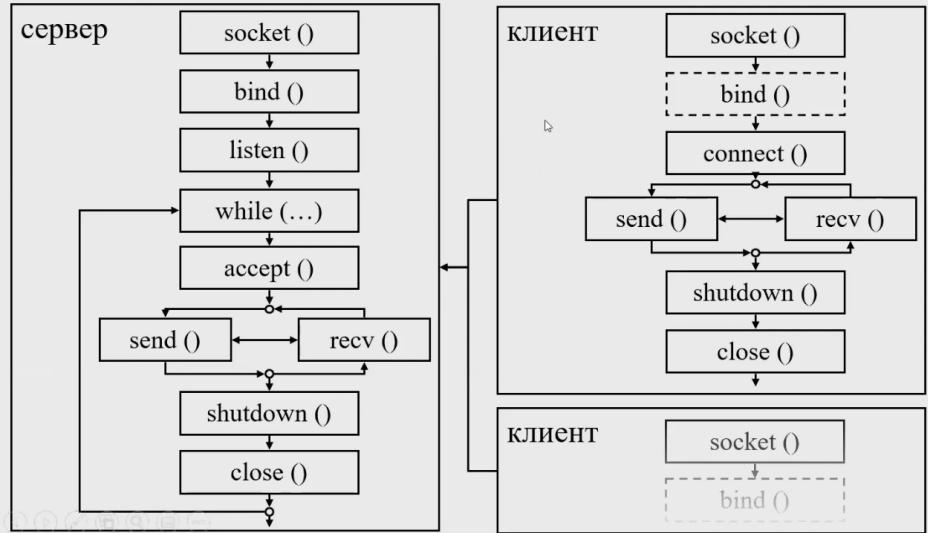
myaddr — указатель на структуру, содержащую адрес
сокета

Структура
адреса для
домена
AF_INET

```
#include <netinet/in.h>  
struct sockaddr_in {  
    short sin_family ;           /* == AF_INET */  
    u_short sin_port ;          /* port number */  
    struct in_addr sin_addr ;   /* host IP address */  
    char sin_zero [ 8 ] ;        /* not used */  
};
```

Предварительное установление соединения

(тип сокета — виртуальный канал или датаграмма)



Прослушивание сокета

Необходимые заголовочные файлы и прототип

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

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

```
int listen (int sockfd, int backlog);
```

Параметры

sockfd — дескриптор сокета

backlog — максимальный размер очереди запросов на соединение

Возвращаемое значение

В случае успешного обращения функция возвращает 0, в случае ошибки — -1. Код ошибки заносится в **errno**.

Запрос на соединение

Необходимые заголовочные файлы и прототип

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

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

```
int connect ( int sockfd, struct sockaddr * serv_addr, int  
addrlen ) ;
```

Параметры

sockfd — дескриптор сокета

serv_addr — указатель на структуру, содержащую адрес сокета, с которым производится соединение

addrlen — реальная длина структуры

Возвращаемое значение

В случае успешного связывания функция возвращает 0, в случае ошибки — -1. Код ошибки заносится в **errno**.

Подтверждение соединения

Необходимые заголовочные файлы и прототип

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

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

```
int accept ( int sockfd, struct sockaddr * addr, int *  
addrlen ) ;
```

Параметры

sockfd — дескриптор сокета

addr — указатель на структуру, в которой возвращается адрес клиентского сокета, с которым установлено соединение (если адрес клиента не интересует, передается NULL).

addrlen — возвращается реальная длина этой структуры.

Возвращаемое значение

- дескриптор нового сокета, соединенного с сокетом клиентского процесса.

Прием и передача данных

Необходимые заголовочные файлы и прототип

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

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

```
int send ( int sockfd, const void * msg, int len,  
          unsigned int flags ) ;
```

```
int recv ( int sockfd, void * buf, int len, unsigned  
          int flags ) ;
```

Возвращаемое значение

Функция возвращает количество переданных байт в случае успеха и -1 в случае неудачи. Код ошибки при этом устанавливается в **errno**.

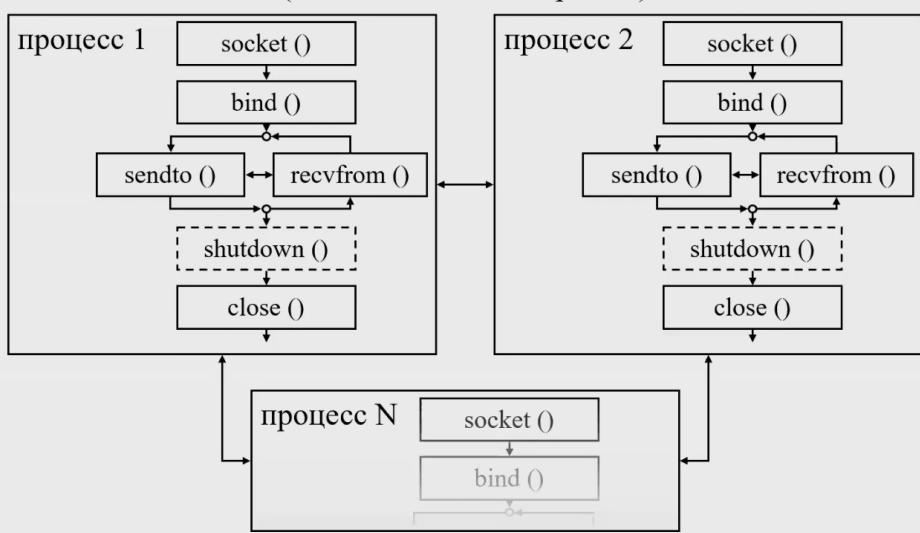
В случае успеха функция возвращает количество считанных байт, в случае неудачи -1 .



37. Сокеты. Схема работы с сокетами без установления соединения.

Сокеты без предварительного соединения

(тип сокета — датаграмма)



Прием и передача данных

Необходимые заголовочные файлы и прототип

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

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

```
int sendto ( int sockfd,
```

```
    const void * msg,
```

```
    int len,
```

```
    unsigned int flags,
```

```
    const struct sockaddr * to,
```

```
    int tolen ) ;
```

```
int recvfrom ( int sockfd,
```

```
    void * buf,
```

```
    int len,
```

```
    unsigned int flags,
```

```
    struct sockaddr * from,
```

```
    int * fromlen ) ;
```



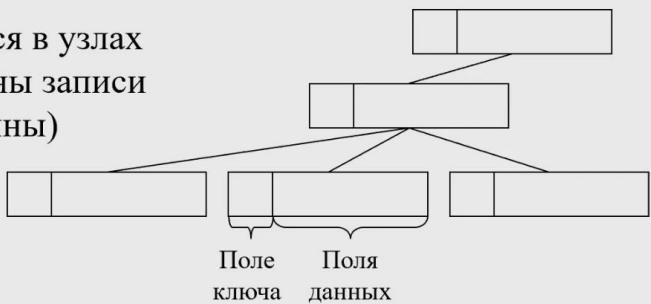
38. Общая классификация средств взаимодействия процессов в ОС UNIX.

39. Файловые системы. Структурная организация файлов. Атрибуты файлов. Основные правила работы с файлами. Типовые программные интерфейсы работы с файлами.

Структурная организация файлов

1. Файл, как **последовательность байтов**
2. Файл, как **последовательность записей переменной длины**
3. Файл, как **последовательность записей постоянной длины**
4. Иерархическая организация файлов (дерево)

Записи находятся в узлах дерева (возможны записи переменной длины)



Атрибуты файла

- Имя
- Права доступа
- Персонификация (создатель, владелец)
- Тип файла
- Размер записи
- Размер файла
- Указатель чтения / записи
- Время создания
- Время последней модификации
- Время последнего обращения
- Предельный размер файла
- ...

Основные сценарии работы с файлами

Начало

Открытие файла (регистрация в системе возможности работы процесса с содержимым файла)



Работа с содержимым файла, с атрибутами файла



Завершение

Закрытие файла — информация системе о завершении работы процесса с открытым файлом

Файловый дескриптор — системная структура данных, содержащая информацию об актуальном состоянии открытого файла.



