**1 Тема 1. Назначение и функции ОС**

1. ОС как базовая часть систем обработки данных (СОД).

«Зачем нужны компьютеры?» - «Компьютеры нужны для обработки данных»

**Система обработки данных (СОД)**

**Общая классификации СОД:**

* сосредоточенные (централизованные) системы
* распределенные системы

**ОС** — это программное обеспечение (ПО), которое устанавливается на аппаратную часть вычислительной техники. ЭВМ:

* техническую часть (Аппаратная часть)
  + МП
  + ОЗУ и ПЗУ
  + Винчестер и CD
* программную часть (ПО)
  + системное ПО
  + прикладное ПО
  + инструментальное ПО

**Системное ПО:**

* операционные системы
* программы–оболочки
* операционные оболочки
* драйверы и утилиты

1. Серверные ОС и рабочие станции.

**Сервер и Рабочая станция в контексте:**

* аппаратного обеспечения ЭВМ
* программного обеспечения

**В настоящее время:**

* сервер (server) — дистрибутив ОС, с установленным системным и прикладным ПО, ориентированные на выполнение функций сервера;
* рабочая станция (desktop) — дистрибутив ОС или ЭВМ, предназначенная для интерактивной работы с пользователем, на которой установлено соответствующее клиентское прикладное программное обеспечение.

1. ОС как виртуальная машина.

**Целевое назначение ОС** — создание виртуальной машины или среды исполнения для работы системного, прикладного и инструментального ПО компьютера.

Любая ОС, являясь средством распределения ресурсов и организуя по определенным правилам управление процессами, скрывает от пользователя и его приложений реальные аппаратные и иные ресурсы, заменяя их некоторой абстракцией.

**Общие характеристики виртуальной машины:**

* единообразная память практически неограниченного объема;
* произвольное количество параллельных виртуальных процессоров;
* произвольное количество внешних устройств (виртуальных), способных работать с памятью виртуальной машины параллельно или последовательно.

**В результате** прикладной программист рассматривает ядро ОС как некоторую ***абстрактную (виртуальную) машину***, которая является средой для выполнения его программ.

Систематизируя различные системные вызовы и развивая идею виртуальной машины, мы с точностью до терминологии можем утверждать, что каждое ядро ОС, абстрагирует три базовых концепции: **файл, пользователь и процесс**.

1. Многослойная структура ОС.

**Классическая архитектура ОС** основана на:

1) концепции иерархической многоуровневой машины;

2) концепции привилегированного ядра ОС;

3) концепции пользовательского режима работы транзитных модулей.

**Модули ядра** выполняют базовые функции ОС:

1) управление процессами, памятью;

2) устройствами ввода-вывода и тому подобное.

**Иерархическая архитектура ОС:**

1. Утилиты, системные программы, приложения пользователей
2. Интерфейс системных вызовов API
3. Менеджер ресурсов, Файловая система, вирт. память,
4. Базовые механизмы ядра, Машинно-зависимые модули ядра ОС
5. Средства аппаратной поддержки ядра

**Такая организация существенно упрощает разработку системы и позволяет:**

1) сначала, "сверху вниз" определить функции слоёв и межслойные интерфейсы;

2) при детальной реализации, двигаясь "снизу вверх", – можно наращивать мощность функций слоёв;

3) модули каждого слоя можно изменять без необходимости изменений в других слоях, но не меняя межслойных интерфейсов!**Привилегированный режим**

1. ОС как базовая часть ПО ЭВМ.

**Ядро ОС**

В ядре решаются внутрисистемные задачи организации вычислительного процесса, недоступные для приложений.

**Особый класс функций ядра** служит для поддержки приложений, создавая

для них так называемую прикладную программную среду. Все приложения обращаются к ядру со специальными запросами – **системными вызовами.**

Функции ядра, которые могут вызываться приложениями, образуют интерфейс прикладного программирования – **API**

**В результате** прикладной программист рассматривает ядро ОС как некоторую ***абстрактную (виртуальную) машину***, которая является средой для выполнения его программ.

Систематизируя различные системные вызовы и развивая идею виртуальной машины, мы с точностью до терминологии можем утверждать, что каждое ядро ОС, абстрагирует три базовых концепции: **файл, пользователь и процесс**.

1. Режимы ядра и пользователя.

**монолитное ядро ОС** – 2t

**микроядерная архитектура ОС** – 4t

**Суть:**

* в состав микроядра входят машинозависимые модули, а также модули, выполняющие базовые механизмы обычного ядра.
* все остальные, более высокоуровневые функции ядра, оформляются как модули, работающие в пользовательском режиме.

1) менеджеры ресурсов, являющиеся неотъемлемой частью обычного ядра, становятся "периферийными" модулями, работающими в пользовательском режиме;

2) внешние по отношению к микроядру компоненты ОС реализуются как обслуживающие

процессы;

3) между собой эти модули взаимодействуют как равноправные партнёры с помощью обмена сообщениями, которые передаются через микроядро.

**Менеджеры ресурсов**, вынесенные в пользовательский режим, называются **серверами**

**ОС.**

1. Ядро и модули ОС.

