**Тема 1**

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

Система обработки данных (СОД) – это совокупность технических средств и программного обеспечения, предназначенная для информационного обслуживания пользователей и технических объектов.

Система обработки данных разделена на два больших класса:

1. Сосредоточенные системы – в них обработка данных ведется отдельной ЭВМ, вычислительным комплексом или вычислительной системой.
2. Распределенные системы – в них обработка данных ведется системами телеобработки или вычислительными сетями.

Каждый элемент такой классификации имеет свою ОС.

ОС – это программное обеспечение (ПО), которое устанавливается на аппаратную часть вычислительной техники. С этой точки зрения у ЭВМ есть:

1. Техническая часть – аппаратное обеспечение компьютера.

Основой архитектуры аппаратной части ЭВМ составляет системный блок, в котором размещены: микропроцессор, ОЗУ (блок оперативного запоминающего устройства), ПЗУ (микросхемы постоянного запоминающего устройства, винчестер и устройства для запуска компакт-дисков и дискет.

1. Программная часть – программное обеспечение компьютера.

Условно разделяется на 3 части:

* Системное ПО – ОС и программы общего пользования. К системному ПО относятся: операционные системы, программы-оболочки, операционные оболочки, драйверы и утилиты.
* Прикладное ПО – программы, обеспечивающие выполнение необходимых работ на ЭВМ.
* Инструментальное ПО – программы, обеспечивающие разработку новых программ для компьютера на различных языках программирования.
* **Серверные ОС и рабочие станции.**

В контексте аппаратного обеспечения ЭВМ:

* Сервер – это специальный компьютер или оборудование для выполнения сервисного ПО. ЭВМ предназначенное для длительной непрерывной работы без выключения или перезагрузки ОС.
* Рабочая станция – это ЭВМ, оборудование которой расширено устройствами мультимедиа и другими системами.

В контексте программного обеспечения ЭВМ:

* Сервер – это любая запущенная программа, ориентированная на обслуживание запросов других программ – клиентов.
* Рабочая станция – это ЭВМ, предназначенная для интерактивной работы с пользователем, на которой установлено клиентское ПО.

Итоговые определения:

* Сервер – дистрибутив ОС или ЭВМ, с установленным системным и прикладным ПО, ориентированная на выполнение функций сервера.
* Рабочая станция (desktop) – это дистрибутив ОС или ЭВМ, предназначенная для интерактивной работы с пользователем, на которой установлено клиентское прикладное ПО.
* **ОС как виртуальная машина.**

При разработке ОС широко применяется абстрагирование, которое является важным методом упрощения и позволяет сконцентрироваться на взаимодействии высокоуровневых компонентов системы, игнорируя детали их реализации. В этом смысле ОС представляет собой интерфейс между пользователем и компьютером.

Архитектура большинства компьютеров на уровне машинных команд очень неудобна для использования прикладными программами. Например, работа с диском предполагает знание внутреннего устройства его электронного компонента – контроллера для ввода команд вращения диска, поиска и форматирования дорожек, чтения и записи секторов и т. д. Ясно, что средний программист не в состоянии учитывать все особенности работы оборудования (в современной терминологии – заниматься разработкой драйверов устройств), а должен иметь простую высокоуровневую абстракцию, скажем, представляя информационное пространство диска как набор файлов. Файл можно открывать для чтения или записи, использовать для получения или сброса информации, а потом закрывать. Это концептуально проще, чем заботиться о деталях перемещения головок дисков или организации работы мотора. Аналогичным образом, с помощью простых и ясных абстракций, скрываются от программиста все ненужные подробности организации прерываний, работы таймера, управления памятью и т. д. Всем этим занимается операционная система. Таким образом, операционная система представляется пользователю виртуальной машиной, с которой проще иметь дело, чем непосредственно с оборудованием компьютера.

Первоначально ОС были монолитными и не имели архитектуры.

Классическая архитектура ОС основана на концепции иерархической многоуровневой машины, привилегированном ядре, пользовательском режиме работы транзитных модулей.

Модули ядра выполняют базовые функции ОС: управление процессами и памятью, устройствами ввода-вывода и тому подобное.

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

Каждое ядро ОС абстрагирует три базовые концепции: файл, пользователь и процесс.

* **Многослойная структура ОС.**

Первоначально ОС были монолитными и не имели архитектуры.

Классическая архитектура ОС основана на концепции иерархической многоуровневой машины, привилегированном ядре, пользовательском режиме работы транзитных модулей.

Модули ядра выполняют базовые функции ОС: управление процессами и памятью, устройствами ввода-вывода и тому подобное.

В концепции многоуровневой иерархической машины структура ОС представляется рядом слоев:

Всего 7 слоев:

1. Аппаратура
2. Средства аппаратной поддержки ОС ядра
3. Машинно-зависимые модули ядра ОС
4. Базовые механизмы ядра
5. Менеджеры ресурсов, файловая система, виртуальная память
6. Интерфейс системных вызовов API
7. Утилиты, системные программы, приложения пользователей

Повышение устойчивости работы ОС обеспечивается тем, что ядро ОС работает в привилегированном режиме – особый режим работы процессора, в котором никакая программа, в другом режиме не влияет на работу процессора.

* **ОС как базовая часть ПО ЭВМ.**

Ядро ОС составляет сердцевину системного ПО, без которого ПО не может выполнять ни одну из своих функций. В ядре решаются внутрисистемные задачи организации вычислительного процесса, недоступные для приложений. Таким образом, ядро ОС является базовым ПО ЭВМ.

Особый класс функции ядра служит для поддержки приложений, создавая для них так называемую прикладную программную среду. Все приложения обращаются к ядру со специальными запросами – системными вызовами. Например, открытие и чтение файла, получение системного времени и другие.

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

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

Каждое ядро ОС абстрагирует три базовые концепции: файл, пользователь и процесс.

* **Режимы ядра и пользователя.**

Ядро работает в специальном привилегированном (защищенном) режиме. Соответственно, режим, в котором работают утилиты и остальное прикладное ПО ОС называется режимом пользователя.

Классическая архитектура ПО ОС содержит монолитное ядро ОС, которое содержит ПО всех уровней, кроме последнего.

Альтернативой классическому варианту архитектуры ОС является микроядерная архитектура ОС, в ней в привилегированном режиме работает только микроядро, в состав которого входят машинно-зависимые модули и модули, влияющие на базовые механизмы обычного ядра. Все остальные функции ядра работают в пользовательском режиме.

Выполнение системного вызова для архитектуры с микроядром сопровождается четырьмя переключениями режимов, а в классической архитектура двумя.

* **Ядро и модули ОС.**

Модули ядра выполняют базовые функции ОС: управление процессами и памятью, устройствами ввода-вывода и тому подобное.

Взаимодействие ядра ОС с аппаратной частью ЭВМ осуществляется через драйверы.

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

Для работы с модулями имеются специальные утилиты, такие как:

* Lsmod – просмотр списка модулей.
* Insmod – инсталляция модулей.
* Rmmod – удаление модуля.
* Modprobe – может выполнять две функции, описанные выше.
* Modinfo – получение информации о модуле.
* **Три базовых концепции ОС: файл, пользователь, процесс.**

ОС UNIX прямо декларируют парадигму «Всё есть файл»

К любому файлу применимы операции чтения, записи и запуска.

*Файловая система* – это поименованная совокупность обычных и специальных файлов.

Обычный *файл* – это именованная упорядоченная последовательность байт.

Специальный *файл* имеет имя и специализацию по назначению.

Все операции с файлами интерпретируются через концепцию пользователя. А понятие *пользователь* интерпретируется через свои элементы:

* Владелец – именованный и индексированный объект ОС, имеющий определенные права на операции с файлами. Имеет индекс владельца или UID. Чем меньше значение, тем более важным является пользователь. Например, пользователь root имеет UID = 0.
* Группа – именованный и индексированный объект ОС, который объединяет владельцев ОС. Имеет индекс группы или GID аналогичен UID и выполняет те же функции для групп.
* Другие – пользователи без имени и идентификатора.

Третья концепция – *процесс* – интерпретируется как запущенная программа или задача.

Процесс – это элементарный управляемый объект ОС, имеющий целочисленный идентификатор PID. Значения PID начинаются с 1 (обычно это процесс – init) – главный родительский процесс, и увеличиваются по мере порождения дочерних процессов.

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

Ядро ОС запускает только один процесс init. Все остальные процессы режима пользователя являются дочерними относительно процесса init.

Запуск любой программы в режиме пользователя осуществляется с помощью двух системных вызовов:

1. Fork(…) – полностью дублирует существующий процесс, вместе со всеми открытыми файлами, порождая новый (дочерний) процесс с новым PID. Программист различает родительский и дочерний процесс только по целочисленному значению, которое возвращает функция fork. (-1 – ошибка, дочерний процесс не создан, 0 – дочерний процесс, >0 – родительский процесс, которому передано значение PID дочернего процесса.
2. Exec(….) используется если дочерний процесс создан для запуска некоторой программы. Обратите внимание, что родительский процесс продолжает контролировать работу уже новой программы и имеет право принудительно завершить ее работу.

* **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 учет многих фактор:

• производитель дистрибутива;

• тип процессора, на который рассчитан дистрибутив;

• лицензия дистрибутива и ценовая политика дистрибьютора;

• поддержка национальных языков;

• типы носителей, на которых распространяется дистрибутив;

• особенности инсталляции;

• сопровождение дистрибутива;

• наличие документации.

**Тема 2**

* **Архитектура x86. BIOS и его функции.**

**х86** – архитектура процессора с одноименным набором команд, которая впервые была реализована в процессорах компании Intel и относится к серии её процессоров ранних моделей – 80086, 80386, 80486.

* Для выполнения арифметических и логических операций служат регистры общего назначения: AX, BX, CX, DX.
* Индексные регистры и указательные регистры служат для формирования массивов.
* Регистр состояния содержит биты, которые изменяются в процессе различных операций.
* Сегментные регистры являются указателями на начало областей оперативной памяти
* Указатель команды – смещение относительно начала сегмента команд, определяемого регистром CS.

Преемником компании Intel, для дешевых процессоров x86, стала компания AMD. Со временем, появились 32-битные и 64-битные процессора.

Начиная с процессора 80386, появился полноценный защищенный режим работы, который стал обеспечивать ядрам ОС привилегированный режим работы.

Среди всего множества команд, которые типично работают на всех процессорах и используют сегмент стека, выделяются две:

1. Вызов функции или команда CALL адрес.
2. Прерывание или команда INTERRUPT номер.

Другим важным аспектом работы аппаратного обеспечения ЭВМ является способ выполнения операций ввода-вывода. Существует три таких способа:

1. Программный ввод – вывод
2. Ввод-вывод, управляемый прерыванием
3. Прямой доступ к памяти (DMA).

Подсистема ввода-вывода должна также учитывать режим работы шины, который может быть пословным или поблочном.

**BIOS** (базовая система вводы-вывода) – часть системного программного обеспечения ЭВМ, реализованная в виде микропрограмм, и обеспечивающая доступ для ОС к материнской плате компьютера.

Есть два типа микросхем:

1. Микросхемы EPROM – содержимое этих микросхем может быть стерто при помощи ультрафиолетового излучения специальным прибором.
2. Микросхемы EPPROM – содержимое этих микросхем может быть стерто при помощи электрического сигнала, при этом вынимать микросхему не обязательно.

Обычному конечному пользователю в BIOS желательно использовать только две возможности:

• установка приоритетов загрузочных устройств;

• установка адресов дополнительных плат расширения компьютера.

Для сохранения настроек BIOS, а также параметров конфигурации компьютера используется микросхема CMOS-памяти. CMOS – технология построения электронных схем.