Типовые программные интерфейсы работы с файлами

open — открытие / создание файла

close — закрытие

read / write — читать, писать (относительно положения указателя чтения / запись)

delete — удалить файл из файловой системы

seek — позиционирование указателя чтение/запись

rename — переименование файла

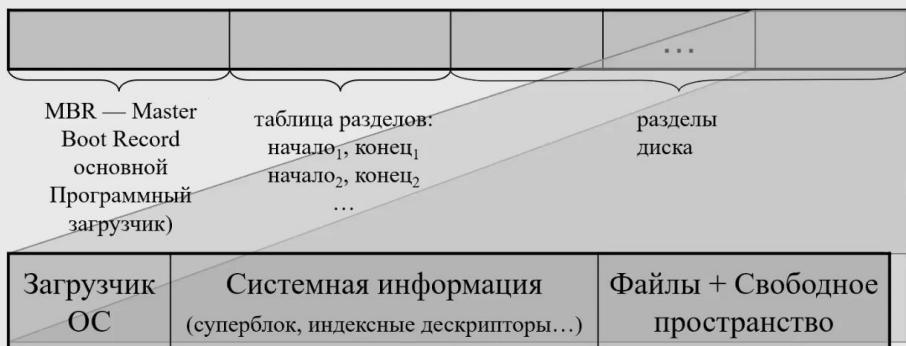
read / write _attributes — чтение, модификация атрибутов файла



40. Файловые системы. Модели реализации файловых систем. Понятие индексного дескриптора.

Подходы в практической реализации файловой системы

Структура «системного» диска



Блок физического HDD → Блок виртуального HDD

Блок файловой системы

Блок файла

Модели реализации файлов

Непрерывные файлы



Name₁ Name₂ Name₃ Name₄ Name₅ Name₆

Достоинства

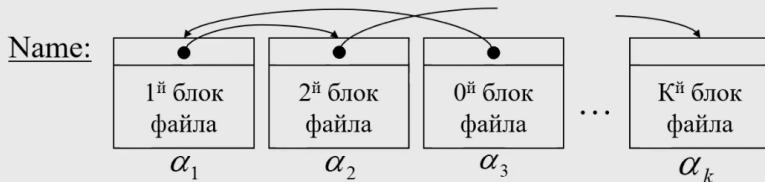
- Простота реализации
- Высокая производительность

Недостатки

- Фрагментация свободного пространства
- Возможность увеличения размера существующего файла

Модели реализации файлов

Файлы, имеющие организацию связанного списка



{ α_i } — множество блоков файловой системы, в которых размещены блоки файла Name.

Достоинства

- Отсутствие фрагментации свободного пространства (за исключением блочной фрагментации)
- Простота реализации
- Эффективный последовательный доступ

Недостатки

- Сложность (неэффективность) организации прямого доступа
- Фрагментация файла по диску
- Наличие ссылки в блоке файла (ситуации чтения 2-х блоков при необходимости чтения данных объемом один блок)

Таблица размещения файловой системы (File Allocation Table — FAT)

0	∅
1	5
2	
3	1
4	
5	0
6	
7	
8	...
...	...
k-1	

номера блоков
файловой системы

блоки файла Name: 3 – 1 - 5

Достоинства

- Возможность использования всего блока для хранения данных файла
- Оптимизация прямого доступа (при полном или частичном размещении таблицы в ОЗУ)

Недостатки

- Желательно размещение всей таблицы в ОЗУ



Индексные узлы (дескрипторы)

Name	Номер 0 ^{го} блока файла
	Номер 1 ^{го} блока файла
	Номер 2 ^{го} блока файла
	Номер 3 ^{го} блока файла
	...
	Номер последнего блока файла

Индексный узел (дескриптор) — системная структура данных, содержащая информацию о размещении блоков конкретного файла в файловой системе.

Достоинства

- Нет необходимости в размещении в ОЗУ информации всей FAT о всех файлах системы, в памяти размещаются атрибуты, связанные только с открытыми файлами

Недостатки

- Размер файла и размер индексного узла (в общем случае прийти к размерам таблицы размещения).

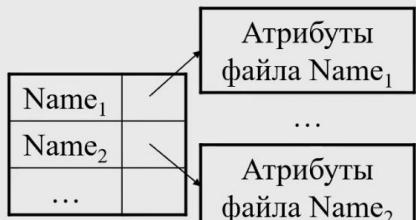
Решение: — ограничение размера файла

— иерархическая организация индексных узлов

Модели организации каталогов

Name ₁	Атрибуты
Name ₂	Атрибуты
...	

Записи каталога фиксированного размера, содержат имя файла и все его атрибуты.



Каталог содержит имя файла и ссылку на атрибуты файла. Размер атрибутов может варьироваться.

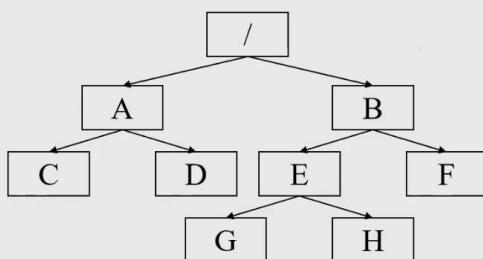
...

Организация «длинных» имен файлов?



Взаимнооднозначное соответствие: имя файла — содержимое файла

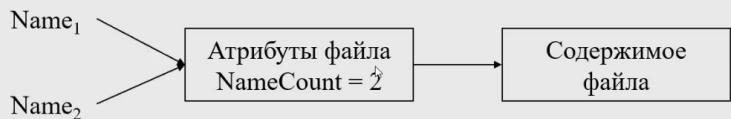
- Содержимому любого файла соответствует единственное имя файла



Взаимнооднозначное соответствие: имя файла — содержимое файла?

2. Содержимому файла может соответствовать два и более имен файла.

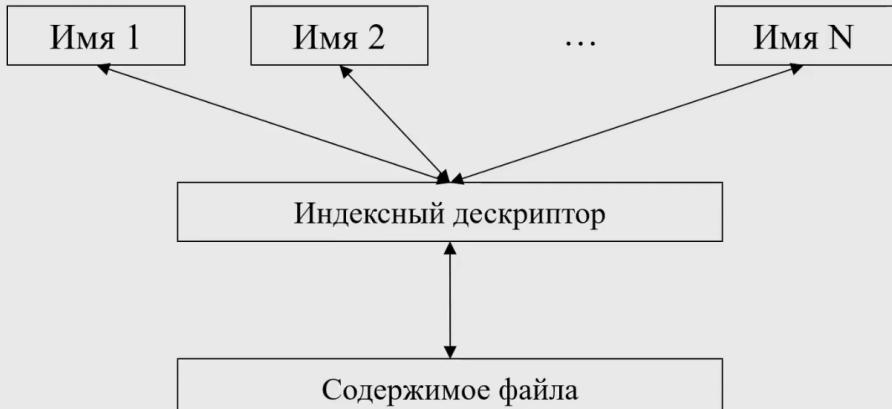
2.1 «Жесткая» связь



2.2 “Символическая” связь



Индексные дескрипторы



41. Файловые системы. Координация использования пространства внешней памяти. Квотирование пространства ФС. Надежность ФС. Проверка целостности ФС.

Координация использования пространства внешней памяти

Проблема — размер блока файловой системы.

«Большой блок»:

- эффективность обмена
- существенная внутренняя фрагментация
(неэффективное использование пространства ВП)

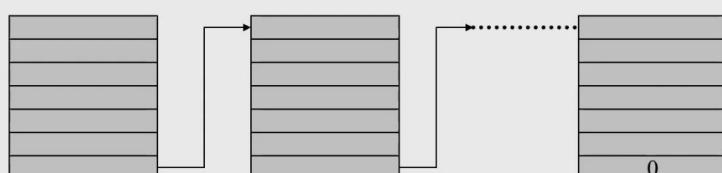
«Маленький блок»:

- эффективное использование пространства ВП
- фрагментация данных файла по диску

Проблема — определение оптимального размера блока.

Учет свободных блоков файловой системы

Связный список свободных блоков



При использовании связного списка свободных блоков в ОЗУ размещается первый блок списка.