**Модули ядра выполняют основные базовые функции ОС**:

1) управление процессами и памятью;

2) управление устройствами ввода-вывода и другими элементами.\

**Драйвера**

**Любое ядро ОС** является одной большой программной, которая**:**

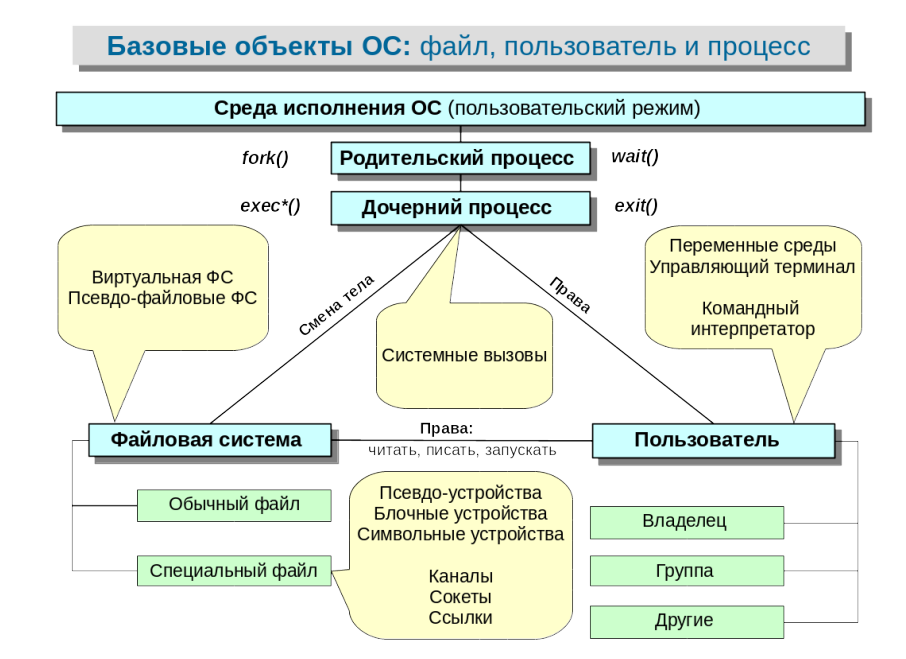
1) выполняется в защищённом режиме (режиме ядра);

2) выполняется в своём собственном адресном пространстве

Перед компиляцией ядра запускается **программа-конфигуратор** (Статические или динамические модули)

Когда ядро ОС загружено в память ЭВМ и начинает работать, запускается **первый пользовательский процесс *init***, имеющий PID=1, обеспечивающий дальнейшую загрузку необходимых модулей.

1. Три базовых концепции ОС: файл, пользователь, процесс.



**концепция файла:**

«Все есть файл»

Три операции: r - чтение, w - запись и x – запуск

**Файловая система** - поименованная совокупность обычных и специальных файлов.

**Обычный и специальный** (имя и специализация) **файл**

**концепция пользователя:**

владелец, группа и другие

**Концепция процесс –** процесс - запущенная программа или задача.

**PID**

1. Системные вызовы fork(...) и exec(...).

**Функция fork(...)** **-** дублирует существующий процесс

**Функции exec\*(...)** **–** запуск программы (файла):

а) после всех проверок на права запуска, указанный файл загружается в пространство дочернего процесса.

б) загруженной программе передаются все ресурсы дочернего процесса, включая открытые и созданные файлы.

1. Дистрибутивы ОС.

Когда говорят, что **на ЭВМ установлена некоторая ОС**, то обычно подразумевают некоторый её дистрибутив, включающий конкретное ядро ОС и другое системное ПО, а также прикладное ПО и системы разработки

**Выбор конкретного дистрибутива предполагает учёт многих факторов.**

Для целей обучения выбран **64-битный базовый дистрибутив Arch Linux.**

**2 Тема 2. BIOS, UEFI и загрузка ОС**

1. Архитектура x86. BIOS и его функции.

**x86:**

1) объем адресуемой памяти 1 Мбайт;

2) работа в реальном режиме: защищённый режим работы отсутствует.

* для выполнения ариф. и лог. операций служат **регистры общего назначения**.
* **Индексные и указательные регистры** служат для формирования массивов.
* **регистр состояния** содержит биты, которые изменяются в процессе выполнения различных операций;
* **сегментные регистры** являются указателями на начало областей оперативной памяти (сегменты);
* **указатель команды** — смещение относительно начала сегмента команд, определяемого регистром CS.

Вызов функции или команда: **CALL** адрес;

Прерывание или команда: **INTERRUPT** номер;

**Три способа** ввода-вывода:  
Программируемый ввод-вывод

Ввод-вывод, управляемый прерываниями

Прямой доступ к памяти (direct memory access – *DMA*)

Режим работы шины - **пословный** или **поблочный (**захват цикла**)**.

Для реализации приоритетной схемы управления используется общий **диспетчер прерываний ОС**.

**BIOS** записана в микросхему **EEPROM** (ПЗУ) и обеспечивает:

• начальное тестирование компьютера;

• последующую загрузку ОС.

Для сохранения настроек BIOS используется микросхема **CMOS-памяти.** В CMOS хранятся параметры конфигурации компьютера. Суммарный объем памяти CMOS составляет **256 байт.**

1. Этапы и режимы POST. UEFI и его стандартизация.

**POST** (Power-On Self-Test)

**Сокращённый тест**, включает четыре этапа:

1. Проверку целостности программ BIOS в ПЗУ, используя контрольную сумму.