Чтобы зайти в BIOS необходимо нажать на определенные клавиши или группы клавиш, после перезагрузки компьютера. Такие как Del, F2,Esc.

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

**POST –** самостоятельное тестирование аппаратного обеспечения компьютера после включения.

Существует полный и сокращенный тест:

Сокращенный тест, включает в себя 4 этапа:

1. Проверка целостности программ BIOS в ПЗУ, используя контрольную сумму.
2. Обнаружение и инициализацию основных контроллеров, системных шин и подключенных устройств.
3. Выполнение программ BIOS, обеспечивающих самостоятельную инициализацию внешних устройств.
4. Определение размера оперативной памяти и тестирования первого ее сегмента: 64 Кбайт.

Полный тест включает в себя большее количество этапов, таких как:

1. Проверка регистров процессора.
2. Проверка контрольной суммы ПЗУ.
3. Проверка системного таймера и порта звуковой сигнализации;

И другие.

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

ПО BIOS выполнив все программы POST, BIOS ищет загрузочное устройство ЭВМ, среди списка доступных, после чего запускает загрузочный код, расположенный в специальном секторе блочного устройства: MBR. MBR — Master Boot Record — специальная структура загрузочного устройства.

ПО UEFI сразу переводит процессор в защищенный режим работы, тем самым, обеспечивая повышенную надежность работы ПО. Кроме того, ПО UEFI способно работать с файловыми системами и более современной структурой блочных устройств — GPT. 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.

У ПО UEFI есть определенные особенности:

1. Работа в защищенном режиме процессора.
2. Возможность защищенной сертификатами загрузки ОС.
3. И другие.

* **Блочные и символьные устройства компьютера.**

Все файлы устройств разделяются на символьные и блочные:

1. Блочными называются внешние устройства компьютера, обмен данными с которыми можно производить только блоками: целостной упорядоченной последовательностью байт; к блочным устройствам относятся «жесткие» диски, магнитные ленты, диски CDROM и другие; на блочных устройствах возможно создание файловых систем;
2. все другие устройства, не являющиеся блочными, называются символьными; обмен данными с символьными устройствами осуществляется по одному байту; например, клавиатура, мышь, консоль экрана, COM-порты, сетевые устройства и другие — символьные.

* **Винчестер и загрузочные устройства.**

Загрузочным устройством ЭВМ является винчестер или «жесткий диск».

Винчестер состоит из набора круглых пластин, которые центральной частью, на некотором расстоянии, надеты на шпиндель, вращающийся посредством электродвигателя. Физические блоки винчестера (сегменты) пронумерованы в системе координат CHS, начиная с сегмента (0, 0, 1).

Адресация CHS не позволяет адресовать более 7,8 Гбайт, поэтому винчестера емкостью более 7,8 Гбайт используют адресацию LBA.

LBA (Logical block addressing) — механизм адресации и доступа к блоку данных на «жестком диске», при котором системному контроллеру нет необходимости учитывать геометрию самого жесткого диска: количество цилиндров, сторон и секторов на цилиндре. Контроллеры современных IDE-дисков в качестве основного режима трансляции адреса используют LBA.

Суть LBA состоит в том, что каждый блок, адресуемый на жестком диске, имеет свой номер - целое число, начиная с нуля и далее: LBA 0 = Цилиндр 0/Головка 0/Сектор 1

* **Загрузочный сектор MBR, его назначение и архитектура.**

MBR (Master Boot Record) — это главная загрузочная запись блочного устройства. Более точно — это код и данные расположенные в первом секторе блочного устройства, которые могут быть использованы для загрузки некоторой ОС.

Структура MBR содержит три основные части:

• небольшой фрагмент исполняемого кода, - 446 байт;

• таблицу разделов.

• специальную сигнатуру.

Согласно «традиции MBR», винчестер может быть разбит на четыре основных раздела. Допускается один из разделов использовать как расширенный раздел и делить его дополнительно.

* **GRUB как универсальный загрузчик ОС.**

Наиболее распространенными загрузчиками ОС являются:

* Windows Boot Manager (bootmgr.exe, winload.exe)
* LILO (LInux LOader) - старый загрузчик ядра Linux;
* GRUB (Grand Unified Bootloader) — новый загрузчик ядра Linux и Hurd
* И другие.

GRUB — официальный загрузчик Linux из проекта GNU. Он может загружать разные ОС, включая MS Windows, с многих разных аппаратных платформ. GRUB (точнее GRUB2) входит и в дистрибутив Arch Linux. Он устанавливается на ЭВМ в процессе инсталляции ОС. Это позволяет свободно использовать и Linux и MS Windows на одном компьютере.

* **Меню и функции GRUB.**

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

Среди основных функций GRUB следует выделить:

* поддержка интерпретатора сценариев, близких по функциональным возможностям языка shell, который собственно говоря и был его прототипом;
* умение работать со структурами MBR и GPT блочных устройств;
* поддержка работы со многими современными устройствами ЭВМ;
* распознавание и умение работать с многими современными файловыми системами;
* поддержка национальных языков и других мультимедийных средств ЭВМ;
* обнаружение на блочных устройствах ЭВМ наличия различных ОС и автоматическое формирование для них сценария меню загрузки.

Хотя не все функции GRUB работают одинаково эффективно, потенциал этого ПО является очевидным.

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

Одним из вариантов использования ПО GRUB является установка его на flashUSB. Такой подход позволяет избежать многих проблем, связанным с недостаточной квалификацией исполнителей.

**Тема 3**

* **Языки программирования и командные интерпретаторы.**

Основным языком программирования ядра ОС и другого системного ПО является язык С. Этот язык специально создавался для написания ОС и постепенно вытеснил языки Ассемблера.

Язык С мало пригоден для создания масштабных приложений. Для этих целей используются языки объектно-ориентированного программирования (ООП), такие как С++, C#, Java и другие.

Целевое назначение ОС — создание виртуальной машины или среды исполнения для работы системного, прикладного и инструментального ПО компьютера. Важнейшая функция такой виртуальной машины — управление программным обеспечением ЭВМ, работающим в режиме пользователя.

Работа любого командного языка опирается на специальное устройство ЭВМ, которое называется терминалом. Терминал – это последовательное устройство ЭВМ, обеспечивающее ввод с клавиатуры потока символов, обрабатывающее эти символы, а затем выводящее результат обработки на экран.

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

• увеличение размера интерпретирующей программы и уменьшение скорости ее загрузки;

• повышенный расход оперативной памяти компьютера.

* **Базовый язык Bourne shell (sh).**

Shell применяется в двух аспектах:

• как расширительное обозначение всех командных интерпретаторов ОС;

• как конкретизация интерпретатора sh (Bourne Shell).

POSIX — набор стандартов, описывающих интерфейсы между операционной системой и прикладной программой.

Определим ряд метапонятий, которые shell учитывает в своей работе:

• shell — это программа (утилита или командный интерпретатор) sh, обычно /bin/sh, который работает в среде ОС: в пользовательском режиме;

• запустить sh может любой процесс, посредством одного из системных вызовов exec\*(); при этом, sh будет использовать среду ОС, в которой работала вызывающая программа;

• процесс sh может сам порождать необходимое количество дочерних процессов, посредством системного вызова fork(), отслеживая их работу и анализируя их коды завершения;

• нулевой целочисленный код завершения означает нормальное выполнение ко- манды дочерним процессом;

• ненулевой целочисленный код завершения означает ошибочное выполнение команды дочерним процессом и дополнительно интерпретируется, в зависимости от ситуации и режимов работы sh.

Каждая программа пользователя, запущенная в виде процесса на компьютере, имеет:

• один системный ввод;

• два системных вывода.

На уровне файловых дескрипторов shell, говорят об устройствах:

• устройство 0 — устройство ввода;

• устройство 1 — устройство нормального вывода программы;

• устройство 2 — устройство вывода ошибок.

На уровне языка С, мы имеем стандартные устройства:

• stdin — устройство ввода;

• stdout — устройство нормального вывода программы;

• stderr — устройство вывода ошибок.

Все shell используют свойства базовых категорий, определенных понятиями: файл, пользователь и процесс.

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

* **Среда исполнения программ.**

Среда выполнения любой программы ОС подразделяется на:

• структуру файловой системы ОС, которую программа использует для ввода и вывода данных;

Для языка sh, среда исполнения определяется условиями видимости той части файловой системы, которая соответствует пользователю, запустившему shell. Как правило, для этих целей используется директория /home. Например, пользователь asu имеет домашнюю директорию /home/asu.

• набор файлов конфигурации, которые определяют параметры данных программы или дополнительные данные конфигурации среды исполнения;

• системные переменные среды, которые наследуются как из среды родительского процесса, а также создаются или удаляются в процессе работы программы.

Все интерпретаторы shell используют переменные среды, которые подразделяются:

• переменные системной среды — для системных процессов;

• переменные пользовательской среды — отдельно для каждого пользователя, вошедшего в систему.

Примеры переменных: HOME - Определяет домашний каталог пользователя. Подразумеваемый аргумент команды cd (1) - основной каталог.

PS1 – основное приглашение (по умолчанию "$ ").

PS2 - Вспомогательное приглашение (по умолчанию "> ").

* **Командная строка: опции и аргументы.**

Каждая строка символов, которая заканчивается символом «Enter», рассматривается терминалом как целостный набор данных, который необходимо передать интерпретатору shell.

Сам интерпретатор анализирует полученную строку, проверяя ее синтаксис и разделяя ее на последовательность отдельных команд и их аргументов.

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

Все команды shell, условно, разделяются на две группы:

• встроенные команды — команды, которые интерпретатор выполняет самостоятельно;

• внешние команды — это программы и утилиты, которые shell ищет в файловой системе и, после проверки прав доступа, пытается запустить, используя системные вызовы fork(...) и exec\*(...).

В общем случае, интерпретатор shell может рассматриваться как команда с аргументами.

Опции интерпретируются shell при его запуске следующим образом:

• если не указаны опции -s или -c, то первый аргумент рассматривается как имя файла, содержащего команды;

• остальные аргументы передаются этому командному файлу как позиционные параметры.

* **Переменные командного языка shell.**

Все переменные Shell называются параметрами. Следует учесть, что есть два вида параметров: позиционные и ключевые.

Знак $ используется для подстановки значений параметра.

Позиционные параметры обозначаются цифрой или одним из символов: \*, @, #, ?, -, $, !.

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

• 0 — параметр 0 — имя вызываемой функции;

• 1 — параметр 1 — аргумент 1;

• 2 — параметр 2 — аргумент 2 и далее.

Ключевые параметры (переменные) обозначаются именами.

* **Специальные символы и имена файлов.**

В командах, работающих с именами файлов, возможно использование шаблонов. Шаблон — набор символов, который добавляет или изменяет имена файлов, используемые в командах интерпретаторов shell, как аргументы.

Примеры использования шаблонов:

• ls .. - вывод списка файлов родительского каталога;

• ls . - вывод списка файлов текущего каталога (каталог, в котором находится пользователь);

• ls .config — вывод списка имен каталога .config;

* **Стандартный ввод/вывод и переадресация.**

Типичный процесс ввода-вывода читает данные с клавиатуры (дескриптор файла 0) и выводит данные на экран терминала (дескрипторы файлов 1 и 2).

В случае, когда для чтения и записи данных используются другие источники информации, применяются следующие правила перенаправления (переадресации) ввода и вывода:

(<слово) Использовать файл слово для стандартного ввода (дескриптор файла 0).

(>слово) Использовать файл слово для стандартного вывода (дескриптор файла 1). Если файла нет, он создается; если есть, он опустошается.