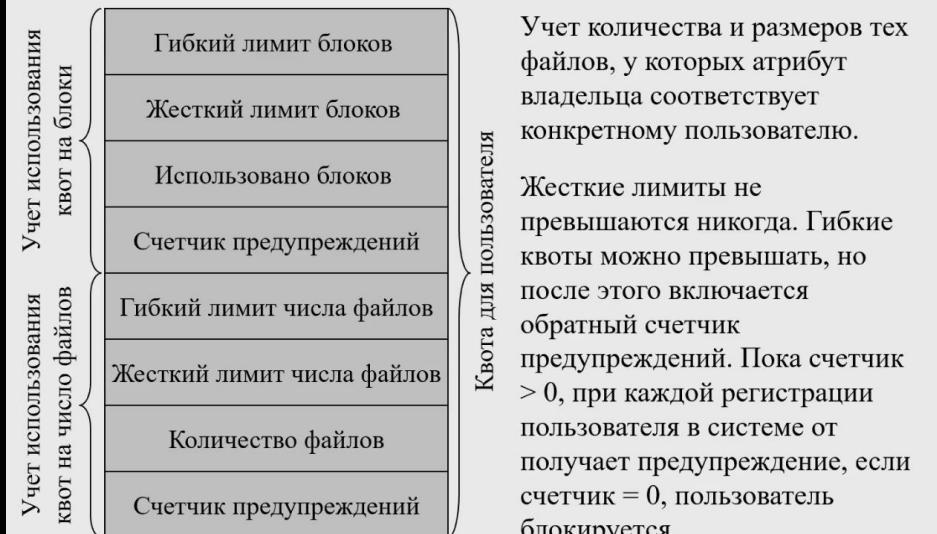
Использование битового массива

Состояние любого блока определяется содержимым бита с номером каждого блока.

Если блок свободен, бит равен 1, занят — 0.

01101.....	10
1001011.....	11
1001011.....	11

Квотирование пространства файловой системы



Надежность файловой системы

- Потеря информации в результате аппаратного или программного сбоя
- Случайное удаление файлов

⇒ Резервное копирование (архивирование):

- Копируются не все файлы файловой системы (избирательность архивирования по типам файлов)
- Инкрементное архивирование (резервное копирование) — единожды создается «полная» копия, все последующие включают только обновленные файлы
- Использование компрессии при архивировании (риск потери всего архива из-за ошибки в чтении/записи сжатых данных)
- Проблема архивирования «на ходу» (во время копирования происходят изменения файлов, создание, удаление каталогов и т.д.)
- Распределенное хранение резервных копий

Надежность файловой системы

Стратегии архивирования

- **Физическая архивация**
 - «Один в один»
 - Интеллектуальная физическая архивация (копируются только использованные блоки файловой системы)
 - Проблема обработки дефектных блоков
- **Логическая архивация** — копирование файлов (а не блоков), модифицированных после заданной даты.

Проверка целостности файловой системы

Проблема — при аппаратных или программных сбоях возможна потеря информации:

- потеря модифицированных данных в «обычных» файлах
- потеря системной информации (содержимое каталогов, списков системных блоков, индексные узлы и т.д.)

Контроль целостности или непротиворечивости файловой системы.

Проверка целостности файловой системы

Контроль непротиворечивости блоков файловой системы:

Модельная стратегия контроля

1. Формируются две таблицы:

- таблица занятых блоков
- таблица свободных блоков

(размеры таблиц соответствуют размеру файловой системы — число записей равно числу блоков ФС)

Изначально все записи таблиц обнуляются.

2. Анализируется список свободных блоков. Для каждого номера свободного блока увеличивается на 1 соответствующая ему запись в таблице свободных
3. Анализируются все индексные узлы. Для каждого блока, встретившегося в индексном узле, увеличивается его счетчик на 1 в таблице занятых блоков
4. Анализ содержимого таблиц и коррекция ситуаций

Проверка целостности файловой системы

Варианты анализа таблиц

- 0 1 2 3 4 5
1.

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

 Таблица занятых блоков

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

 Таблица свободных блоков
- Непротиворечивость файловой системы соблюдена.

Проверка целостности файловой системы

- 0 1 2 3 4 5
2.

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

 Таблица занятых блоков

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

 Таблица свободных блоков
- Пропавший блок
- Оставить как есть, но система «замусоривается».
- Добавить в список свободных блоков файловой системы.
- 0 1 2 3 4 5
3.

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

 Таблица занятых блоков

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

 Таблица свободных блоков
- Дубликат свободного блока – пересоздание списка свободных блоков.

Проверка целостности файловой системы

4.

0	1	2	3	4	5
1	2	0	1	0	1

 Таблица занятых блоков

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

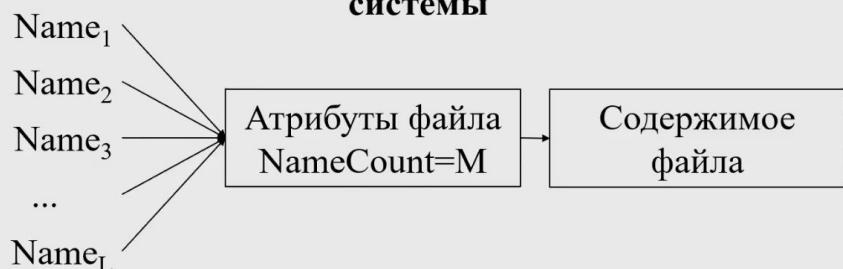
 Таблица свободных блоков
- Дубликат занятого блока \Rightarrow автоматическое решение
максимально затруднено, имеет место потеря информации в одном из файлов.

Действие

1. $Name_1 \rightarrow$ копируется $Name_1^2$
2. $Name_2 \rightarrow$ копируется $Name_2^2$
3. Удаляются $Name_1, Name_2$
4. Запускается переопределение списка свободных блоков
5. Обратное переименование файлов и фиксация факта их возможной проблемности

Проверка целостности файловой системы

Контроль непротиворечивости файлов файловой системы



Возможны варианты:

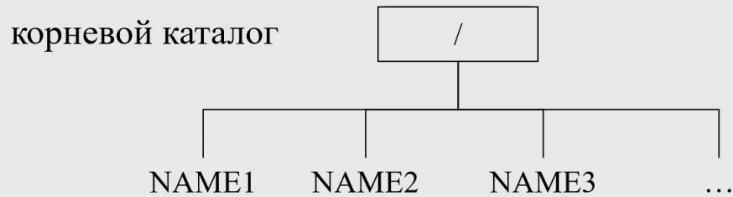
1. $L = M$ — все в порядке
2. $L > M \Rightarrow NameCount = L$
3. $L < M$

42. Примеры реализаций файловых систем. Организация файловой системы ОС UNIX. Виды файлов. Права доступа. Логическая структура каталогов.

Модельная организация каталогов файловых систем

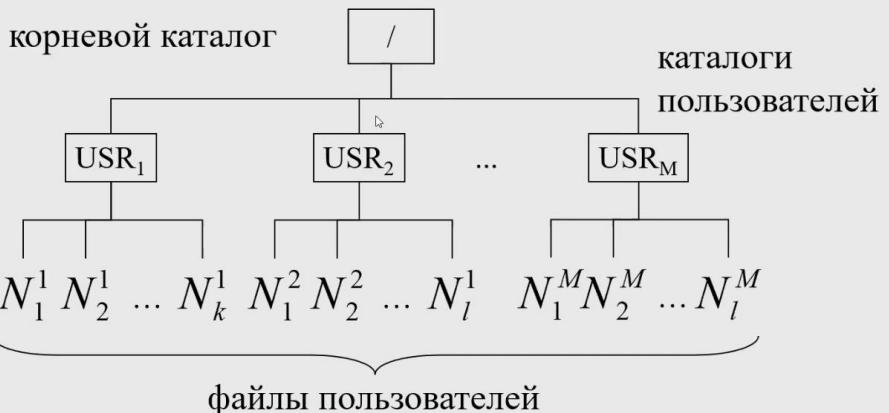
Каталог — компонент файловой системы, содержащий информацию о содержащихся в файловой системе файлах. Каталоги являются специальным видом файлов.

Модель одноуровневой файловой системы



Модельная организация каталогов файловых систем

Модель двууровневой файловой системы

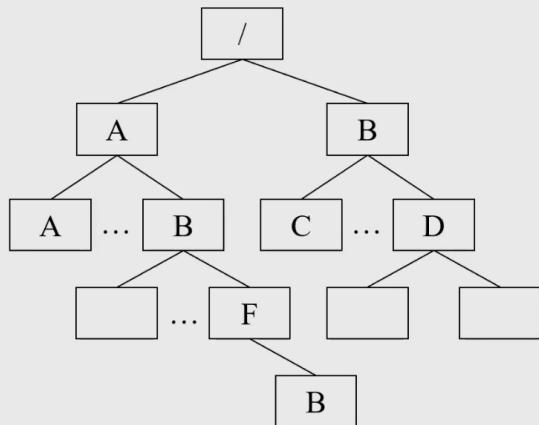


Модельная организация каталогов файловых систем

Иерархические файловые системы

Понятия

- имя файла
- полное имя файла
- относительное имя
- домашний каталог
- текущий каталог



Организация ФС UNIX. Виды файлов

Файл UNIX — это специальным образом именованный набор данных, размещенный в системе.