2. Обнаружение и инициализацию основных контроллеров, системных шин и подключённых устройств.

3. Выполнение программ BIOS, обеспечивающих самостоятельную инициализацию внешних устройств.

4. Определение размера оперативной памяти и тестирования первого её сегмента: 64 Кбайт.

Обычно, **UEFI** имеет новый графический интерфейс, предполагающий улучшить «реликтовый BIOS».

**MBR** — Master Boot Record — специальная структура загрузочного устройства

**Вектор прерывания** — адрес программы в памяти ОЗУ (обработчика прерывания), которая будет исполняться процессором, когда такое прерывание произойдёт.

**GPT** — GUID Partition Table — новая структура блочных устройств, позволяющая разбивать их на 128 основных разделов, в отличие от структуры MBR, которая допускает наличие только 4-х основных разделов.

**особенности ПО UEFI:**

Может работать в трех режимах:

Защищенный режим.

Открытый режим

Режим совместимости

UEFI также **умеет работать с GPT** разделами.

1. Блочные и символьные устройства компьютера.

Все файлы устройств разделяются на **символьные** и **блочные.**

Имена устройств находятся в специальной **директории /dev**; в нее смонтирована специальная область ядра dev с файловой системой типа **devtmpfs**;

1. Винчестер и загрузочные устройства.

**CHS**

**LBA** - каждый блок, адресуемый на жёстком диске, имеет свой номер

BOIS передать управление загрузочному коду: обычно командой **long jump**.

1. Загрузочный сектор MBR, его назначение и архитектура.

**MBR** (Master Boot Record)

Структура MBR содержит **три основные части**:

* небольшой фрагмент исполняемого кода, - 446 байт;
* таблицу разделов (Partition table), - 64 байт;
* специальную сигнатуру, - 2 байта.

**Первый байт Partition table** содержит признак активности раздела

Следующие **три байта** задают начало раздела в системе координат (C, H, S).

**Пятый байт** обозначает код файловой системы

Следующие **три байта** задают окончание раздела в координатах (C, H, S)

Завершают строку два **четырёхбайтовых числа**, задающие начало раздела и его длину в секторах.

Структура блочного устройства — **GPT**:

а) Каждое блочное устройство может быть разбито на 128 разделов, записи о которых дублируются в конце устройства (отрицательные номера LBA).

б) Все разделы GPT являются основными.

в) Размер LBA=512 байт и LBA0 — для совместимости соответствует MBR.

г) Отдельный LBA содержит записи о 4-х разделах: по 128 байт на раздел.

д) Для идентификации раздела используются GUID.

1. GRUB как универсальный загрузчик ОС.

**GRUB** — официальный загрузчик Linux из проекта GNU

устанавливается на ЭВМ в процессе инсталляции ОС. Это позволяет свободно использовать и Linux и MS Windows на одном компьютере

1. Меню и функции GRUB.

**ПО GRUB** можно рассматривать как маленькую однопользовательскую ОС специального назначения: **интерактивная загрузка различных ОС**

**Среди** основных **функций GRUB** следует выделить:

• поддержку интерпретатора сценариев (близкого к shell);

• умение работать со структурами MBR и GPT блочных устройств;

• поддержка работы со многими современными устройствами ЭВМ;

• распознавание и умение работать со многими современными файловыми системами;

• обнаружение на блочных устройствах ЭВМ наличия различных ОС и автоматическое формирование для них сценария меню загрузки

Основная часть ПО GRUB располагается в разделе блочного устройства и устанавливается в процессе инсталляции на него ОС Linux. По умолчанию, оно помещается в директорию **/boot/grub**, туда же помещается автоматически созданный файл конфигурации **grub.cfg**

**3 Тема 3. Языки управления ОС**

1. Языки программирования и командные интерпретаторы.

Основным языком программирования ядра ОС – **язык С**

Командными интерпретаторами или **shell**

**Каждая строка** языка рассматривается как **команда с аргументами**

Строка, Команда, Аргумент, Конец строки.

**Общая проблематика интерпретаторов** заключается в том, что увеличение функциональных возможностей shell влечёт:

• увеличение размера интерпретатора и уменьшение скорости его загрузки;

• повышенный расход оперативной памяти компьютера.

1. Базовый язык shell (sh).

**Выбор языка sh**:

* стандартизация языка (**POSIX** — набор стандартов)
* современные ядра ОС Linux запускают интерпретатор sh при обнаружении **init**
* интерпретатор **bash**

**метапонятия**:

shell – работает в пользовательском режиме

запустить sh может любой процесс, посредством одного из системных вызовов exec\*();

процесс sh может сам порождать необходимое количество **дочерних процессов (fork())**

Каждая программа пользователя имеет – **один системный ввода и два системных вывода**

Все shell используют свойства базовых категорий, определённых понятиями: **файл, пользователь и процесс**

Все интерпретаторы shell могут использовать текстовые файлы как программы. Такие файлы называют **скрипты или сценарии**.

1. Среда исполнения программ.

**Среда выполнения** подразделяется на:

* структуру файловой системы ОС (/home/asu)
* набор файлов конфигурации (/etc/profile)
* системные переменные среды (системной или пользовательской среды)

Для языка **sh**, среда исполнения определяется условиями видимости той **части файловой системы**, которая соответствует **пользователю**, запустившему shell

Окружение — это набор пар (имя, значение), которые передаётся выполняемой программе

1. Командная строка: опции и аргументы.