(>>слово) Использовать файл слово для стандартного вывода. Если файл существует, то выводимая информация добавляется в конец, то есть, сначала производится поиск конца файла; в противном случае файл создается.

* **Программные каналы.**

Общие конструкции языка shell используют специальные файлы ОС, которые называются каналами (неименованными каналами).

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

В языке sh, для организации программных каналов между простыми командами, используется понятие конвейер.

Конвейер - это последовательность команд, разделенных знаком | (вертикальная черта).

Список - это последовательность одного или нескольких конвейеров, разделенных символами;, &, && или || и, быть может, заканчивающаяся символом ; или &.

* **Сценарии.**

Простую команду или простой конвейер можно набрать и выполнить в окне консоли (терминала). Сложные конструкции языка sh — программы — пишутся в файлах, которые называются сценариями.

Сценарий — последовательность простых команд и конвейеров, оформленных с помощью управляющих конструкций.

Кроме управляющих конструкций, sh содержит ряд встроенных (специальных) команд. Например, cd [каталог] Сделать текущим заданный каталог. Если каталог не указан, используется значение переменной HOME.

echo [аргумент ...] Выдать аргументы на стандартный вывод, разделяя их пробелами.

* **Фоновый и приоритетный режимы.**

В интерактивном режиме, shell взаимодействует с конкретным пользователем посредством консоли (терминала):

• пользователь в консоли набирает (редактирует) цепочку символов и, в конце цепочки нажимает клавишу «Ввод»;

• shell проводит синтаксический анализ введенной цепочки, выделяет простые команды, формирует конвейер команд и запускает задание;

• когда задание, которое может состоять из множества процессов, завершится, shell выдаст на консоль приглашение на ввод новой цепочки символов.

Задания, выполняющиеся указанным способом, называются заданиями, выполняющимися в 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. В этом случае:

• shell выводит на консоль номер задания, заключенный в квадратные скобки, и номер PID родительского процесса задания;

• после этого, shell выводит на консоль приглашение пользователю для ввода новой цепочки символов.

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

* **Отмена заданий.**

Для отмены заданий, выполняющихся в фоновом режиме, используется команда kill, которая в качестве аргумента может использовать номер задания или PID. В результате применения этой команды, задание прекращает работу, а созданные им процессы уничтожаются.

* **Прерывания.**

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

При этом:

• выполнение задания приостанавливается и shell выдает пользователю приглашение на ввод новой цепочки символов;

• командой fg (foreground) задание можно перевести в приоритетный режим;

• командой bg (background) задание можно перевести в фоновый режим.

* **Завершение работы ОС.**

Если запустить ОС может любой пользователь, который включит питание ЭВМ и, возможно, выберет в меню тип загружаемой ОС, то для выключения компьютера, пользователь должен иметь права на запуск определенных команд.

Работая в графической оболочке, пользователь для выключения ЭВМ использует соответствующее меню. В этом случае, команды и сам процесс выключения ОС — скрыт от пользователя.

**Тема 4**

* **Устройства компьютера.**

Термин устройства компьютера ассоциируются с аппаратурой ЭВМ.

• доступ к устройствам ЭВМ возможен только через ядро ОС;

• все устройства ЭВМ имеют отображение в виде имен в ФС ОС.

С другой стороны, устройства компьютера являются ресурсами ОС, которые управляются ядром ОС.

С третьей стороны, сами файловые системы ОС располагаются на устройствах компьютера.

• 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 соответствующих файловых систем; эти имена соответствуют вершинам файловых систем (ФС);

• псевдоустройство с именем rootfs является корнем виртуальной файловой системы (VFS — Virtual File System).

Общая семантика слова устройство соответствует семантике английского слова device. Реальные файловые системы ОС, на которых долговременно хранится информация, находятся на физических устройствах ЭВМ. Поскольку физические устройства типа винчестера разбиваются на разделы, то реальные ФС создаются в разделах блочных устройств. Именно разделы блочных устройств монтируются к директориям VFS.

В терминологии ядра, имя устройства, которое отображается в директории /dev, называется узел (node).

NODE - узел - специальная именованная структура, отображаемая в директории /dev, создаваемая для связи ядра ОС с физическим устройством ЭВМ.

Узлы создаются как для блочных, так и для символьных устройств. Каждый узел имеет имя, тип, старший номер и младший номер.

* **BOOT-сектор и разделы винчестера.**

Винчестер или «жесткий диск», который является основным блочным устройством для долговременного хранения информации и одновременно - оперативно используется в работе ОС. Кроме того, именно на одном из разделов винчестера размещается файловая система, рассматриваемая ядром ОС как корневая ФС.

Все блочные устройства, на которых размещаются конкретные ФС имеют свою структуру, определяемую физическим типом устройства. Такую структуру имеют и физические устройства, называемые винчестер.

Ограничиваясь только классической структурой MBR, можно выделить:

• 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.

Для примера, в ОС УПК АСУ, такими утилитами являются:

• fdisk - позволяет показать все блочные устройства ЭВМ, а также обеспечить разбиение устройства на нужное количество разделов;

• mkfs — позволяет создать файловую систему на заданном разделе заданного блочного устройства;

• mknod — позволяет создавать узлы в ФС, которые именуют блочные и символьные устройства для доступа к ним ПО пользовательского режима;

• mount — позволяет монтировать (подключить) конкретную ФС к древовидной структуре файлов ядра ОС;

• umount — позволяет размонтировать (отключить) конкретную ФС от древовидной структуры файлов ядра ОС.

Следуя основной парадигме UNIX-систем: «Все есть файл», разработчики ОС создавали свои файловые системы, которые монтировались при загрузке ОС и использовались, в дальнейшем, как уникальные хранилища информации.

* **Загрузочные сектора разделов.**

Для блочного устройства типа винчестер, минимальным физически читаемым объемом информации (данных) является сектор, который для современного оборудования ЭВМ равен 512 байт.

Понятие сектора связано физической организацией обмена данными в блочных устройствах. Файловые системы (ФС), которые стали создаваться для различных нужд ОС, стали использовать понятие блока.

Понятие блока связано с логической организацией обмена данными между ОС и блочными устройствами. Соответственно размер блока может быть равен одному сектору или кратен целому числу таких секторов.

Кластер — это последовательный набор секторов винчестера, который на логическом уровне рассматривается как единица обмена данными между ОС и блочным устройством.

Альтернативный подход предполагает смещение начала первого раздела относительно сектора MBR. Такое смещение позволяет записать стразу же после сектора MBR ПО загрузчика, что и используется нами при установке ПО GRUB.

* **Структура файловой системы FAT32 (VFAT).**

FAT32 - файловая система компании Microsoft. Она является преемницей файловых систем FAT8, FAT12 и FAT16.

VFAT — расширение FAT, дающее возможность использовать имена файлов длиной до 255 символов.

Фактически, VFAT и используется во всех ОС, как универсальный хранитель информации.

FAT – File Allocation Table или Таблица размещения файлов.

32 – число бит, используемое для нумерации кластеров (блоков) раздела файловой системы (на самом деле используется 28 бит).

Кластер – это объем данных, которыми оперирует файловая система. Обычно, для 8 Гбайт раздела используется 4 КБайтные кластеры (8 секторов диска).

Один сектор диска – 512 байт.

Загрузочный сектор раздела содержит:

• блок параметров диска (BPB), в котором содержится информация о разделе: размер и количество секторов, размер кластера, метка тома и другие;

• загрузочный код – программу, с которой начинается процесс загрузки операционной системы (для MS-DOS и MS Windows-9X – файл Io.sys).

Загрузочный сектор содержит в себе, кроме кода загрузчика, таблицу BPB и двухбайтовую сигнатуру 55AA (0xAA55).

BPB — BIOS Parameter Block — таблица содержашая множество параметров, определяющих характеристики блочного устройства и самой файловой системы.

Таблица размещения файлов (FAT) – это массив целых чисел с длинной, равной количеству кластеров раздела файловой системы.

Номер элемента этого массива – это номер кластера в разделе файловой системы. Отдельный элемент массива:

• для FAT16 — это 2-х байтовые числа;

• для FAT32 — это 4-х байтовые числа.

Значение элемента таблицы FAT — это ссылка на следующий номер элемента таблицы FAT.

За таблицами FAT располагается корневой каталог файловой системы (Root Directory), размером:

• 512 байт, для FAT16.

• 2048 байт, для FAT32, сейчас — до 65535 элементов записей по 32 байта.

Корневой каталог файловой системы – список записей по 32 байта.

Область данных — все остальные кластеры раздела. Они используются для хранения подкатологов и файлов.

Подкатологи – это файлы, содержащие последовательности 32-битных записей имен файлов и подкаталогов, указанного выше формата.

Содержимое файлов — данные некоторой последовательности кластеров. Последовательность кластеров — произвольная и неупорядоченная, но отражена в виде цепочки, в таблице FAT.

* **Структура файловой системы EXT2FS.**

ext2 (ext2fs) - Second Extended File System — базовая файловая система ОС Linux. Является преемницей ext, которая поддерживала файлы размером до 2 Гбайт и длину имени файла до 255 байт.

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. Каждая группа блоков имеет одинаковое число блоков, кроме последней, которая может быть неполной.

Начало каждой группы блоков имеет адрес. Размер блока может быть 1, 2 или 4 килобайта, что определяется в момент форматирования – при создании файловой системы.

Блок является адресуемой единицей дискового пространства.

Каждая группа блоков имеет одинаковое строение:

Первый элемент группы блоков - суперблок, который одинаков для всех групп.

Все остальные элементы - индивидуальны для каждой группы.

* **Сравнение файловых систем.**

Структуры файловых систем FAT32 и ext2fs демострируют два разных подхода к обработке данных:

• простоту управления, характерную для FAT32;

• скорость обработки больших файлов, характерную для ext2fs.

В любом случае, обе файловые системы имеют общую метаструктуру, которая определяется наличием:

• секторов загрузчика ОС;

• суперблока, характеризующего файловую систему и раздел, в котором она расположена;

• набора таблиц для разметки и учета использованных блоков данных;

• корневого каталога, с которого начинается логическое построение ФС;

• набора блоков данных, в которых размещаются подкатологи и файлы ФС.

Преимущества FAT32:

• простота реализации и эффективность использования всего физического пространства блочных устройств малой емкости;

• широкая известность и распространенность, делающая ее «универсальным хранилищем» временных ФС.

Недостатки FAT32:

• практическая ограниченность размера ФС и слабая защищенность, делающая ее непригодной для современных ОС и технологий хранения информации;

• высокая фрагментированность ФС в процессе эксплуатации, что снижает скорость ее работы и негативно воздействует на устройства хранения данных.

Файловая система ext - это Extended minix.

Преимущества ext2:

• высокое быстродействие;

• поддержка файлов до 2 Гбайт.

Недостатки ext2:

• отсутствие журналирования, снижающее ее надежность;

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

Ext3 и ext4 являются журналированными ФС.

Журналируемая файловая система — это ФС, которая сохраняет список изменений, проводимых с файловой системой, перед фактическим их осуществлением.

* **Стандартизация структуры ФС.**

Структура ФС на уровне файлов и каталогов может быть произвольной, за некоторыми ограничениями, например:

• обязательно наличие корневой ФС ОС;

• обязательно наличие корневого каталога отдельных ФС.

В общем случае, файловая система Linux разделена на три крупных уровня иерархий:

1. Первый уровень иерархии ФС составляют каталоги, которые управляются пользователем root и другими администраторами системы.
2. Второй уровень иерархии ФС, по замыслу, содержит данные, которые доступны только для чтения и могут использоваться всеми пользователями или несколькими компьютерами сети. В идеале, модифицироваться могут только файлы каталога /usr/local, в который администратор может устанавливать новые программы, не поставляющиеся с системой.
3. Третий уровень иерархии ФС — подобен второму и продолжается в директории /usr/local.