Виды файлов

- Обычный файл (regular file)
- Каталог (directory)
- Специальный файл устройств (special device file)
- Именованный канал (FIFO)
- Ссылка (link)
- Сокет (socket)

Права доступа

Категории пользователей

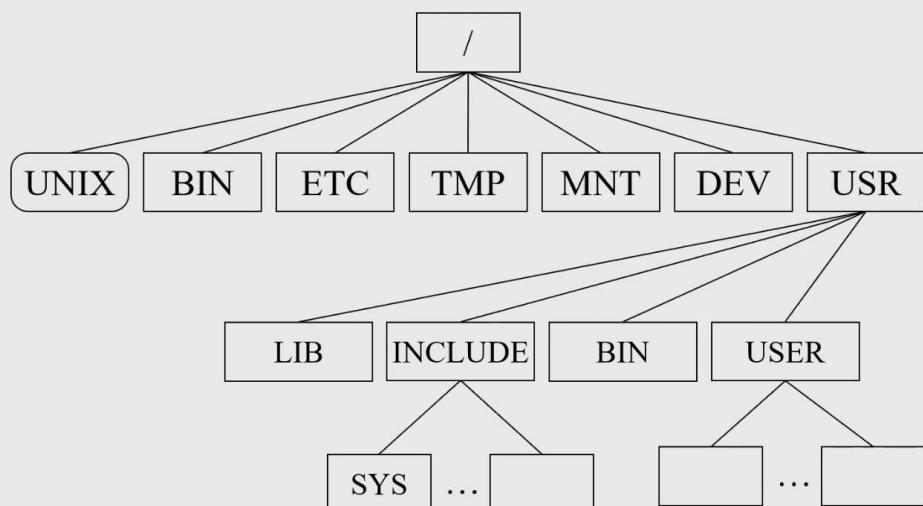
- Пользователь
- Группа
- Все пользователи системы

Права

- На чтение
- На запись
- На исполнение



Логическая структура каталогов



43. Примеры реализаций файловых систем Внутренняя организация ФС. Модель версии UNIX SYSTEM V.

Модель версии System V

Структура ФС

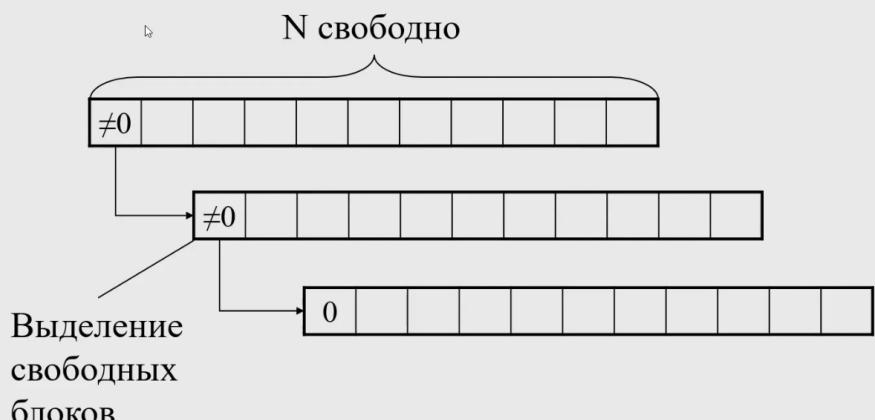
Суперблок	Область индексных дескрипторов	Блоки
-----------	--------------------------------	-------

Суперблок файловой системы содержит оперативную информацию о текущем состоянии файловой системы, а также данные о параметрах настройки.

Индексный дескриптор — специальная структура данных ФС, которая ставится во взаимнооднозначное соответствие с каждым файлом.

Блоки — свободные, занятые под системную информацию, занятые файлами.

Работа с массивами номеров свободных блоков



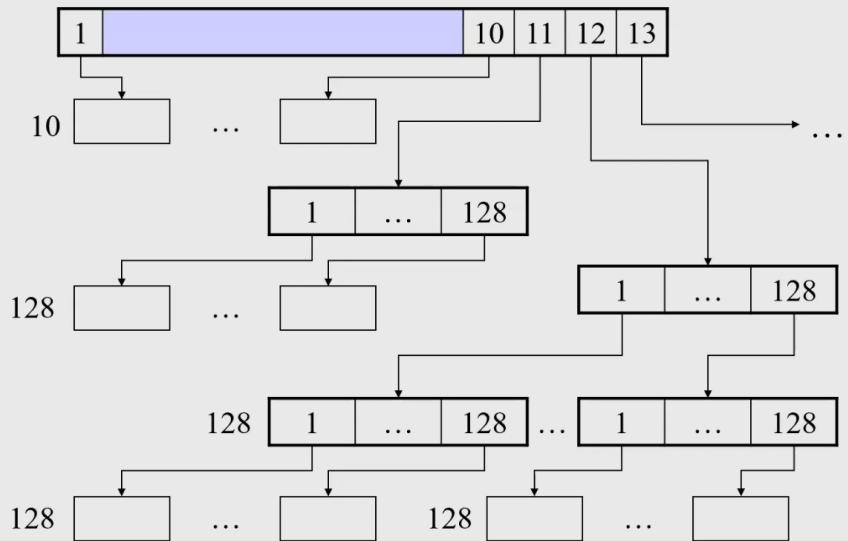
Работа с массивом свободных ИД

- Освобождение ИД
 - Есть свободное место — номер → элемент массива
 - Нет свободного места — номер «забывается»
- Запрос ИД
 - Поиск в массиве
 - Массив пустой — обновление массива
 - Массив не пустой — OK

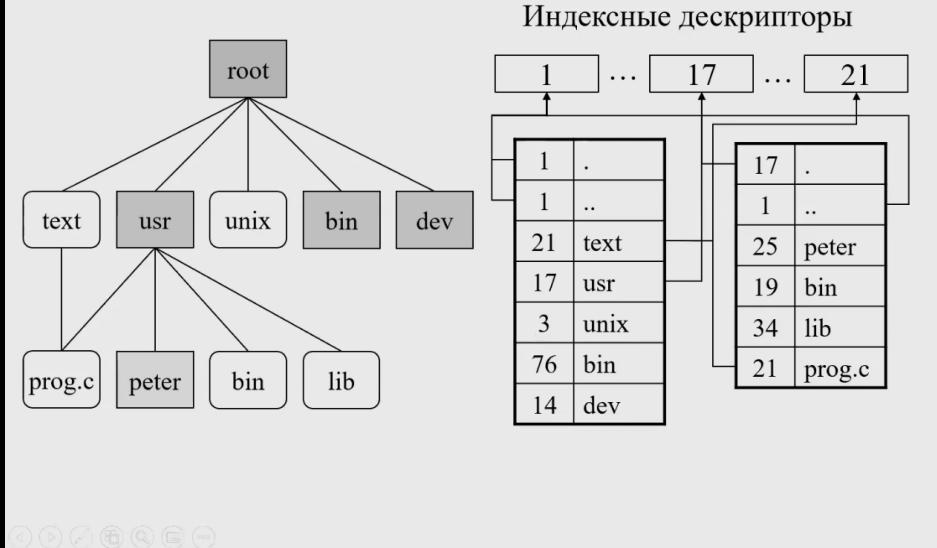
Адресация блоков файла

Модельный пример: блок 512б; ссылка на блок 4б

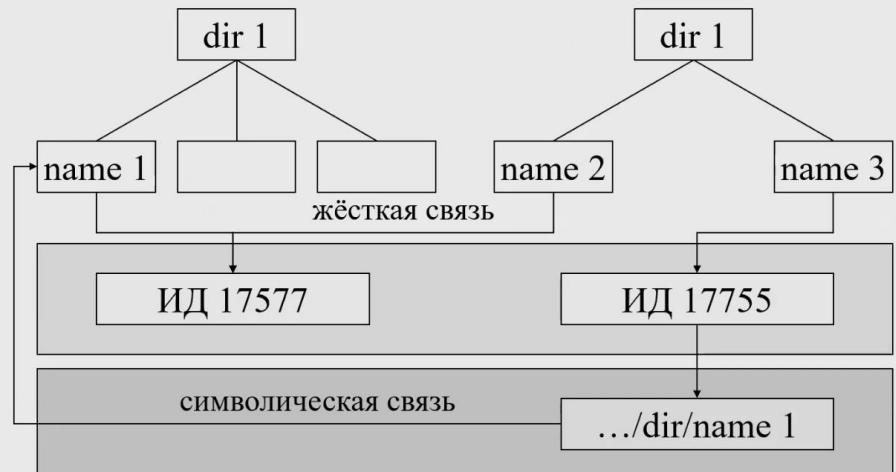
$$(10+128+128^2+128^3) \times 512$$



Файл каталог



Установление связей



Достоинства ФС модели версии System V

- Оптимизация в работе со списками номеров свободных индексных дескрипторов и блоков
- Организация косвенной адресации блоков файлов

Недостатки ФС модели версии System V

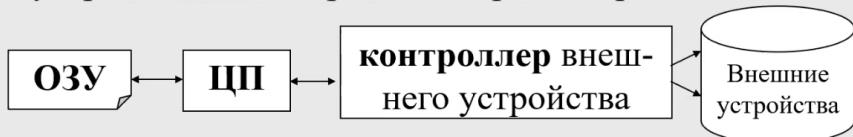
- Концентрация важной информации в суперблоке
- Проблема надежности
- Фрагментация файла по диску
- Ограничения на возможную длину имени файла

44. Управление внешними устройствами. Архитектура организации управления внешними устройствами, основные подходы, характеристики.