**Первое слово** в строке **shell** всегда воспринимается как **команда**, а остальные слова — как **аргументы команды**

Разделение команд – точка с запятой

Перенос строки – обратный слеш

когда shell запускается посредством системного вызова **exec\*(...)** и первым символом нулевого аргумента является -, то сначала читаются и выполняются команды из файлов **/etc/profile и $HOME/.profile.**

**встроенные и внешние команды**

**Опции shell**

1. Переменные shell.

различаются два типа **параметров**: **позиционные и ключевые**

**Позиционные параметры** обозначаются цифрой или одним из символов:

**\*, @, #, ?, -, $, !.**

Значения цифровых позиционных параметров устанавливаются при вызове shell-функций или командой set

**Ключевые параметры** (переменные) обозначаются именами.

завершают предыдущее слово:

; & ( ) | ^ < > пробел табуляция перевод\_строки

Все эти символы могут экранироваться одинарными или двойными кавычками.

• **двойные кавычки** могут экранировать одинарную кавычку;

• **двойные кавычки** не мешают подстановке параметров.

1. Специальные символы и имена файлов.

**Шаблон** — набор символов, который добавляет или изменяет имена файлов, используемые в командах интерпретаторов shell, как аргументы

1. Стандартный ввод/вывод и переадресация.

<слово – файл слово для стандартного вывода (дискриптор 0)

>слово – создает файл (дескриптор файла 1)

>>слово – подстановка в конец файла

Важен порядок переназначения: shell производит переназначение слева направо.

1. Программные каналы.

**Простая команда** — это последовательность слов, разделённых пробелами

Первое слово – **аргумент 0** – имя команды

**Значение простой команды** — это её код завершения

**Каналы** — специальные файлы ОС, создаваемые посредством системного вызова **pipe(...)** и служащие для организации обмена данными (сообщениями) между процессами (программами).

**Конвейер** — это последовательность команд, разделённых знаком |

* Стандартный вывод всех команд, кроме последней, направляется посредством системного вызова pipe(2) на стандартный ввод следующей команды конвейера.
* Каждая команда выполняется как самостоятельный процесс.
* shell ожидает завершения последней команды. Ее код завершения становится кодом завершения конвейера.

**Список** — это последовательность одного или нескольких конвейеров

**Двойные кавычки** разрешают подстановки ключевых и позиционных параметров, **одинарные кавычки** не разрешают подстановки.

Дополнительно, shell использует **обратные кавычки**: shell читает цепочки символов, заключённые в обратные кавычки, и интерпретирует их как команды.

1. Сценарии.

Сценариями (программы)

Управляющих конструкций (for, while, if, case … in)

Кроме управляющих конструкций, sh содержит ряд встроенных (специальных) команд (continue, break).

1. Фоновый и приоритетный режимы.

Задания, выполняющиеся **приоритетном режиме**. Shell блокирует ввод новых цепочек символов до завершения таких заданий.

Если пользователь, перед нажатием клавиши «Ввод» укажет **символ &**, то задание будет выполняться в **фоновом режиме**.

Для просмотра списка запущенных заданий используется команда **jobs**

Для перевода фонового задания в приоритетный режим работы, используется команда **fg** (foreground).

1. Отмена заданий.

Для отмены заданий, выполняющихся в фоновом режиме, используется команда **kill**, которая в качестве аргумента может использовать **номер задания или PID**

1. Прерывания.

Выполнение задания в приоритетном режиме можно прервать, используя комбинацию клавиш Ctrl-Z.

1. Завершение работы ОС.

для выключения компьютера, пользователь должен иметь права на запуск команд:

• halt [OPTION] …

• poweoff [OPTION] …

• reboot [OPTION] …

• shutdown [OPTION] … TIME [MASSAGE]

**4 Тема 4. Управление файловыми системами ОС**

1. Устройства компьютера.

доступ к устройствам ЭВМ возможен только через ядро ОС

все устройства ЭВМ имеют отображение в виде имён в ФС ОС

* **все устройства ОС** делятся на блочные и символьные;
* **блочные устройства ОС** могут содержать файловые системы;
* **символьные устройства ОС** — устройства ОС, не являющиеся блочными;
* **в ядре ОС создаются блочные устройства**, которые не относятся к аппаратным средствам ЭВМ: псевдоустройства ядра ОС или устройства nodev;
* **псевдоустройства ядра ОС** имеют имена, которые почти все совпадают с именами соответсвующих файловых систем; эти имена соответствуют вершинам файловых систем (ФС);
* **псевдоустройство с именем rootfs** является корнем виртуальной файловой системы (VFS — Virtual File System).

1. GRUB загружает ядро ОС
2. передает файл с временной файловой системой
3. инициации внутренних параметров ядра
4. монтирование (подключение) корневой файловой системы
5. ядро ОС создаёт первый процесс, которым является интерпретатор shell
6. shell начинает выполнять сценарий /init

Все псевдоустройства ядра ОС обозначены как **nodev**.

Все ОС UNIX и Linux отображают имена файлов устройств в директорию **/dev** корневой файловой системы

**Node**

**Devtmpf**

**Mknod –** для создания устройств

1. BOOT-сектор и разделы винчестера.

• классическая структура MBR;

• новая структура GPT.