Наиболее распространенные команды для работы с директориями:

• 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.

* **Модули и драйверы ОС.**

Ядро ОС — это большая программа, работающая в собственном защищенном (привелигированном) режиме.

Любой процесс работает в режиме пользователя. Потому, любой процесс имеет доступ к устройствам только через ядро ОС.

Чтобы работать с конкретным устройством, ядро ОС имеет специальное ПО, которое называется драйвером.

ПО драйвера имеет два интерфейса:

• интерфейс вызова драйвера ядром ОС, который использует концепцию узла ФС или node, подразделяя устройства на символьные и блочные; определяется разработчиками ОС

• интерфейс устройства, который использует сам драйвер для работы с конкретным устройством через общую шину ЭВМ. определяется архитектурой аппаратной части ЭВМ и конструктивными особенностями конкретного устройства.

Драйверы традиционно пишутся в виде модулей, которые хранятся в ФС ОС и загружаются по мере необходимости.

Поскольку без некоторой части драйверов ОС не сможет даже загрузиться, такие драйверы включаются в ядро ОС статически. Обычно, перед компиляцией ядра ОС, запускается программа-конфигуратор, с помощью которой определяется: какие драйверы будут включаются в ядро статически, а какие будут подгружаться динамически: во время загрузки ОС или по мере необходимости.

В общем случае, модуль — системное ПО ОС, которое не обязательно является драйвером. Главное, что это ПО предназначено для работы в защищенном режиме пространства ядра ОС.

Использование модулей предполагает учет их особенностей:

• управление модулями из режима пользователя осуществляется утилитами insmod, rmmod, lsmod, modinfo и modprobe;

• разработка модулей предполагает знание не только технологии их написания и отладки, но также - архитектуры и ПО ядра ОС;

• вывод информации в пространство пользователя модуль осуществляет посредством функции printk(...), которая записывает сообщение в файл /var/log/ messages;

• для просмотра сообщений модулей ядра используется утилита dmesg;

• файлы конфигурации и сами модули находятся в директориях ФС: /etc и

/lib/modules.

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

Один из общепринятых способов повышения мобильности ПО - обеспечение стандартизации окружения приложений. Это подразумевает стандартизацию ПО ОС, утилит и программных интерфейсов самих приложений. Таким средством является стандарт POSIX (Portable Operating System Interface), который описывает различные интерфейсы операционной системы.

Стандарт POSIX - обязательный элемент современной дисциплины разработки прикладных систем.

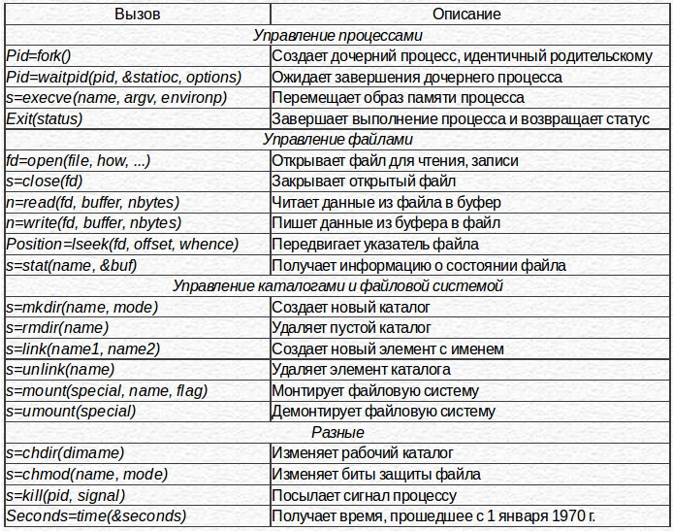
Список системных вызовов ОС, разделенных на группы:

• управление процессами;

• управление файлами;

• управление каталогами и файловыми системами;

• разные, которые нельзя полностью отнести к предыдущим группам.



* **Три концепции работы с устройствами.**

Для того чтобы, чтобы уже созданные ФС стали доступны приложениям ОС,

должны быть созданы узлы (node), которые расположены в директории /dev.

Узлы ОС могут быть созданы из пространства пользователя:

• с помощью системного вызова mknod(...) из программы на языке С;

• с помощью утилиты mknod посредством команды языка shell.

Исторически, сложилось три концепции (технологии) именования и отслеживания актуальности наличия устройств: manual, udev и systemd.

1. Концепция manual (ручная) появилась первой – начало 2000-х. Она предполагает использование, упомянутых выше, системного вызова mknod(...) или утилиты mknod.

Обычно, созданием устройств занимается программа-инсталлятор ОС, которая проверяет архитектуру компьютера на наличие имеющихся устройств, а также создает узлы устройств: автоматически или по требованию администратора ОС.

1. Концепция udev – 2003 год.

При старте ОС запускается демон udevd, который, используя шину D-Bus, отслеживает подключение новых устройств и создает для них нужные узлы.

Когда устройства отключается от ЭВМ, udevd удаляет соответствующие им узлы, кроме тех, которые были созданы в ручную.

Одновременно, также решается проблема поддержания в активном состоянии неиспользованных узлов.

1. Концепция systemd» - 2010.

Ее цель — унификация взаимодействия между процессами режима пользователя.

В частности:

* отслеживание динамики подключения устройств (вместо udevd);
* эффективное отслеживание групп процессов, порожденных системными вызовами fork(...) и объединенных одной прикладной целью (cgroups).
* **Разделы дисков и работа с ними.**

Когда созданы все ФС, на всех разделах блочных устройств, работа с файловой системой ОС становится достаточно простой:

• необходимо определить точки монтирования (директории) корневой ФС ОС для подключения к ним (монтирования) нужных сопутствующих ФС и затем, правильно записать желаемую конфигурацию в файле /etc/fstab;

• выполнить команду mount -a, и желаемая структура ФС ОС будет создана для последующей эксплуатации.

Хотя такой подход требует определенной квалификации пользователя и ориентирован больше на администраторов серверов.

Виртуальная файловая система VFS:

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

Это позволяет сконцентрировать в одном месте пользовательский интерфес для разных типов ФС, что создает унифицированный эталон для разработки драйверов ОС.

В результате, VFS:

• упрощает процесс создания драйверов ФС, что повышает их надежность;

• сокращает время разработки, что создает конкурентные преимущества самой ОС;

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

Все равно, основу управления ФС составляют утилиты mount и umount,

* **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 внешнего блочного устройства от конкретного каталога ФС, в видимой пользователю общей части дерева корневой ФС.

Общее правило монтирования/демонтирования ФС:

• монтирование осуществляется командой mount с указанием узла ФС и директории монтирования, при этом: старое содержимое директории становится невидимым, а монтируемая ФС — видимой;

• демонтирование осуществляется командой umount с указанием или узла ФС или директории монтирования, при этом: старое содержимое директории становится видимым, а демонтируемая ФС — невидимой;

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

Используется множество специальных ФС, рассмотрим некоторые из них.

Термин loopback означает - «обратная петля». Применительно к нашей тематике он будет означать:

• циклическое устройство, когда мы будем говорить о блочном устройстве;

• циклическая ФС, когда мы будем говорить о файловой системе, которая создана в отдельном файле.

Узлы циклических устройств находятся в директории /dev

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

Широкое распространение получила технология хранения файловых систем в сильно сжатых файлах для live-дистрибутивов. Такой подход позволяет практически на прямую использовать технологии созданные для работы с ФС на CD дисках.

Другая специальная ФС - overlayfs — является вариантом реализации каскадной файловой системы для ОС Ubuntu и ее клонов.

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, например для объединения ФС нескольких сайтов.

Overlayfs обеспечивает надежное монтирование двух файловых систем на разных уровнях.

Последняя специальная ФС, которую мы кратко рассмотрим — FUSE.

FUSE — Filesystem in Userspace — файловая система в пространстве пользователя. FUSE реализована в виде модуля для UNIX-подобных ОС. Этот модуль позволяет пользователям без привилегий создавать свои ФС, без необходимости переписывать код ядра ОС. Это достигается за счет запуска кода файловой системы в пространстве пользователя, в то время как модуль FUSE предоставляет мост для актуальных интерфейсов ядра.

* **Дисковые квоты.**

Рано или поздно может случиться так, что некоторая ФС на внешнем носителе, подключенная к древовидной структуре ядра ОС, заполнится и блокирует нормальную работу ОС.

По умолчанию, ФС ext2fs резервирует 5% места для пользователя root.

Принято считать, что для нормальной работы ФС требуется свободное пространство:

• не менее 10% дискового пространства (пространство раздела);

• не менее наибольшего файла в файловой системе.

Администратор ОС должен отслеживать наличие достаточного дискового пространства ФС. Для этих целей имеется специальное системное ПО под общим названием quota. Естественно, ядро ОС должно поддерживать команды этого пакета.

Чтобы включить квоты для конкретной файловой системы, нужно в корень ФС поместить бинарные файлы:

• quota.user — для персональных квот;

• quota.group — для групповых квот.

*Управление квотами* выполняется с помощью команд:

|  |  |
| --- | --- |
| quotastats | Проверка поддержки квот ядром ОС. |
| quotacheck | Сканирование заданной ФС и первоначальное создание файлов  *quota.user и quota.group*. |
| edquota | Редактор квот. |
| quotaon | Активация настроек квот для ФС. |
| quotaoff | Деактивация настроек квот для ФС. |
| quota | Проверка пользователем, установленных для него квот. |

**Тема 5**

* **Однопользовательский и многопользовательский режимы работы ОС.**

Современное понятие однопользовательского режима означает не тот факт, что ОС не может поддерживать многопользователский режим, а то что:

• или отключен контроль разграничения прав пользователей, при одновременной изоляции ЭВМ от внешних воздействий, например, отключение от сети;

• или остановлена работа программ всех пользователей, кроме администратора, например, суперпользователя 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 файловую систему, создает 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.

Практически всегда, под именем пользователя понимается контекст, соответствующий понятию владелец, который относятся применительно к файлам и процессам.

Информационное обеспечение многопользовательского режима ОС, прежде всего, поддерживается группой системных файлов.

* **Разграничение прав пользователей.**

Общая парадигма концепции пользователя подразумевает, что все пользователи ОС работают автономно и не мешают друг другу, кроме системного администратора.

Это достигается двумя основными мерами:

• 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.

На самом деле, имя пользователя имеет вспомогательное второстепенное значение. Главным показателем пользователя является его идентификатор UID: с увеличением номера UID права пользователя уменьшаются.

Аналогичный критерий справедлив для групп пользователей, права которых определяются идентификатором GID.

Условно, все пользователи разделяются на две категории:

• системные пользователи — root, sysadm и другие администраторы;

• обычные пользователи — те, которые используют прикладное программное обеспечение ОС и не занимаются администрированием.

* **Login и система доступа Linux-PAM.**

Любому пользователю, для нормальной работы с ОС, необходимо пройти процедуру регистрации в системе: получить имена, идентификаторы, пароли и место для работы. Указанную процедуру выполняет администратор ОС.

Процедура входа в ОС может быть: текстовой, когда на консоль терминала выводится приглашение login:, или графической, когда выводится некоторое стилизованное окно приглашения.

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

После ввода необходимой информации начинается с процедуры login, которая подразделяется на:

• идентификацию (аутентификацию), подразумевающую совпадение имени и пароля, зарегистрированных в системе;

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

Фактически, авторизация не заканчивается завершением утилиты login. Она проводится постоянно, когда пользователь обращается к файлам или взаимодействует с процессами.