Управление внешними устройствами. Архитектура



1. **Непосредственное управление** внешними устройствами центральным процессором



2. **Синхронное управление** внешними устройствами использованием контроллеров внешних устройств
3. **Асинхронное управление** внешними устройствами с использованием контроллеров внешних устройств

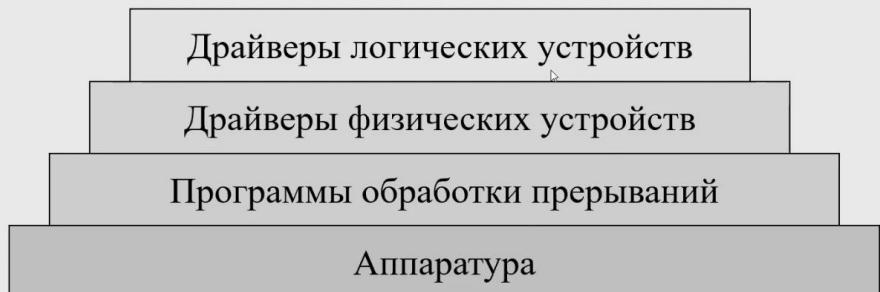
Управление внешними устройствами. Архитектура



4. Использование **контроллера прямого доступа к памяти (DMA)** при обмене.
5. Управление внешними устройствами с использованием **процессора или канала ввода/вывода**.



Программное управление внешними устройствами



Программное управление внешними устройствами

- унификация программных интерфейсов доступа к внешним устройствам (унификация именования, абстрагирование от свойств конкретных устройств)
- обеспечение конкретной модели синхронизации при выполнении обмена (синхронный, асинхронный обмен)
- обработка возникающих ошибок (индикация ошибки, локализация ошибки, попытка исправления ситуации);
- буферизация обмена
- обеспечение стратегии доступа к устройству (распределенный доступ, монопольный доступ);
- планирование выполнения операций обмена

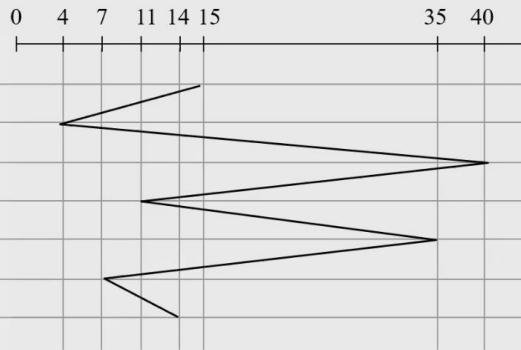
45. Управление внешними устройствами. Планирование дисковых обменов, основные алгоритмы.

Планирование дисковых обменов

Рассмотрим модельную ситуацию:
головка HDD позиционирована на дорожке 15
Очередь запросов к дорожкам: 4, 40, 11, 35, 7, 14

FIFO

Путь головки	L
15 → 4	11
4 → 40	36
40 → 11	29
11 → 35	24
35 → 7	28
7 → 14	7
общ.	135
средн.	22,5



Планирование дисковых обменов

Shortest Service Time First — «жадный» алгоритм — на каждом шаге поиск обмена с минимальным перемещением

SSTF

Путь головки	L
15 → 14	1
14 → 11	3
11 → 7	4
7 → 4	3
4 → 35	31
35 → 40	5
общ.	47
средн.	7,83

LIFO

Путь головки	L
15 → 14	1
14 → 7	7
7 → 35	28
35 → 11	24
11 → 40	29
40 → 4	36
общ.	126
средн.	20,83

Планирование дисковых обменов

PRI — алгоритм, основанный на приоритетах процессов.

SCAN

Путь головки	L
15 → 35	20
35 → 40	5
40 → 14	26
14 → 11	3
11 → 7	4
7 → 4	3
общ.	61
средн.	10,16

«Лифт» — сначала «движение» в одну сторону до «упора», затем в другую, также до «упора»

Для \forall набора запросов
перемещений $\leq 2 \times \text{число_дорожек}$

Планирование дисковых обменов

N-step-SCAN

Разделение очереди на подочереди длины $\leq N$ запросов каждая (из соображений FIFO). Последовательная обработка очередей. Обрабатываемая очередь не обновляется. Обновление очередей, отличных от обрабатываемой.

Борьба с «залипанием» головки (интенсивный обмен с одной и той же дорожкой).

46. Управление внешними устройствами. Организация RAID систем, основные решения, характеристики.

RAID системы

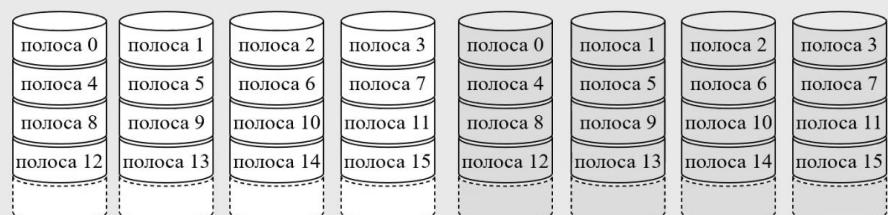
RAID (Redundant Array of Independent (Inexpensive) Disks) — избыточный массив независимых (недорогих) дисков (1987, Беркли, Калифорния, США).

RAID система — набор физических дисковых устройств, рассматриваемых операционной системой, как единое дисковое устройство (данные распределяются по физическим устройствам, образуется избыточная информация, используемая для контроля и восстановления информации).

Уровни RAID

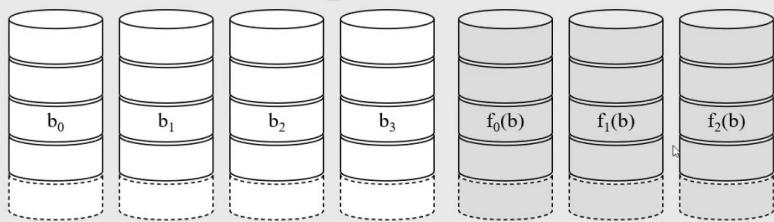


RAID 0 (без избыточности)

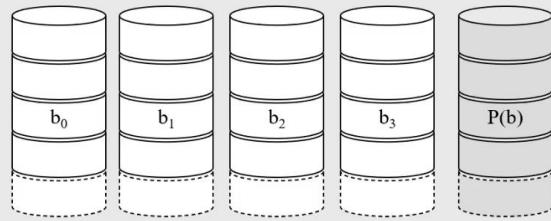


RAID 1 (зеркалирование)

Уровни RAID



RAID 2 избыточность с кодами Хэмминга



RAID 3 (четность с чередующимися битами)

Пример: 4 диска данных, один — четности: Потеря данных на первом диске

$$X4(i) = X3(i) \text{ XOR } X2(i) \\ \text{XOR } X1(i) \text{ XOR } X0(i)$$

$$X1(i) = X4(i) \text{ XOR } X3(i) \\ \text{XOR } X2(i) \text{ XOR } X0(i)$$

Уровни RAID



RAID 4

Пример: 4 диска данных, один – четности:

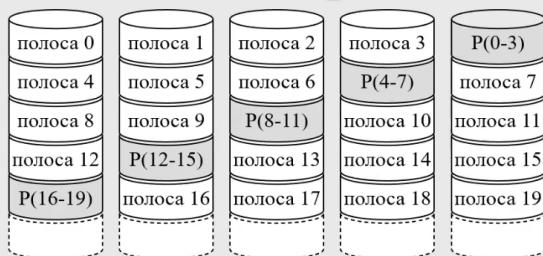
Изначально:

$$X4(i) = X3(i) \text{ XOR } X2(i) \text{ XOR } X1(i) \text{ XOR } X0(i)$$

После обновления полосы на диске X1:

$$X4_{\text{new}}(i) = X4(i) \text{ XOR } X1(i) \text{ XOR } X1_{\text{new}}(i)$$