* **главный загрузочный сектор** (MBR), который не зависит от типа ОС;
* **загрузочные секторы** (блоки) логических дисков (разделов), которые зависят от ОС только в плане поддерживаемых ей типов ФС;
* **специальные области разметки и корневой каталог**, зависящие от типа файловой системы;
* **область данных** – файлы и каталоги конкретной файловой системы;
* **цилиндр** для выполнения диагностических операций чтения/записи.\

Фактически, задача загрузки первых реализаций ОС сводилась:

• **к использованию загрузочного блока** для загрузки ядра и передачи ему информации о разделе конкретного блочного устройства;

• **ядро ОС**, проведя инициализацию внутренних параметров и внешних устройств ЭВМ, находила этот раздел и монтировала его **как корневую файловую систему**.

1. Загрузочные сектора разделов.

**Сектор,   
Блок (один, или кратное число секторов)**

**Кластер = Блок**

**LBA (CHS)**

**GUID Partition Table**

В BIOS для загрузки ОС необходимо в разделе корневой файловой системы выделить область для ПО загрузчика, а в программный код первых **446 байт** MBR прописать адрес загрузчика.

1. Структура файловой системы FAT32 (VFAT).

**FAT32 (Microsoft)**

**VFAT -** использовать имена файлов длиной до 255 символов (utf16)

**FAT** – File Allocation Table или Таблица размещения файлов.

32 – число бит, используемое для нумерации кластеров

**Один сектор** диска – 512 байт.

Загрузочный сектор раздела содержит:

 блок параметров диска (BPB)

 загрузочный код

На этапе логического форматирования раздела:

• загрузочный сектор (boot sector);

• таблица размещения файлов (FAT1 и FAT2);

• каталог;

• область данных. **1)** **Загрузочный сектор** – содержит код загрузчика, таблицу BPB и двухбайтовую сигнатуру 55AA

**Резервная загрузочная запись (7-9 сектор)**

**2) Таблица размещения файлов (FAT) –** это массив целых чисел с длинной,

равной количеству кластеров раздела файловой системы.

**3) корневой каталог файловой системы** (Root Directory),

Короткие и длинные имена

**4) Область данных** — все остальные кластеры раздела. Они используются для хранения подкаталогов и файлов.

**Содержимое файлов** — данные некоторой последовательности кластеров.

**Последовательность кластеров** — произвольная и неупорядоченная, но отражена в виде цепочки, в таблице FAT

1. Структура файловой системы EXT2FS.

**ext2** (ext2fs) - Second Extended File System — базовая файловая система ОС Linux.

**Поддерживает** файлы размером — до 16 ГБайт - 2 ТБайт.

**Размер файловой системы** — до 2 - 32 Тбайт. **Максимум** **файлов** — до 1018 штук.

**Структура дискового раздела** ext2fs представляет последовательность группы блоков, которые нумеруются, начиная с 1.

**Каждая группа блоков** состоит из последовательности блоков, которые также нумеруются, начиная с 1

**Размер блока** может быть 1, 2 или 4 килобайта, что определяется в момент форматирования – при создании файловой системы.

1. Супер-блок

**Первые 1024 байта раздела** занимает загрузочная запись раздела.

**Суперблок**, хранится в первой группе блоков и имеет смещение 1024 байта

1. Описание группы блоков

**Отдельный элемент** этого описания имеет структуру длиной **32 байта**.

1. Битовая карта блоков

каждый бит которой показывает, отведён ли соответствующий ему блок какому-либо файлу

1. Битовая карта индексных дескрипторов

какие именно дескрипторы заняты и каким типом файлов.

1. Таблица индексных дескрипторов

имеет размер 128 байт

**дескриптор корневого каталога**.

Каталог файловой системы - файл, состоящий из записей переменной длины

Поле **i\_mode** - тип и права доступа к файлу.

Система адресации данных - **i\_block**

1. Область блоков данных
2. Сравнение файловых систем.

• простоту управления, характерную для FAT32;

• скорость обработки больших файлов, характерную для ext2fs.

* секторов загрузчика ОС;
* суперблока,;
* набора таблиц для использованных блоков данных;
* корневого каталога;
* набора блоков данных.

Преимущества FAT32

• простота реализации

• широкая известность

Недостатки FAT32:

• практическая ограниченность размера ФС и слабая защищённость,

• высокая фрагментированность ФС

**NTFS**

Преимущества ext2:

• высокое быстродействие;

• поддержка файлов до 2 Тбайт

Недостатки ext2:

• отсутствие журналирования, снижающее её надёжность;

• недостаточные, по современным меркам, размеры поддерживаемых разделов ФС и размера файлов

**Ext3 ext4**

1. Стандартизация структуры ФС.

• обязательно наличие корневой ФС ОС;

• обязательно наличие корневого каталога отдельных ФС.

• **dev** — директория размещения узлов устройств ЭВМ (devices);

• **usr** — директория ориентированная на универсальные ресурсы ОС;

• **home** — директория для рабочих (домашних) областей пользователей;

• **bin** — директория расположения исполняемых файлов ОС (binary).

**Первый уровень иерархии ФС**

Управление пользователем root и другими администраторами системы

**Второй уровень иерархии ФС**

содержит данные, которые доступны только для чтения

**Третий уровень иерархии ФС**

подобен второму и продолжается в директории /usr/local.