Система PAM введена для создания дополнительного уровня защиты между приложениями и различными протоколами и способами идентификации и авторизации.

Модули PAM — это динамически загружаемые библиотеки, которые находятся в директориях /lib/security или /usr/lib/security.

Технологическая концепция модулей PAM предполагает разделение их на четыре типа:

* Auth Выполняют аутентификацию, то есть подтверждают, что пользователь является именно тем, кем он представился в системе.
* account Разрешают или запрещают конкретному пользователю вход в систему. Это решение может зависеть от даты, времени суток, системных ресурсов и т.д.
* session Осуществляют действия, которые должны быть выполнены до или после входа пользователя: занесение информации в журнальные файлы, монтирование устройств и другое.
* passwd Изменяют пароль пользователя.
* **Команды управления пользователями.**

Теоретическая концепция пользователя поддерживается соответствующей информационной и управляющей инфраструктурой ОС.

Хотя работа с пользователями предполагает всего три действия: создание, удаление и модификацию, - наличие множества конфигурационных файлов, привязанных к древовидной структуре ФС, превращает сам процесс управления в непростую задачу.

Чтобы упростить этот процесс, используются три команды (утилиты): useradd, userdel и usermod, расположенные обычно в директории /usr/sbin.

Useradd – создание нового пользователя.

Userdel – удаление пользователя.

Usermod - изменение параметров пользователя выполняется командой

Особое место в управлении пользователями занимает утилита passwd, которая управляет паролями пользователя и ограничивает его работу на уровне сеанса.

Общие правила применения утилиты passwd:

• администратор ОС может изменить пароль любого пользователя;

• работа утилиты, практически всегда происходит в интерактивном режиме; например, новый пароль вводится дважды;

• обычный пользователь может сменить только свой пароль, предварительно набрав старый.

**Тема 6**

* **Подсистема управления процессами.**

Процесс — это элементарный управляемый объект ОС, имеющий целочисленный идентификатор PID — Process Identification, обеспечивающий функциональное преобразование файлов (данных) с правами, которые определяются объектами пользователь.

Значения PID начинаются с 1 (обычно - это процесс init) — главный родительский процесс, и увеличиваются по мере порождения дочерних процессов. Новому процессу присваивается номер на 1 больше, чем максимальный номер существующего или существовавшего с момента запуска ОС процесса.

* все процессы находятся в среде пользователя (функционируют в пользовательском режиме);
* ядро ОС имеет подсистему управления процессами, которая функционирует в защищенном (привилегированном) режиме процессора.

В самой подсистеме управления процессами выделяются три подсистемы:

• взаимодействие процессов;

• планировщик;

• распределение памяти.

Управление процессами включает в себя два противоположно направленных воздействия:

1. процесс воздействует на подсистему управления процессами напрямую, через API ядра ОС посредством системных вызовов, или косвенно, через подсистему управления файлами;
2. подсистема управления процессами воздействует на процессы посредством трех выделенных механизмов: взаимодействия процессов, планировщика и средств распределения памяти.

* **Системные вызовы ОС по управлению процессами.**

Системный вызов fork(…) - обеспечивает порождение нового (дочернего процесса).

Системный вызов \_exit(…) - обеспечивает завершение процесса, вызвавшего его, удаляет PID процесса из таблицы (списка) процессов и передает родительскому процессу значение статуса завершения.

Системный вызов wait(…) - используется родительским процессом для ожидания завершения дочернего процесса и получения его PID.

Системный вызов exec(…) - обеспечивает различные варианты вызовов для модификации процесса.

* **Стандарты POSIX и сигналы.**

Понятие сигнала приводится как в инструментальных средствах языка программирования С, так и в стандарте POSIX (Portable Operating System Interface for uniX), описывающего соответствующие интерфейсы ядра ОС.

Стандарт POSIX-2001, под сигналом понимает механизм, с помощью которого процесс или поток управления уведомляют о некотором событии, произошедшем в системе, или подвергают воздействию этого события.

Примерами подобных событий могут служить:

• аппаратные исключительные ситуации;

• специфические действия процессов.

Термин "сигнал" используется также для обозначения самого события.

Говорят, что сигнал генерируется (или посылается) для процесса (потока управления), когда происходит вызвавшее его событие:

• выявлен аппаратный сбой;

• отработал таймер;

• пользователь ввел с терминала специфическую последовательность символов;

• процесс обратился к функции kill(...) и другие.

Примеры сигналов:

SIGHUP - Сигнал разъединения. Подразумеваемая реакция - аварийное завершение процесса.

SIGILL - Некорректная команда. Подразумеваемая реакция - аварийное завершение и создание файла с образом памяти процесса.

SIGINT - Сигнал прерывания, поступивший с терминала. Подразумеваемая реакция - аварийное завершение процесса.

SIGKILL - Уничтожение процесса (этот сигнал нельзя перехватить для обработки или проигнорировать). Подразумеваемая реакция - аварийное завершение процесса.

SIGPIPE – Попытка записи в канал, из которого никто не читает. Подразумеваемая реакция - аварийное завершение процесса.

* **Подсистема управления оперативной памятью.**

В активной форме (выполняться) процесс может только в оперативной памяти ЭВМ или ОЗУ.

ОЗУ — оперативное запоминающее устройство, которое часто называют основной памятью (ОП) ЭВМ.

MMU — Memory Managment Unit — устройство управления памятью.

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

Цель функционирования MMU — создание для нужд процессора виртуального (относительного) пространства памяти, в котором он работает.

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

Первоначально, все процессы ОС максимально разделены и защищены.

Все процессы обращаются к ядру ОС, требуя, захватывая и потребляя ресурсы. Ядро ОС планирует запуск процессов на выполнение, обслуживая процессы ресурсами и устраняя возникающие противоречия, когда процессы начинают конкурировать за общий ресурс.

Альтернативный подход к взаимодействию процессов - использование возможностей файловой системы ОС.

Преимущество: возможность использования любых средств взаимодействия.

Недостатки:

• алгоритмическая сложность разрешения всех тупиковых ситуаций, возникающих в приложениях;

• необходимость реализации средств взаимодействия непосредственно в приложениях;

• неэффективная и ненадежная реализация взаимодействия процессов, слабо обеспеченная возможностями ядра ОС;

• локальная семантика и интерпретация механизмов взаимодействия ограничивающая переносимость самих приложений.

Комбинированные подходы к взаимодействию процессов:

• максимальное использование возможностей ядра ОС;

• разработка новых эффективных алгоритмов взаимодействия процессов, которые включаются в функционал ядра ОС.

Одним из таких подходов является проект IPC - InterProcess Communication, который предполагает, что:

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

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

В стандарте POSIX-2001 разделяемый объект памяти определяется как объект, представляющий собой память ЭВМ, которая может быть параллельно отображена в адресное пространство более чем одного процесса.

Единицей отображения разделяемой памяти являются сегменты, в которые процесс может поместить данные любой структуры, читать данные этой структуры и модифицировать их.

Механизм IPC гарантирует процессам, что выделяемый сегмент памяти будет не меньше размера записываемой структуры данных и будет сохранен ядром ОС, даже если процесс завершит свою работу.

Таким образом, вариант взаимодействия с помощью разделяемой памяти обеспечивает процессам:

• асинхронный способ взаимодействия, не требующий одновременного существования всех взаимодействующих процессов;

• наиболее быстрый доступ к общим структурированным данным.

* **Передача сообщений.**

В стандарте POSIX-2001, очереди сообщений — набор типизированных объектов, которыми процессы могут асинхронно взаимодействовать между собой, не затрачивая свои прикладные ресурсы на организацию и сопровождение самих очередей. Очереди сообщений - это наиболее семантически нагруженный способ взаимодействия процессов через линии связи, в котором на передаваемую информацию накладывается определенная структура, так что процесс, принимающий данные, может четко определить, где заканчивается одна порция информации и начинается другая. Такая модель позволяет задействовать одну и ту же линию связи для передачи данных в двух направлениях между несколькими процессами.

Непосредственная работа с очередями сообщений обеспечивается с помощью четырех системных вызовов:

* Msgget - Создает очередь сообщений или подключается к уже созданной очереди, используя сгенерированный функцией ftok(...) уникальный ключ. Возвращает целочисленный дескриптор очереди, который используется остальными тремя системными вызовами.
* Msgsend - Помещает в очередь сообщение заданного типа.
* Msgrcv - Читает из очереди сообщение заданного типа.
* Msgctl - Управляет очередью с использованием специальной структуры типа msgid\_ds. В частности, удаляет очередь из системы.
* **Главный родительский процесс init.**

Процесс — это элементарная защищенная единица выполняемого в режиме пользователя программного кода, которым управляет ядро ОС, предоставляя ему системные ресурсы компьютера в виде процессорного времени и обмена данными с внешними устройствами.

Главный родительский процесс — процесс init:

• первый созданный ядром ОС и последний в работе системы;

• являющийся родителем для всех остальных процессов.

Уникальные свойства процесса init делают его особенным в системе:

• он не имеет родительского процесса, а создается ядром ОС;

• обладает свойствами родительского процесса, но не обладает свойствами дочернего процесса;

• выполняет сугубо системные функции, организуя работу других процессов;

• его удаление означает остановку работы ОС.

Особая важность процесса init делает его объектом постоянных исследований и усовершенствований.

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

Подходы к управлению процессов:

1. Монопольный подход к управлению процессами применяется:

* во встроенных системах, когда вычислительный процесс решает одну или несколько простых задач, не требующих сложных организационных мероприятий по управлению процессами;
* в узко специализированных системах, требующих централизованного управления всеми вычислительными аспектами приложения;
* в монопольном режиме работы администратора ОС, необходимом для настройки или восстановления работы ОС;
* при загрузке современных ОС, предполагающей промежуточный этап работы ОС перед загрузкой и монтированием основной файловой системы.

1. Upstart - cистема инициализации, основанная на обработке сигналов.

Основные понятия upstart:

• job — работа — общее название запускаемого ПО;

• task - задача - разновидность работы, предполагающая запуск и завершение;

• service - сервис - разновидность работы, аналог демона, которая перезапускается при падении или аварийном завершении.

1. Подход UNIX System V ориентирован на универсальное применение. Фактически он является классическим примером, организующим управление процессами в UNIX и Linux системах, применяемым и до настоящего времени. Основная идея System V - ядро и ОС в целом могут работать на различных уровнях.
2. Systemd - демон инициализации, призванный унифицировать управление устройствами и процессами, заменив существующие программные средства, включая init.   
   Sytemd является свободным ПО с лицензией GNU v.2.1, который опирается на концепцию сервиса, выделяя:

* сокет-активные и шино-активные сервисы, что часто приводит к лучшему распараллеливанию взаимозависимых сервисов;
* сервисные процессы cgroups, использующие специальные идентификаторы групп, вместо идентификаторов процессов PID, что гарантирует отслежива- ние главных демонов приложений и не допускает потери процессов, при их разветвлении.
* **Порождение и завершение процессов, просмотр их состояния и изменение приоритета.**

Новые процессы создаются запуском скрипта или исполняемого приложения.

Процессы завершаются:

• нормально - в соответствии с работой алгоритма приложения, который исполняется процессом;

• аварийно или нормально — при получении процессом сигнала;

• аварийно — при завершении процесса системой.

Для посылки сигналов процессам, используется утилита kill.

Для управления процессами также используются команды:

• прерывания — нажатие клавиш Ctl-Z;

• bg — перевод процесса в фоновый режим;

• fg — перевод процесса в приоритетный режим.

* **Состояния процессов в ядре ОС.**

Все процессы ОС выполняются в режиме пользователя. Управление работой процессов осуществляет ядро ОС. Для организации управления процессами ядро ОС:

• присваивает процессу уникальный идентификатор PID;

• выделяет процессам для работы некоторые кванты времени t;

• переводит процесс в одно из состояний, обеспечивая мультипрограммный режим работы ОС.