Уровни RAID

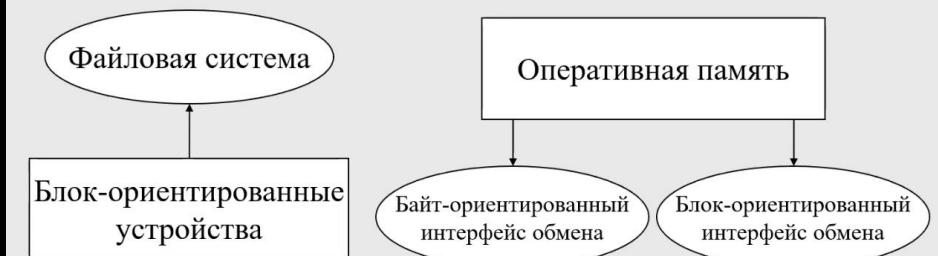


RAID 5 (распределенная четность — циклическое распределение «четности»)

47. Внешние устройства в ОС UNIX. Типы устройств, файлы устройств, драйверы.

Файлы устройств, драйверы

- Иерархия драйверов
- Специальные файлы устройств (/dev)
 - Байт-ориентированные устройства
 - Блок-ориентированные устройства



Файлы устройств

- Содержимое файлов устройств размещается исключительно в соответствующем индексном дескрипторе
- Структура ИД файла устройства:
 - «Старший номер» (major number) устройства
 - Тип устройства
 - «Младший номер» (minor number) устройства
- Системные таблицы драйверов устройств:
 - **bdevsw**
 - **cdevsw**

Системные таблицы драйверов устройств

- Запись таблицы — **коммутатор устройства**
- Типовой набор точек входа в драйвер:
 - **βopen()**, **βclose()**
 - **βread()**, **βwrite()**
 - **βioctl()**
 - **βintr()**
 - **βstrategy()**

Ситуации, вызывающие обращение к функциям драйвера

- Старт системы, определение ядром состава доступных устройств
- Обработка запроса ввода/вывода
- Обработка прерывания, связанного с данным устройством
- Выполнение специальных команд управления

Включение/удаление драйверов в систему

- «Жёсткое», статистическое встраивание драйверов в код ядра
- Динамическое включение драйвера в систему
 - Загрузка и динамическое связывание драйвера с кодом ядра
 - Инициализация драйвера и соответствующего ему устройства

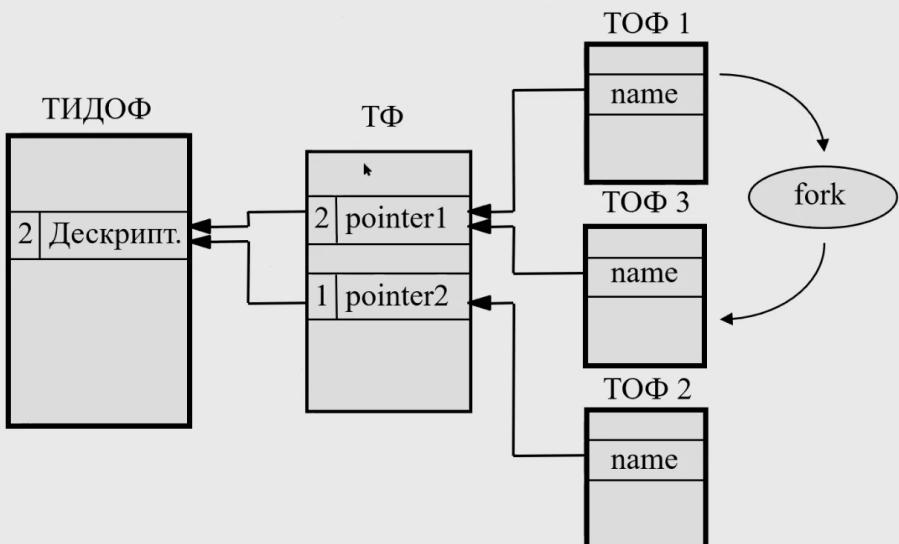
48. Внешние устройства в ОС UNIX. Системная организация обмена с файлами. Буферизация обменов с блокоориентированными устройствами.

Организация обмена данными с файлами

- Таблица индексных дескрипторов открытых файлов (размещается в памяти ядра ОС)
- Таблица файлов (размещается в памяти ОС)
- Таблица открытых файлов



Пример



Буферизация при блок-ориентированном обмене

1. Поиск заданного блока в буферном пуле. Нашли-переходим на шаг 4
2. Поиск буфера в буферном пуле для чтения и размещения заданного блока
3. Чтение блока в найденный буфер
4. Изменение счётчика времени во всех буферах
5. Содержимое буфера передаётся в качестве результата

Буферизация при блок-ориентированном обмене

- Оптимизация работы ОС, за счет минимизации реальных обращений к физическому устройству
- Недостатки:
 - Критичность к несанкционированным отключениям питания
 - Разорванность во времени факта обращения к системе за обменом и реальным обменом

Борьба со сбоями

- Наличие параметра, определяющего периоды времени, через которые осуществляется сброс системных данных, который может оперативно меняться
- Пользовательская команда SYNC
- Избыточность системы, позволяющая восстанавливать информацию

49. Управление оперативной памятью. Одиночное непрерывное распределение. Распределение разделами. Распределение перемещаемыми разделами.

Управление оперативной памятью

Основные задачи

1. Контроль состояния каждой единицы памяти (свободна/распределена)
2. Стратегия распределения памяти (кому, когда и сколько памяти должно быть выделено)
3. Выделение памяти (выбор конкретной области, которая должна быть выделена)
4. Стратегия освобождения памяти (процесс освобождает, ОС “забирает” окончательно или временно)

Управление оперативной памятью

Стратегии и методы управления

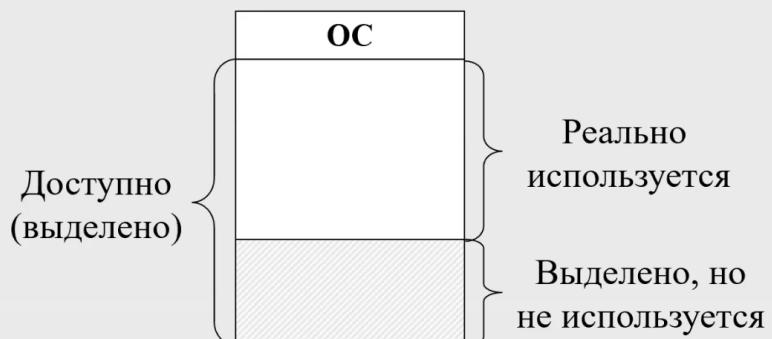
1. Одиночное непрерывное распределение
2. Распределение разделами
3. Распределение перемещаемыми разделами
4. Страницное распределение
5. Сегментное распределение
6. Сегментно-страницное распределение

План рассмотрения стратегий управления

1. Основные концепции
2. Необходимые аппаратные средства
3. Основные алгоритмы
4. Достоинства, недостатки

Одиночное непрерывное распределение

Основные концепции



Одиночное непрерывное распределение

Необходимые аппаратные средства

- Регистр границ + режим ОС / режим пользователя
- Если ЦП в режиме пользователя попытается обратиться в область ОС, то возникает прерывание

Алгоритмы: очевидны.

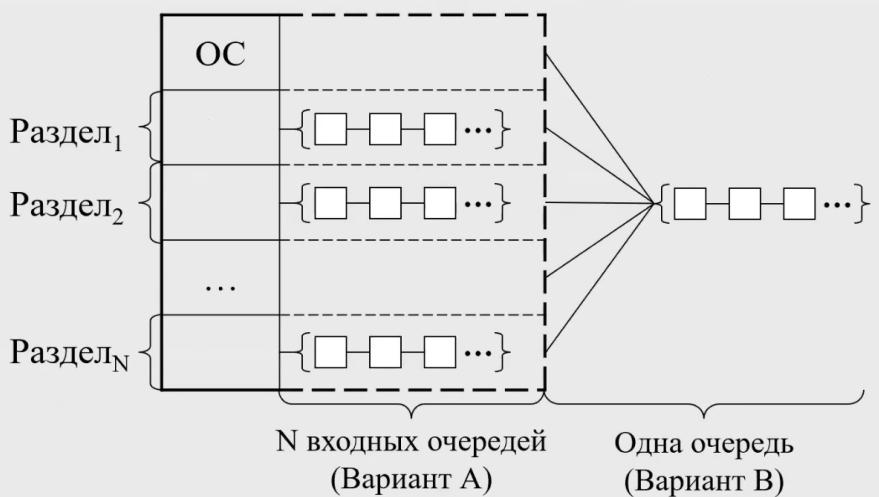
Достоинства: простота.