**корневая директория** изменяемых данных **/var** (game, mail, cache, log, tmp)

С другой стороны, **видимая для пользователя ФС**, создаёт для него **иерархическую систему координат**

• mkdir директория; - создание директории;

• rmdir директория; - удаление директории;

• ls [опции] [шаблон]; - вывод на консоль списка файлов и директорий;

• cat файл; - вывод содержимого файла на консоль

1. Модули и драйверы ОС.
2. интерфейс вызова драйвера ядром ОС
3. интерфейс устройства

Драйвера пишутся в виде модулей

Статические и динамические модули

Модуль предназначен для работы в защищённом режиме пространства ядра ОС

1. Системные вызовы ОС по управлению устройствами и файловыми системами.

POSIX  
1. Основные определения

2. Описание прикладного программного C интерфейса к системным сервисам

3. Описание интерфейса к системным сервисам на уровне командного языка и

служебных программ

4. Детальное разъяснение положений стандарта, обоснование принятых решений.

Системные вызовы

• управление процессами;

• управление файлами;

• управление каталогами и файловыми системами;

• разные, которые нельзя полностью отнести к предыдущим группам.

1. Три концепции работы с устройствами.

**Mknod** – создание ноды в /dev

**manual**, **udev** и **systemd**.

**ФС devfs** и **демона devfsd (**постоянный циклический запуск**)**

**udev**:  
**технология D-Bus -** специальная система межпроцессного взаимодействия

при старте ОС запускается **демон udevd**, который,использует **шину D-Bus**

**systemd** - унификация взаимодействия между процессами режима пользователя.

эффективное отслеживание групп процессов, порождённых системными вызовами fork(...) и объединённых одной прикладной целью

1. Разделы дисков и работа с ними.

**VFS** — Virtual File System

* Суть идеи — расположить VFS между приложениями и конкретными файловыми системами на внешних носителях.

• Это позволяет сконцентрировать в одном месте пользовательский интерфейс для разных типов ФС, что создаёт унифицированный эталон для разработки драйверов ОС.

**В результате, VFS**:

• **упрощает процесс создания драйверов ФС**, что повышает их надёжность;

• **сокращает время разработки**, что создаёт конкурентные преимущества самой ОС;

• **уменьшает время отладки** и исправления ошибок, что повышает актуальность их применения

1. Монтирование и демонтирование устройств.

Монтирование файловой системы — подключение конкретного раздела внешнего блочного устройства к конкретному каталогу ФС, в видимой пользователю общей части дерева корневой ФС.

Демонтирование файловой системы — отключение конкретного раздела

**Mount и umount**

1. Файловые системы loopback, squashfs, overlayfs и fuse.

**Loopback -** циклическое устройство (блочное устройство), ФС

**/dev/loop1, /dev/loop2…**

**Циклические ФС** обычно создаются в файлах, которые могут быть сжатыми и зашифрованными

**Overlayfs** - является вариантом реализации каскадной файловой системы

Каскадная файловая система (**КФС**) — это виртуальная ФС, позволяющая «прозрачно видеть» две изолированные ФС как одну.

**Каскадно-объединенное монтирование** — одновременное монтирование разных нескольких файловых систем как одну

**FUSE —** файловая система в пространстве пользователя

Этот модуль позволяет пользователям без привилегий **создавать свои ФС**

Fusermount – монтирование демонтирование

1. Дисковые квоты.

ext2fs резервирует 5% места для пользователя **root**.

Рекомендуется размешать /home, /opt, /tmp, /var, /usr/local в разных разделах диска

Администратор ОС должен **отслеживать наличие достаточного дискового пространства ФС**. Для этих целей имеется специальное системное ПО под общим названием **quota**

**5 Тема 5. Управление пользователями ОС**

1. Однопользовательский и многопользовательский режимы работы ОС.

**MS-DOS** –

автоматическое монтирование всех ФС

отсутствие концепции пользователя

**первые ОС UNIX –**

Мультипрограммности тоже не было

пути к файлам уже появились

наличие команд chown, getuid и setuid

Современное понятие **однопользовательского режима** означает

* или отключён контроль разграничения прав пользователей
* только суперпользователь root

**Современные ОС** загружаются в два этапа:

• на первом этапе, после загрузки и запуска ядра ОС специальным загрузчиком, например GRUB, ядро распаковывает в оперативную память временную файловую систему и запускает первый процесс init;

• сам процесс init — обычно скрипт, выполняемый интерпретатором shell, устанавливает необходимые модули ОС, ищет и монтирует корневую ФС, создаёт терминальные устройства и запускает на них программы login, удаляет временную файловую систему и завершает работу; все это делается в однопользовательском режиме ОС;

• на втором этапе, пользователи, которые начинают проходить процедуру login, работают уже в многопользовательском режиме ОС.

/etc/passwd /etc/shadow /etc/group /etc/gshadow

1. Разграничение прав пользователей.

все пользователи ОС **работают автономно** и не мешают друг другу, кроме системного администратора.

• каждый пользователь имеет право работать только с теми файлами, директориями и файловыми системами, к которым он имеет доступ;

• пользователь root, с идентификатором UID=0, может делать абсолютно все.

UID и GID

• системные пользователи — UID < 999;

• пользователь live-дистрибутива — UID=999 и GID=999;

• обычные пользователи — UID > 999.