Классическая для UNIX-систем схема планирования, учитывает три состояния: готовность, выполнение и ожидание.

1. Выполнение - активное состояние процесса, во время которого процесс обладает всеми необходимыми ресурсами и непосредственно выполняется процессором.
2. Ожидание - пассивное состояние процесса, процесс заблокирован, он не может выполняться по своим внутренним причинам, он ждет осуществления некоторого события, например, завершения операции ввода-вывода, получения сообщения от другого процесса, освобождения какого-либо необходимого ему ресурса.
3. Готовность - также пассивное состояние процесса, но в этом случае процесс заблокирован в связи с внешними по отношению к нему обстоятельствами: процесс имеет все требуемые для него ресурсы, он готов выполняться, однако процессор занят выполнением другого процесса.

* **ОС реального времени.**

Системы реального времени - допускают интерактивное взаимодействие с пользователем, но ориентированы на автоматическое управление различными техническими системами. Поскольку требования к стратегии планирования определяется требованиями к прикладной задаче управления, то критерии ОС реального времени должны выражаться через терминологию задач.

Основные требования к задачам реального времени:

• окончание работы к сроку и исключение потери данных;

• предсказуемость, которая предполагает предотвращение деградации качества в мультимедийных системах.

Общая стратегия планирования в таких системах - запуск на выполнение каждого процесса через интервал времени, не превышающий некоторого значения T.

* **Алгоритм разделения времени.**

Все стратегии планирования реализуются конкретными алгоритмами планирования.

В общем случае, планирование процессов включает решение следующих задач:

• определение момента времени для смены выполняемого процесса;

• выбор процесса на выполнение из очереди готовых процессов;

• переключение контекстов "старого" и "нового" процессов.

Примениельно к алгоритмам, существует два типа процедур планирования:

• вытесняющая многозадачность

• невытесняющая многозадачность

Preemptive multitasking - вытесняющая многозадачность - способ, при котором решение о переключении процессора, с одного процесса на другой, принимается планировщиком операционной системы, а не самой активной задачей.

Non-preemptive multitasking - невытесняющая многозадачность - способ планирования процессов, при котором активный процесс выполняется до тех пор, пока он сам, по собственной инициативе, не отдаст управление планировщику ОС для того, чтобы тот выбрал из очереди другой, готовый к выполнению процесс.

Реализация алгоритмов разделения времени, как и других алгоритмов планирования, сталкивается с существенными проблемами, что делает их достаточно сложными:

• проблема синхронизации — борьба процессов за один ресурс, которую мы рассмотрим в последующих подразделах, на примере разделяемой памяти;

• проблема тупиков — бесконечное ожидание ресурса, который занят и не освобождается другим процессом; этот вопрос выходит за рамки первой части данной дисциплины.

**Тема 7**

* **Язык С как стандарт взаимодействия с ОС.**

Язык С рассматривается в двух контекстах:

• как высокоуровневый инструмент для написания самих ОС;

• как низкоуровневый инструмент для написания прикладных программ, взаимодействующих с ядром ОС.

У каждого интерфейса есть две стороны: вызывающая и вызываемая.

Стандарт POSIX ориентирован в первую очередь на вызывающую сторону, что делает его полезным как для системных, так и прикладных программистов.

Цель стандарта POSIX - сделать приложения мобильными на уровне исходного кода языка. Это значит, что, при переносе программ языка С на другую операционную платформу, потребуется только новая компиляция исходных текстов программ.

Реализация или операционная система, соответствующая стандарту POSIX, должна поддерживать все обязательные служебные программы, функции, а также заголовочные файлы с обеспечением специфицированного в стандарте поведения.

* **Системные операции для работы с файловой системой.**

Основные системные вызовы, разделенных на две группы

• управление файлами;

• управление каталогами и файловыми системами.

Системный вызов open(...) открывает некоторый файл и, в случае успешного завершения, возвращает целочисленный номер дескриптора. В случае ошибочного завершения, функция open(...) возвращает отрицательное значение -1.

Системный вызов close(...) закрывает открытый файл и освобождает номер дескриптора для последующего использования.

В случае успешного завершения вызова, возвращается целочисленное значение 0. В случае ошибки — целочисленное значение -1.

Системный вызов read(...) читает заданное число байт из файла, определен- ного его дескриптором. В случае успешного завершения операции чтения, данные помещаются во внешний буфер и возвращается реальное число прочитанных байт.

Для вызова этой функции используется три аргумента:

• fd — целочисленный номер дескриптора открытого файла;

• buf — указатель на внешний буфер, в который записываюся читаемые байты;

• count — количество байт запрашиваемых для чтения из файла.

Системный вызов write(...) записывает заданное число байт из буфера в файл, определенный его дескриптором. В случае успешного завершения операции, данные помещаются в файл, начиная с текущей позиции, и возвращается реальное число записанных байт.

В этом системном вызове используются те же аргументы, что и в функции read(...). В случае ошибки записи, возвращается значение -1.

Системный вызов lseek(...) устанавливает позицирование в файле, определенном его дескриптором. В случае успешного завершения операции, возвращается место новой позиции в байтах от начала файла.

Для вызова этой функции используется три аргумента: fd — целочисленный номер дескриптора открытого файла; offset — целочисленное значение смещения в байтах; whence — директива, задающая интерпретацию смещения:

* **Создание специальных файлов.**

Сама VFS (виртуальная файловая система) реализована в ядре ОС и все операции с файлами осуществляются в пространстве ее представлений. Фактически, VFS, является виртуальным коммутатором всех внешних файловых систем, подключенных (монтированных) к ней.

Наряду с изученной ранее функцией open(...), которая позволяет как откры- вать, так и создавать файлы, в стандарте POSIX имеется функция mknod(...), обеспечивающая возможность создавать и различные специальные файлы.

Специальные файлы, после их создания, получают свои номера i-узлов, но, в отличие от обычных файлов, они в действительности есть только указатели на соответствующие драйверы устройств в ядре. По сравнению с обычными файлами файлы устройств имеют три дополнительных атрибута, которые характеризуют устройство, соответствующее данному файлу:

1. класс устройства - фактически соответствует понятию тип устройства.
2. старший номер устройства - группирует типы устройств, например, жесткий диск или звуковая плата.
3. младший номер устройства - применяется для нумерации устройств одного типа, другими словами, - устройств с одинаковыми старшими номерами.

* **Запрос информации о статусе файлов.**

Системный вызов stat(...) по заданному имени файла позволяет получить информацию о свойствах любых типов файлов.

Дополнительно, стандарт POSIX предоставляет функции:

• fstat(...) вместо имени файла использует номер его дескриптора;

• lstat(...) использует имя ссылки на файл и возвращает информацию о самой ссылке, а не о файле, на который она ссылается.

В стандарте POSIX предусмотрены макросы, которые проверяют, является ли файл:

• S\_ISLNK(m) — символьной ссылкой;

• S\_ISREG(m) — обычным файлом;

• S\_ISDIR(m) — каталогом;

• S\_ISCHR(m) — символьным устройством;

• S\_ISBLK(m) — блочным устройством;

• S\_ISFIFO(m) — каналом FIFO;

• S\_ISSOCK(m) — сокетом.

* **Каналы.**

Операционные системы предоставляют программисту два типа каналов:

• ***полудуплексные каналы UNIX;***

Канал создается в ядре ОС. Когда процесс создает канал, ядро устанавливает два файловых дескриптора для пользования этим каналом. Один такой дескриптор используется, чтобы открыть путь ввода в канал (запись), в то время как другой применяется для получения данных из канала (чтение).

Основное назначение неименованных полудуплексных каналов — взаимодействие между родительским и дочерними процессами. В этом случае используется то свойство, что дочерний процесс наследует все ресурсы, открытые родительским процессом, в том числе и открытые каналы.

• ***именованные каналы FIFO.***

Для целей взаимодействия не родственных («чужих») процессов предназначены именованные каналы FIFO, которые также создаются в ядре ОС, но имеют имена, отображаемые в файловой системе.

Типичное применение каналов FIFO - разработка приложений «клиент — сервер», когда:

• несколько процессов могут записывать или читать FIFO одновременно;

• режим работы с FIFO - полудуплексный, что позволяет процессам общаться только в одном из направлений.

Создать канал FIFO можно с помощью функции mknod(...), если во втором параметре mode установить тип файла S\_IFIFO, а третьему параметру dev присвоить значение 0.

Стандарт POSIX предоставляет для создания именованных каналов специальный системный вызов mkfifo(...)

* **Дублирование дескрипторов файлов.**

Интерпретатор shell, запуская процесс, передает ему три дескриптора файлов: 0 — для чтения; 1 и 2 — для записи.

Стандарт POSIX предоставляет два системных вызова дублирования дескрипторов файлов: dup и dup2.

Обе функции, в случае нормального завершения вызова:

• возвращают новый дескриптор файла;

• функция dup2(…), если требуется, предварительно закрывает старый дескриптор.

Старый и новый дескрипторы можно использовать друг вместо друга. Они имеют общие блокировки, указатель позиции в файле и флаги.

Если произошла ошибка, то возвращается значение -1

* **Монтирование и демонтирование ФС.**

Все ОС имеют системные вызовы, предназначенные для подключения и отключения внешних файловых систем к корневой файловой системе. Стандарт POSIX определяет эти функции как mount и unmount.

В функции mount указывается: source — раздел блочного устройства или файл, содержащий файловую систему, известную ядру ОС; target — директория, присутствующая в файловой системе, к которой монтируется внешняя файловая система; mountflags — опции, с которыми монтируется файловая система;

В функции unmount указывается special — это source или target, которые специфицируют точку монтирования.

* **Ссылки на имена файлов.**

Программисту и обычному пользователю, любая отдельная файловая система ОС представляется в виде иерархии имен файлов, в которой особое место занимают файлы типа директорий (каталогов).

В стандарте POSIX для создания и удаления каталогов определяет два системных вызова: mkdir(...) и rmdir(...).

mkdir(...) – создание новой директории. При успешном завершении возвращается ноль или -1, если произошла ошибка.

rmdir(...) – удаление директории. В случае успеха возвращается ноль. При ошибке возвращается -1,

Для создания новых имен, которые интерпретируются как имена файлов, стандарт POSIX предоставляет два системных вызова:

• link() - создает новую ссылку на существующий файл (на i-узел) внутри одной файловой системы, известную также как "жесткая" ссылка; она имеет те же свойства, что и старое имя, поэтому их оригинальность их установить невозможно; В случае успеха возвращается ноль. При ошибке возвращается -1

• symlink() - создает символьную ссылку на имя предположительно существующего файла, независимо от файловой системы, в которой оно должно находиться; если такого файла не существует, то ссылка называется «висячей». В случае успеха возвращается ноль. При ошибке возвращается -1

**Тема 8**

* **Классификация способов управления памятью ОС.**

С появлением и совершенствованием первых ОС, которые также, как и прикладное ПО, являются программным обеспечением, стали применять три способа загрузки:

• первый способ предполагает первоначальную загрузку ОС, обычно в младшие адреса ОЗУ.

• второй способ стал применяться в некоторых карманных компьютерах и встроенных системах, которые обладают малыми ресурсами и, как правило, разными конструктивными особенностями. здесь ОС пишется в ПЗУ, которое соответствует верхним адресам памяти, оставляя ОЗУ полностью для ПО пользователей;

• третий способ соответствует большинству современных систем, подобных IBM PC;

В классификации способов распределения памяти выделяются две большие группы подходов:

* + - методы, *не использующие внешнюю память*, обычно, накопители на жестких дисках;
    - методы, *использующие внешнюю память*, которая теоретически продолжает (расширяет, увеличивает) ОЗУ ЭВМ.