Недостатки

1. Часть памяти не используется
2. Процессом/заданием память занимается все время выполнения
3. Ограничение на размеры задания

Распределение неперемещаемыми разделами

Основные концепции



Распределение неперемещаемыми разделами

Необходимые аппаратные средства

1. Два регистра границ
2. Ключи защиты (PSW)



Распределение неперемещаемыми разделами

Алгоритмы

Модель статического определения разделов

А. Сортировка входной очереди процессов по отдельным очередям к разделам. Процесс размещается в разделе минимального размера, достаточного для размещения данного процесса. В случае отсутствия процессов в каких-то подочередях — неэффективность использования памяти.

Распределение неперемещаемыми разделами

Алгоритмы

Модель статического определения разделов

Б. Одна входная очередь процессов

1. Освобождение раздела \Rightarrow поиск (в начале очереди) первого процесса, который может разместиться в разделе.

Проблема: большие разделы \leftrightarrow маленькие процессы

2. Освобождение раздела \Rightarrow поиск процесса максимального размера, не превосходящего размер раздела.

Проблема: дискриминация «маленьких» процессов

3. Оптимизация варианта 2. Каждый процесс имеет счетчик дискриминации. Если значение счетчика процесса $\geq K$, то обход его в очереди невозможен

Распределение неперемещаемыми разделами

Достоинства

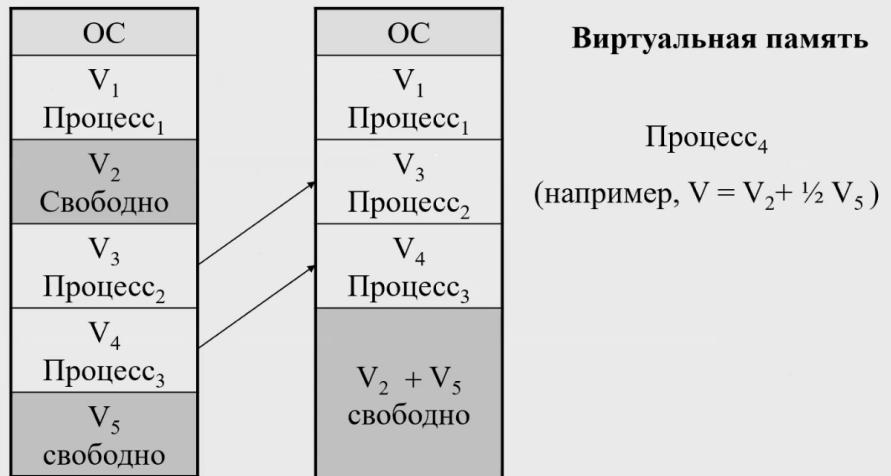
1. Простое средство организации мультипрограммирования
2. Простые средства аппаратной поддержки
3. Простые алгоритмы

Недостатки

1. Фрагментация
2. Ограничение размерами физической памяти
3. Весь процесс размещается в памяти — возможно неэффективное использование

Распределение перемещаемыми разделами

Основные концепции



Распределение перемещаемыми разделами

Необходимые аппаратные средства

1. Регистры границ + регистр базы
2. Ключи + регистр базы

Алгоритмы:

Распределение перемещаемыми разделами

Достоинства

1. Ликвидация фрагментации

Недостатки

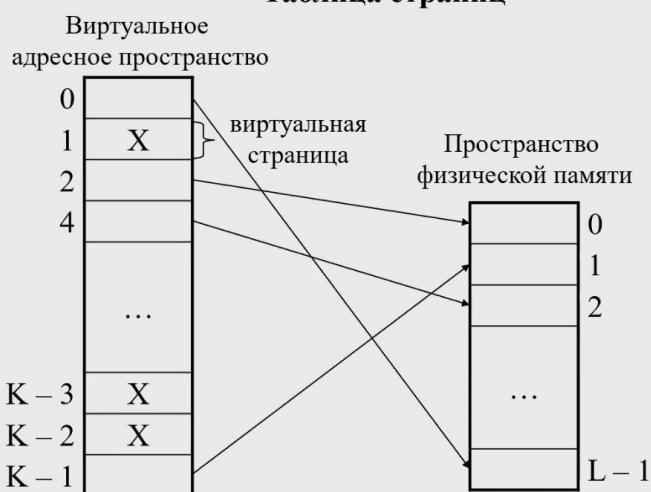
1. Ограничение размером физической памяти
2. Затраты на перекомпоновку

50. Управление оперативной памятью. Страницное распределение.

Страницное распределение

Основные концепции

Таблица страниц



Страницное распределение

Основные концепции

Таблица страниц — отображение номеров виртуальных страниц на номера физических.

Проблемы

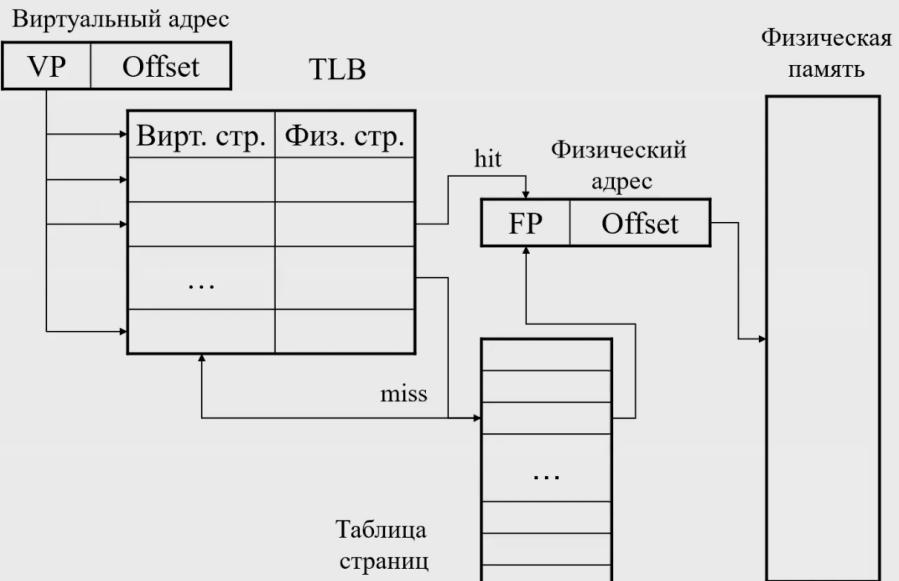
1. Размер таблицы страниц (количество 4КВ страниц при 32-х разрядной адресации — 1 000 000; любой процесс имеет собственную таблицу страниц)
2. Скорость отображения

Страницное распределение

Необходимые аппаратные средства

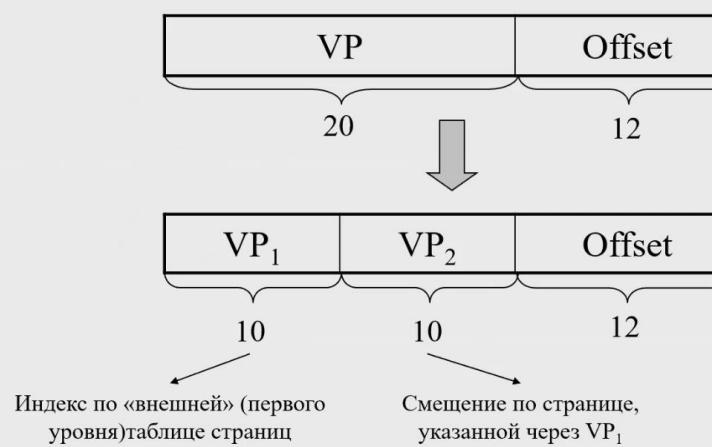
1. Полностью аппаратная таблица страниц (стоимость, полная перегрузка при смене контекстов, скорость преобразования)
2. Таблица страниц в ОЗУ + Регистр начала таблицы страниц в памяти (простота, управление смены контекстов, медленное преобразование)
3. Гибридные решения

TLB (Translation Lookaside Buffer)



Иерархическая организация таблицы страниц

Двухуровневая организация



Иерархическая организация таблицы страниц

Проблема — размер таблицы страниц.