Как правило, обычному пользователю доступны:

• все файлы и каталоги его домашней директории, положение которой задано

системной переменной HOME;

• права записи в каталоги /tmp и /var/tmp;

• права монтирования и демонтирования внешних устройств, которые прописаны в файле /etc/fstab с опцией user

Действительные значения UID и GID (записанные в файлах с паролем)

Эффективные ID пользователя и ID группы – при запуске процессов

Сохранённые ID пользователя и ID группы – сохранные процессом (первые эффективные айди)

1. Login и система доступа Linux-PAM.

**В результате регистрации**:

• имена, идентификаторы и пароли будут записаны в соответствующие файлы: /etc/passwd, /etc/group, /etc/shadow и /etc/gshadow;

• в директории /home будет создана директория с именем пользователя;

• в директорию /home будут перенесены директории и файлы, находящиеся в директории /etc/skel, которые составляют начальный скелет рабочей области любого пользователя.

После загрузки ОС и перехода ее в многопользовательский режим, запускается процесс «**Менеджер сеанса**»

Процедура входа в ОС может быть текстовой или графической

требуется набрать имя и пароль, а возможно и другие сведения, например, домен или язык работы с системой. Это зависит от настроек «Менеджера сеансов»

**процедура login:**

• идентификацию (аутентификацию), подразумевающую совпадение имени и

пароля, зарегистрированных в системе;

• авторизацию, подразумевающую создание среды для работы программ пользователя и фиксирование прав, которыми пользователь обладает

**Модули PAM** — это динамически загружаемые библиотеки, которые находятся в директориях /lib/security или /usr/lib/security.Практическая реализация этой концепции предполагает централизованное использование файлов конфигурации.

1. Команды управления пользователями

Хотя работа с пользователями предполагает всего три действия: **создание, удаление и модификацию**,

**Useradd, userdell, usermod**

**Passwd –** смена пароля

**6 Тема 6. Управление процессами ОС**

1. Подсистема управления процессами.

**Процесс**

**Значения PID**

все процессы находятся **в среде пользователя**

ядро ОС имеет **подсистему управления процессами**

**• взаимодействие процессов;**

**• планировщик;**

**• распределение памяти**

процесс воздействует на подсистему управления процессами **напрямую**, **через API** ядра ОС посредством **системных вызовов** или **косвенно**, через **подсистему управления файлами**;

1. Системные вызовы ОС по управлению процессами.

Максимально, **самостоятельные возможности** процесса можно расширить до **уровня возможностей всех библиотек**, которые он использует.

(**PID**)

подверженность «**жизненному циклу**»

существование в **оперативной памяти ЭВМ**

**планировщик**

**подсистема распределения памяти**

**подсистему взаимодействия процессов**,

Системный вызов **fork**(…) - обеспечивает порождение нового процесса

Системный вызов \_**exit**(…) - обеспечивает завершение процесса

Системный вызов wait(…) - исп. Род. процессом для ожидания завершения доч. процесса.

1. Стандарты POSIX и сигналы.

Развитие способов управления процессами начинается с **сигналов**.

Захват и ожидание задачей ресурсов может привести к **тупикам**

**Сигнал** - механизм, с помощью которого процесс или поток управления уведомляют о некотором событии

Сигналы:

* выявлен аппаратный сбой;
* отработал таймер;
* пользователь ввёл с терминала
* процесс обратился к функции kill(...)

Сигнал процессу может быть послан в следующих случаях:

• либо из командной строки с помощью служебной программы kill;

• либо из процесса с помощью одноимённой функции

1. Подсистема управления оперативной памятью.

ОЗУ — оперативное запоминающее устройство

Процессор взаимодействует с ОП через блок управления памятью (MMU).

Назначение MMU — преобразование для процессора логических адресов ОП в физические и — обратно

Цель MMU — создание для процессора виртуального пространства памяти

Вызвано:

* использования для целей программирования языков высокого уровня
* реализации для ОС мультипрограммного режима работы

**Относительность логической адресации** процессора обеспечивается: наличием регистром сегментов и смещения

У каждого процесса есть адресное пространство, разделённое на 3 логических сегмента:

text segment data segment stack segment

1. Системные вызовы ОС по управлению памятью. Разделяемая память.

**Базовые средства взаимодействия процессов** обеспечиваются:

а) средствами поддержки «жизненного цикла процесса»

б) механизмом сигналов: (слабая информативность сигналов)

**Альтернативный подход к взаимодействию процессов** — использование возможностей файловой системы ОС

Преимущество: возможность использования любых средств взаимодействия.

а) алгоритмическая сложность

**Комбинированные подходы** к взаимодействию процессов:

а) максимальное использование возможностей ядра ОС;

**проект IPC**, который предполагает, что:

а) **все процессы**, желающие взаимодействовать, обращаются к ядру ОС и получают от него уникальный ключ;

б) **используя уникальный ключ**, процессы создают в ядре ОС свои ресурсы и потребляют ресурсы, уже созданные другими процессами.

1. Передача сообщений.

**Очереди сообщений**

1. Главный родительский процесс init.