Методы, не использующие внешнюю память, предполагают, что ОЗУ достаточно для решения самой крупной задачи, или, что программу можно исполнять по частям, для которых ОЗУ достаточно. Здесь, в большинстве случаев, задачи распределения памяти ставятся как оптимальное разбиение ОЗУ на сегменты (отдельные и не пересекающиеся участки памяти), обеспечивающие максимальную эффективность использования процессорного времени ЭВМ.

Методы, использующие внешнюю память, предполагают, что ОЗУ расширяется за счет памяти медленных внешних устройств и становится возможным одновременное выполнение необходимого числа «параллельно» работающих процессов.

* **Программный и аппаратный способы управления памятью.**

Нормальная работа компьютера обеспечивается двумя способами адресации памяти ЭВМ: программным и аппаратным.

Программный способ адресации неявно присутствует в любом исходном тексте программы в виде операторов команд программы, которые обращаются к объявленным переменным данных и осуществляют различные управляющие действия по определению порядка выполнения самих команд.

Аппаратный способ адресации заложен сам процессор:

• специализацией регистров процессора: сегментные, смещения и индексные;

• методами адресации команд процессора;

• шинами (каналами) процессора.

MMU — Memory Managment Unit — диспетчер памяти. В его обязанности входит:

• виртуализация адреса, при взаимодействии процессора с ОЗУ;

• аппаратное обеспечение алгоритмов распределения блоков ОЗУ в различных режимах адресации.

* **Страничная и сегментная адресации памяти.**

Сегментная адресация ОЗУ реализуется с помощью специальных регистров процессора, которые так и называются: регистры сегментов и регистры смещения. Сегментный способ адресации ОЗУ всегда использует таблицу сегментов, которая сама размещается в ОЗУ и предназначена для решения двух основных задач:

• виртуализация относительных адресов программы относительно физической памяти ЭВМ;

• управление свопингом и перемещение сегментов в физическом пространстве ОЗУ. Управление свопингом — управление перемещением сегментов программ из ОЗУ в специальные файлы или разделы свопинга, расположенные на внешних ЗУ, и обратно.

Основная проблема сегментгой адресации ОЗУ заключается в ее фрагментации, которая вызвана разным размером загружаемых и выгружаемых сегментов:

• при равных размерах сегментов, фрагментация была бы минимальна или ее удалось бы устранить;

• чем меньше разделы сегментов, тем меньше становятся непроизводительные потери ОЗУ.

Идея использования блоков одинакового размера, для адресации физической и логической частей ОЗУ, стала называться страничной организацией памяти.

Основное достоинство страничной адресации ОЗУ - отсутствие фрагментации, что не требует затрат ресурса ЭВМ на ее устранение.

Основная проблема адресации с помощью страниц - большой размер таблиц адресации, которые должны храниться в ОЗУ.

Другая проблема, с которой сталкивается страничный способ адресации — значительные затраты времени, связанные с поиском информации о странице в больших таблицах страниц.

Чтобы разрешить эту проблему, стали использовать таблицу быстрой трансляции адресов или — буфер TLB.

TLB — Translation Lookaside Buffer — ассоциативная кэш-память, в которую записывается реально используемая страница, перед тем как команды и данные будет использовать сам процессор.

* **Комбинированный способ адресации памяти.**

Преимущества адресации имеются у каждого способа:

• сегментная адресация памяти — требует относительно малое число записей в таблице сегментов.

• страничная адресация памяти — устраняет проблемы фрагментации, пос- кольку ОЗУ адресуется с помощью произвольного доступа, следовательно, свободную страницу ОЗУ можно найти в любом месте памяти.

Отсюда возникает идея смешанной (комбинированной) адресации, которая состоит в том, что:

• отдельная программа (процесс) отображается в виде нескольких записей в общей таблице сегментов;

• каждый сегмент, выделенный программе, отображается внутри нее в виде

одной или нескольких таблиц страниц.

Таким образом, сегментно-страничная адресация памяти похожа на подобную двухуровневую страничную адресацию. При этом:

• существенно уменьшается размер корневой таблицы, постоянно хранящейся в ОЗУ и как следствие ускоряется поиск в ней нужной страницы;

• трехкомпонентная адресация (номер сегмента, номер страницы, смещение) требует аппаратной поддержки виртуализации памяти с помощью MMU, без которой она становится не эффективной;

• появляется возможность оптимизации схем адресации на базе учета разных свойств разных сегментов программы; например, сегменты кода предназначены только для чтения.

* **Системные вызовы ОС по управлению памятью.**

Виртуальная структура процесса состоит из отдельных частей, которые, по- традиции, также называются сегментами:

• cегмент text - содержит машинные инструкции и константы; его содержание не изменяется при выполнения процесса;

• сегмент data - содержит внешние и статические переменные, которые инициализированы, при компиляции программы процесса;

• сегмент bss - содержит внешние и статические переменные, которые неинициализированы, при компиляции программы процесса;

• сегмент stack - используется для хранения значений автоматических переменных и параметров функций, а также для организации рекурсивных вызовов функций.

Системные функции:

• malloc(...) - выделяет область неинициализированной памяти, размер которой указан в байтах

• free(...) — освобождает память ОЗУ, которая ранее была выделена одной из функций: malloc(...), calloc(...) или realoc(...);

• calloc(...) - выделяет память ОЗУ под массив данных, размерность которого задана первым аргументом, а размер каждого элемента массива в байтах, задан вторым аргументом; при этом, значение каждого байта выделенной памяти обнуляется;

• realloc(...) - изменяет размер уже выделенного блока памяти, используя первый аргумент как указатель, а второй — как нужный размер;

**Тема 9**

* **Подсистема управления процессами.**

Подсистема управления процессами находится в ядре ОС. Основная ее функция — обеспечение мультипрограммного режима работы ОС, что связано с:

1. созданием процессов в системе и удаление их из системы, предполагает управление основной памятью ЭВМ;
2. переключением процессов в режимы «Готовность», «Выполнение» и «Ожидание».

Все системы должны обеспечить:

1. Справедливость — предоставление каждому процессу справедливой доли процессорного времени.
2. Принудительное применение политики - контроль за выполнением принятой политики;
3. Баланс — поддержка занятости всей системы.

Системам пакетной обработки данных необходима:

* *Пропускная способность* — максимальное количество задач в час.
* *Оборотное время* — минимизация времени, затрачиваемого на ожидание обслуживания и обработку задачи.
* *Использование процессора* — поддержка постоянной занятости процессора.

Интерактивным системам важно:

* *Время отклика* — быстрая реакция на запросы.
* *Соразмерность* — выполнение пожеланий пользователя.

Системам реального времени требуется:

* *Окончание работы к сроку* — предотвращение потери данных.
* Предсказуемость - предотвращение деградации качества в мультимедийных системах.
* **Синхронизация процессов.**

Основы синхронизации заложены в самой модели процесса:

• процесс создается на основе родительского процесса, наследуя от него программный код и все открытые ресурсы;

• родительский процесс отслеживает завершение дочернего процесса, тем самым синхронизируя иерархию процессов и разгружая ядро ОС от прикладных аспектов взаимодействия процессов.

Общий «механизм» синхронизации процессов с помощью каналов основан на функциях блокирования операций чтения из канала и записи в канал:

• неименованные или полудуплексные каналы UNIX обеспечивают синхронизацию только родственных процессов;

• именованные каналы, в которых задействована файловая система ОС, обеспечивает синхронизацию всех процессов, даже еще не запущенных в системе.

Недостатки такой синхронизации:

• неформатированный обмен данными, что требует выделения переданного сообщения на каждой взаимодействующей стороне;

• необходима дополнительная синхронизация, даже при реализации схемы один-ко-многим.

Существенный недостаток использования каналов - невозможность взаимодействия произвольных процессов ОС.

* **Стандарты POSIX.**

Процессы являются основными функциональными элементами операционных систем (ОС). Формальное определение процесса дается, через определение его атрибутов.

Процесс - адресное пространство вместе с выполняемыми в нем потоками управления, а также системными ресурсами, которые этим потокам требуются.

Идентификатор процесса - положительное целое число, которое однозначно идентифицирует процесс в течение времени его жизни.

Время жизни процесса - период времени от его создания до возврата идентификатора операционной системе.

Активный процесс - процесс, созданный с помощью функции fork(...) до его завершения и имеющий, по крайней мере, один поток управления и собственное адресное пространство.

Зомби-процесс - завершившийся процесс, подлежащий ликвидации после того, как код его завершения будет передан ожидающему этого другому процессу.

Родительский процесс - процесс, создавший данный процесс.

* **Системные вызовы ОС по управлению процессами.**

Системные вызовы разделены на три группы управления:

* атрибутами процессов;
* порождения и завершения процессов;
* потоки и сигналы процессов.

Каждый процесс ОС имеет свой уникальный целочисленный идентификатор ID (PID).

Такие же идентификаторы имеются и у групп процессов — GID.

Чтобы из программы, написанной на языке С, определить нужные идентификаторы атрибутов процессов, предусмотрены три системные функции getpid(...), getppid(...), getpgrp(...),

Семантика этих функций следующая:

• getpid(...) - возвращает PID текущего процесса;

• getppid(...) - возвращает PID родительского процесса;

• getpgrp(...) - возвращает GID текущего процесса;

• getsid(...) - возвращает идентификатор сессии (SID) для процесса, заданного идентификатором PID.

setpgid(...) предназначен для установки идентификатора группы процессов, например, в целях управления заданиями. Его выполнение влечет либо присоединение к существующей группе процессов, либо создание новой группы в рамках сеанса, в которую входит вызывающий процесс.

Setsid - создает новый сеанс для текущего процесса, в котором этот процесс становится ведущим группы.

* **Системный вызов fork() и каналы процессов.**

Группа системных вызовов, связанных с порождением и завершением процессов.

• fork(...) - порождает новый дочерний процесс;

• wait(...) - ожидает завершение любого дочернего процесса, регистрируя их в системе и завершая сам процесс удаления из системы;

• waitpid(...) - ожидает завершение процесса с конкретным номером PID.

exec\* - Набор системных вызовов для смены «тела» текущего процесса.

Exit - Нормальное завершение процесса с заданным статусом.

Atexit - Регистрация функции, которая будет вызвана при нормальном завершении процесса (при вызове функции exit(...)).

* **Нити (threds).**

Каждый, запущенный ОС, процесс имеет одну нить исполнения, которая называется главной или начальной нитью. Дополнительные нити (threads) создаются программистом из главной нити посредством набора системных вызовов.

Все дополнительные нити процесса разделяют:

• его программный код;

• глобальные переменные;

• системные ресурсы.

Но каждая дополнительная нить имеет:

• собственный программный счетчик;

• свое содержимое регистров;

• свой стек.

Создание новой нити обеспечивается с помощью системного вызова clone(...).

Для работы с нитями разработано большое количество функций. Далее, мы рассмотрим только функции, которые реализуются следующими системными вызовами ОС:

• pthread\_create() - создание нити;

• pthread\_exit() - завершение работы нити;

• pthread\_join() - другой вариант завершения работы нити;

• pthread\_self() - получение идентификатора нити.

* **Сигналы POSIX.**

Формально, стандарт POSIX-2001 под сигналом понимает «механизм», с помощью которого процесс или поток управления уведомляют о некотором событии, произошедшем в системе, или подвергают воздействию этого события.