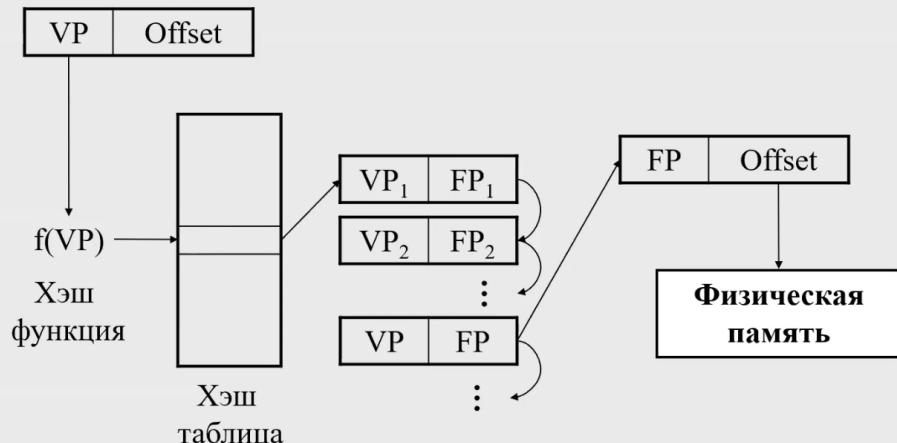
Объем виртуальной памяти современного компьютера — $2^{32} \dots 2^{64}$ байт

Пример

$$\left\{ \begin{array}{l} V_{\text{вирт.}} = 2^{32} \\ V_{\text{стр.}} = 2^{12} \text{ (4KB)} \end{array} \right. \Rightarrow \text{Количество виртуальных страниц} — 2^{20} (\underline{\text{много}})$$

Решение — использование многоуровневых таблиц страниц ($2^x, 3^x, 4^x$)

Использование хэш-таблиц (функция расстановки)



Инвертированные таблицы страниц



Проблема — поиск по таблице (хэширование)

Решение проблемы перезагрузки таблицы страниц при смене обрабатываемых ЦП процессов



Замещение страниц

Проблема загрузки «новой» страницы в память.

Необходимо выбрать страницу для удаления из памяти (с учетом ее модификации и пр.)

Алгоритм NRU (Not Recently Used — не использовавшийся в последнее время)

Используются биты статуса страницы в записях таблицы страниц

R — обращение
M — изменение

} устанавливаются аппаратно

обнуление — программно (ОС)

Замещение страниц

Алгоритм

1. При запуске процесса M и R для всех страниц процесса обнуляются
2. По таймеру происходит обнуление всех битов R
3. При возникновении страничного прерывания ОС делит все страницы на классы:
 - Класс 0: $\begin{cases} M=0 \\ R=0 \end{cases}$
 - Класс 1: $\begin{cases} M=1 \\ R=0 \end{cases}$
 - Класс 2: $\begin{cases} M=0 \\ R=1 \end{cases}$
 - Класс 3: $\begin{cases} M=1 \\ R=1 \end{cases}$
4. Случайная выборка страницы для удаления в непустом классе с минимальным номером

Замещение страниц

Алгоритм FIFO

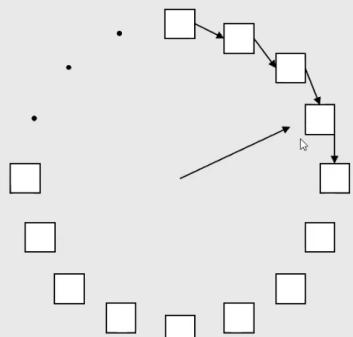
«Первым прибыл — первым удален» — простейший вариант FIFO. Проблема «справедливости»

Модификация алгоритма (алгоритм вторая попытка)

1. Выбирается самая «старая страница». Если R=0, то она заменяется
2. Если R = 1, то R обнуляется, обновляется время загрузки страницы в память (т.е. переносится в конец очереди). На п.1

Замещение страниц

Алгоритм «Часы»



1. Если $R = 0$, то выгрузка страницы и стрелка на позицию вправо
2. Если $R = 1$, то R обнуляется, стрелка на позицию вправо и на п.1



Замещение страниц

Алгоритм NFU (Not Frequently Used — редко использовавшаяся страница)

Для каждой физической страницы i – программный счетчик Count_i

0. Изначально Count_i – обнуляется для всех i .

1. По таймеру $\text{Count}_i = \text{Count}_i + R_i$

Выбор страницы с минимальным значением Count_i

Недостатки

- «помнит» старую активность
- при большой активности, возможно переполнение счетчика

Замещение страниц

Модификация NFU — алгоритм старения

Модификация:

1. Значение счетчика сдвигается на 1 разряд вправо
2. Значение R добавляется в крайний левый разряд счетчика



51. Управление оперативной памятью. Сегментное распределение.

Сегментная организация памяти

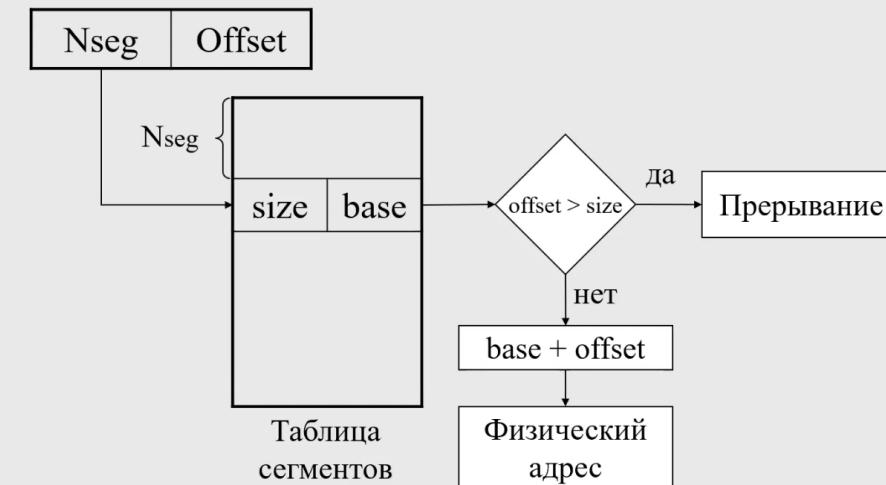
Основные концепции

- Виртуальное адресное пространство представляется в виде совокупности сегментов
- Каждый сегмент имеет свою виртуальную адресацию (от 0 до $N - 1$)
- Виртуальный адрес: <номер_сегмента, смещение>

Сегментная организация памяти

Необходимые аппаратные средства

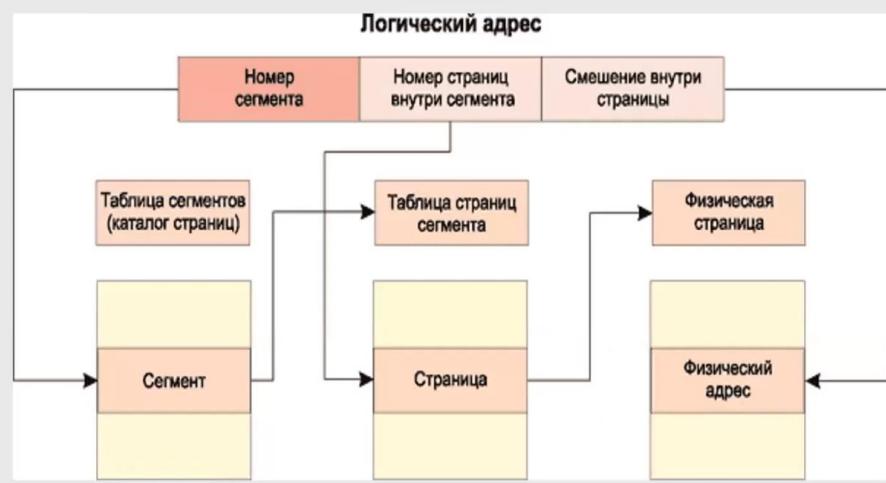
Виртуальный адрес:



Сегментно-страничная организация памяти

Основные концепции:

Nseg | Npage | Offset



52. Вычислительная система. Кэширование информационных потоков на уровнях аппаратуры и ОС.

53. Язык программирования С. Общая характеристика. Типы, данные, классы памяти. Правила видимости.

Структура программы. Препроцессор. Интерфейс с ОС UNIX.