**процесс создаётся клонированием** из родительского процесса

**процесс наследует** все права пользователя от родителя

**процесс управляется** **родителем**

**процессу разрешается** **модифицировать себя**

**процессу разрешается создавать дочерние процессы**

**Мультипрограммный режим -** параллельное выполнение множества процессов на одном компьютере

**Мультизадачный режим** - в пространстве пользователя организуется параллельное выполнение вычислений, посредством одного или нескольких процессов, с использованием специальных механизмов взаимодействия: **синхросигналы, семафоры и передачу данных**

Легковесные процессы (**thin**) - потоки выполнения, нити или треды (threads).

**Зомби**

Главный родительский процесс **— процесс init**

**Классический подход** создания процесса init:

1. **загрузка ядра ОС**
2. **передача управления** от загрузчика ядру
3. **работа ядра в активном режиме** (инициализирует свои параметры и основные устройства ЭВМ, находит и монтирует корневую фс, создаёт среду для пользовательского режима, находит в корневой ФС исполняемый файл init и запускает его)
4. **переключение ядра в пассивный режим** (передача управления подсистеме управления процессами)
5. **работа ядра в пассивном режиме**

**Современный подход:**

2. передача управления дополняется передачей ядру массива данных, содержащих временную ФС

3. распаковывает временную файловую систему и выбирает вариант создания первого процесса: если в корне ФС имеется сценарий init, то создаются структуры для файла /bin/sh, иначе создаётся среда для файла /sbin/init;

1. Четыре подхода к управлению процессами: монопольный режим, System V, upstart и systemd.

**Монопольный подход** к управлению процессами применяется:

**во встроенных системах**

**в монопольном режиме работы** администратора ОС

**при загрузке** **современных ОС**

Например, ОС ArchLinux, используя универсальный загрузчик GRUB, указывает в файле grub.cfg местоположение ядра ОС (vmlinuz) и файла initrd — сжатой временной ФС.

Подход UNIX **System V** ориентирован на универсальное применение

ядро и ОС в целом могут работать на различных **уровнях (runlevel)**

Существенными недостатками подхода System V являются:

1) строго последовательная организация

2) слабые возможности по отслеживанию групп процессов, решающих общую задачу, что приводит к потере контроля над отдельными процессами

**Upstart:**

Во время загрузки ОС, используется новая программа /sbin/init, которая:

1) **запускает сразу все приложения**, для которых имеются файлы конфигурации;

2) если приложение не может запуститься по причине отсутствия некоторых ресурсов, оно переводится с помощью сигналов в режим ожидания предоставления таких ресурсов;

3) сигналы старта и останова приложений принимаются init через шину DBus

**Job (**общее назначение ПО**) task (**запуск или завершение**) service (**аналог демона**)**

**upstart** снимает проблему следования строгому порядку запуска всех процессов

**Systemd** — демон инициализации, призванный унифицировать управление устройствами и процессами

1) **сокет-активные и шино-активные сервисы**, что часто приводит к лучшему распараллеливанию взаимозависимых сервисов;

2) **сервисные процессы cgroups**, использующие специальные идентификаторы групп

1. Порождение и завершение процессов, просмотр их состояния и изменение приоритета.

сигналы имеют **целочисленные номера**

**семантика сигнала** должна быть понятна

Для посылки сигналов процессам, используется утилита **kill**

Пользователь может изменять приоритет выполняемых процессов используя уровни **тактичности**.

• -20 — максимальный приоритет;

• +19 — минимальный приоритет

1. Состояния процессов в ядре ОС.

**Планировщик** — часть ПО ядра ОС, который организует некоторую очередь процессов и переключает их в различные состояния, обеспечивая мультипрограммный режим их функционирования.

**Выполнение** - активное состояние процесса

**Ожидание** - пассивное состояние процесса, процесс заблокирован по своим внутренним причинам

**Готовность** - также пассивное состояние процесса, но в этом случае процесс заблокирован в связи с внешними обстоятельствами

**Контекст процесса -** данные о состоянии операционной среды

**Дескриптор процесс –** информация о состоянии процесса

**Очередь процессов -** дескрипторы отдельных процессов, объединённые в Списки

**Создание и завершение процесса**

Целенаправленное обеспечение «жизненного цикла процессов» называется стратегией диспетчеризации

1. ОС реального времени.

**Системы пакетной обработки данных** – старый способ

**Интерактивные системы** - непосредственное взаимодействие пользователя и ЭВМ

**Системы реального времени -** допускают интерактивное взаимодействие с user, но ориентированы на автоматическое управление различными техническими системами

Основные требования к задачам реального времени:

1) **окончание работы к сроку** и **исключение потери данных**;

2) **предсказуемость**

1. Алгоритм разделения времени.

**алгоритмы планирования**

1) определение момента времени для смены выполняемого процесса;

2) выбор процесса на выполнение из очереди готовых процессов;

3) переключение контекстов "старого" и "нового" процессов

**на квантовании** и **на приоритетах**

**На квантовании** смена при исчерпании кванта процессорного времени

очередь готовых процессов - **FIFO** или **LIFO**

**На приоритетах**

Приоритет может назначаться:

1) **директивно**,администратором системы

2) **вычисляться самой ОС** по определенным правилам;

3) оставаться фиксированным – **статические приоритеты**

4) изменяться во времени – динамические приоритеты.

**относительные** приоритеты и **абсолютные** приоритеты

Существует два типа процедур планирования:

1) **невытесняющая** многозадачность

2) **вытесняющая** многозадачность