Системные вызовы для работы с сигналами:

• signal() - установка нового обработчика сигнала текущего процесса;

• kill() - посылка сигнала процессу по его иденитфикатору PID;

• raise() - посылка сигнала текущему процессу;

• pause() - остановка процесса до получения сигнала;

• sleep() - остановка процесса на заданное число секунд, либо до получения сигнала.

Примеры сигналов:

SIGHUP - Сигнал разъединения. Подразумеваемая реакция - аварийное завершение процесса.

SIGILL - Некорректная команда. Подразумеваемая реакция - аварийное завершение и создание файла с образом памяти процесса.

SIGINT - Сигнал прерывания, поступивший с терминала. Подразумеваемая реакция - аварийное завершение процесса.

SIGKILL - Уничтожение процесса (этот сигнал нельзя перехватить для обработки или проигнорировать). Подразумеваемая реакция - аварийное завершение процесса.

SIGPIPE – Попытка записи в канал, из которого никто не читает. Подразумеваемая реакция - аварийное завершение процесса.

**Тема 10**

* **Проблемы распределения ресурсов ОС.**

В борьбе за общий разделяемый ресурс, процессы входят в состояние состязания.

Состояние состязания - ситуация, в которой два или более процессов считывают или записывают данные одновременно, а результат зависит от того, кто из них был первым.

Основной способ решения этих проблем - запрет на одновременное использование разделяемого ресурса или взаимное исключение.

Взаимное исключение предполагает, что одновременно не могут выполняться две критические секции разных процессов.

Критической секцией (критической областью) называется часть кода процесса, который обеспечивает доступ к разделяемым ресурсам.

Чтобы сформулировать общую проблематику использования любых разделяемых ресурсов, были сформулированы четыре необходимых условия:

• два процесса не должны находиться в критических областях;

• в программе не должно быть предположений о скорости или количестве процессов;

• процесс, находящийся в критической области, не может блокировать другие процессы;

• невозможна ситуация, когда процесс вечно ждет попадания в критическую область.

Состояние взаимной блокировки возникает тогда, когда процесс успел захватить только часть разделяемых ресурсов, необходимых ему для выполнения работы, а другую часть захватили другие процессы. В такой ситуации, все процессы переходят в состояние бесконечного ожидания необходимых им ресурсов.

Примером задачи, приводящей к взаимным блокировкам, является «Задача об обедающих философах»

* **Системный пакет IPC.**

IPC (Inter Process Communication) — ориентирован на решение трех проблем:

• надежная и простая передача данных и сообщений от одного процесса другому;

• контроль над деятельностью процессов в критических ситуациях;

• согласование действий между процессами.

Непосредственная реализация пакета разделена на четыре части:

• средства адресации IPC

• семафоры,

• разделяемая память и очереди сообщений

* **Утилиты управления средствами пакета IPC.**

У пользователя ОС, для работы со средствами межпроцессного взаимодействия System V IPC, имеются три основные утилиты, которыми следует пользоваться по мере необходимости:

• ipcmk — создание различных ресурсов средств IPC; ipcmk позволяет открывать разделяемые сегменты памяти, очереди сообще- ний и массивы семафоров.

• ipcs - вывод отчета о состоянии средств межпроцессного взаимодействия; ipcs выводит информацию о системных средствах межпроцессного взаимодействия System V IPC.

• ipcrm - удаление очередей сообщений, наборов семафоров и разделяемых сегментов памяти. ipcrm удаляет указанные очереди сообщений, семафоры и разделяемые сегменты памяти из системы. Требуемые объекты System V IPC задаются идентификатором их создания или любым связанным с ними ключом.

* **Семафоры.**

Одним из первых механизмов, предложенных для синхронизации поведения процессов, стали семафоры.

Основная идея реализации этого механизма, предполагает, что:

• семафор - это минимальный примитив синхронизации, служащий основой для более сложных механизмов синхронизации, определенных в прикладной программе;

• у семафора есть значение, которое представляется целым числом в диапазоне от 0 до 32767;

• прикладная реализация механизма синхронизации обеспечивается набором (массивом) семафоров, операции над этими наборами, для приложений являются атомарными;

• гарантом атомарности операций наборами является ядро ОС, в котором и реализованы механизмы синхронизации.

Непосредственно в пакете IPC, работа с семафорами осуществляется с помощью трех системных вызовов:

1. Системный вызов semget(...) предназначен для выполнения операции доступа к массиву IPC-семафоров или создание такого массива.
2. Системный вызов semop(...) предназначен для выполнения операций A, D и Z на основе ключа, который получен системным вызовом semget(...).
3. системный вызов semctl(...) предназначен для управления набором семафоров

* **Задача об обедающих философах.**

Семафоры:

• обеспечивают программиста надежными средствами синхронизации процессов;

• требуют от программиста разработки и реализации алгоритмов, обеспечивающих взаимное исключение процессов и устранение их взаимных блокировок.

Описание задачи:

За круглым столом сидит несколько философов.

В каждый момент времени каждый из них либо беседует, либо ест.

Для процесса еды одновременно требуются две вилки. Поэтому, прежде чем в очередной раз перейти от беседы к приему пищи, философу необходимо дождаться, пока освободятся обе вилки - слева и справа от него, и взять их в руки.

Немного поев, философ кладет вилки на стол и вновь присоединяется к беседе.

Требуется разработать программную модель обеда философов.

Описание действий философа:

философ какое-то время беседует (trnd), затем пытается взять вилки слева и справа от себя; когда ему это удается, он некоторое время ест (ernd), после чего освобождает вилки; так продолжается до тех пор, пока не будет съеден весь обед, после чего программа, моделирующая действия философа, закончит свою работу.

Для нормальной работы философа необходимы два ресурса, которые обеспечиваются двумя семафорами.

**Тема 11**

* **Прикладные средства пакета IPC.**

Чтобы обеспечить эффективную разработку приложений, особенно в плане взаимодействия асинхронно выполняющихся процессов, был разработан пакет System V IPC.

Разделяемая память — информационный объект данных, создаваемый и хранящийся в ядре ОС, который процесс может:

• создать или удалить;

• подключить к своему пространству данных или отсоедениться от него;

• работать с ним как с собственной структурой данных.

Очередь сообщений — универсальный «механизм» временного хранения в ядре ОС последовательности типизированных данных, которые процессы могут помещать и извлекать для своих нужд.

В совокупности с универсальным «механизмом» адресации (идентификации) и «механизмом» семафоров, разделяемая память и очереди сообщений образуют набор прикладных средств системного пакета System V IPC.

* **Разделяемая память.**

В стандарте POSIX-2001 разделяемый объект памяти определяется как объект, представляющий собой память ЭВМ, который может быть параллельно отображен в адресное пространство более чем одного процесса.

Таким образом, процессы могут иметь общие области виртуальной памяти и разделять содержащиеся в них данные.

Единицей разделяемой памяти является сегмент, который может быть создан одним из асинхронно взаимодействующих процессов.

Такой сегмент продолжает существовать, пока один из процессов не удалит его или ядро ОС не будет перезапущено.

Создаваемый сегмент памяти имеет атрибуты идентификации (адресации), которые являются общими для всех средств пакета IPC, а также индивидуальные права доступа, которые обеспечиваются каждым процессом, подключающим этот сегмент.

Общий список системных вызовов, обеспечивающих работу с разделяемыми сегментами памяти, имеет вид:

1. Создание нового или получение идентификатора уже существующего разделяемого сегмента памяти осуществляется системным вызовом shmget(...).
2. Уже созданный сегмент разделяемой памяти подстыковывается к адресному пространству процесса с помощью системного вызова \*shmat
3. Отсоединение, ранее подключенного сегмента разделяемой памяти, осущест вляется системным вызовом: shmdt
4. Общее управление сегментами разделяемой памяти, с использованием структуры типа shmid\_ds, осуществляется посредством системного вызова: shmctl

* **Задача о читателях и писателях.**

Аппарат разделяемых сегментов памяти предоставляет нескольким процессам возможность одновременного доступа к общей области памяти.

В качестве учебного примера, демонстрирующего совместное использование синхронизации и разделяемые сегменты памяти, рассмотрим задачу «Читатели- писатели». Общая интерпретация этой задачи — следующая.

Писатель, владея публичным ресурсом, периодически пишет на нем книги

Читатели обращаются к публичному ресурсу писателя.

Общие требования:

• процессы-читатели имеют одновременный доступ на операцию чтения, но обязаны ждать, пока процесс-писатель не закончит свою работу;

• процесс-писатель должен дождаться завершения процесса чтения читателей, но не допускает к чтению процессы, которые пришли во время его написания новой книги.

* **Передача сообщений.**

Третьим и последним средством, входящим в System V IPC, являются очереди сообщений.

Очереди сообщений - это наиболее семантически нагруженный способ взаимодействия процессов через каналы связи, в котором на передаваемую информацию накладывается определенная структура, так что процесс, принимающий данные, может четко определить, где заканчивается одна порция информации и начинается другая.

Такая модель позволяет задействовать один и тот же канал связи для передачи данных в двух направлениях между несколькими процессами.

Очереди сообщений, как семафоры и разделяемая память, являются средством связи с непрямой адресацией, что требует:

• инициализации их, для организации взаимодействия процессов;

• специальных действий, для освобождения системных ресурсов по окончании взаимодействия.

Общий набор примитивов передачи сообщений представлен четырьмя системными вызовами:

1. Системный вызов msgget() предназначен для выполнения операции доступа к очереди сообщений.
2. Системный вызов msgsnd() предназначен для помещения сообщения в очередь сообщений.
3. Системный вызов msgrcv() предназначен для получения сообщения из очереди сообщений.
4. Системный вызов msgctl() предназначен для получения информации об очереди сообщений, изменения ее атрибутов и удаления из системы.

**Тема 12**

* **Графические среды ОС.**

X Window - сложная система, основой которой является программа X-сервер, которая через драйверы устройств взаимодействует с видеокартой ЭВМ, клавиатурой, мышью и монитором компьютера.

Именно Х-сервер:

• устанавливает и переключает графические режимы видеокарты ЭВМ;

• рисует элементы изображений;

• определяет координаты мыши;

• формирует программные прерывания при нажатии кнопок мыши и клавиатуры.

Все остальные программы, включая менеджер окон, взаимодействуют с Х- сервером по особому протоколу, который называется X-протокол, или протокол сетевой связи (X Network Protocol).

Х-сервер обрабатывает 4 типа сообщений:

• Запрос – клиент требует нарисовать что-либо в окне или запрашивает у сервера информацию;

• Ответ – сервер отвечает на запрос;

• Событие – сервер сообщает клиенту о событии, например, о нажатии клавиши пользователем;

• Ошибка – сервер сообщает об ошибке в запросе клиента.

* **Рабочий стол пользователя.**

Оконный менеджер - это приложение, которое отвечает за размещение, декорирование окон и за взаимодействие между окнами:

• изменение размеров,

• максимизация,

• свертывание,

• закрытие.

Оконный менеджер взаимодействует с графическим сервером X11, который, в свою очередь, занимается взаимодействием c видеокартой и устройствами ввода/ вывода: клавиатура, мышь и монитор.

Именно оконный менеджер обеспечивает функциональность графической среды пользователя.

Чтобы максимально обеспечить удобство работы пользователя, в каждой ОС, разрабатывается набор приложений, который называются окружение рабочего стола или просто — рабочий стол.

* **Различие графических сред ОС.**

Имеются операционные системы, которые не могут работать без графического окружения.

К таким ОС относятся: MS Windows и MacOS.

Для UNIX и Linux, графическое окружение ОС является необязательным и может быть отключено с целью экономии ресурсов ЭВМ.

В целом, оконные менеджеры могут работать:

• как вместе с окружением рабочего стола;

• так и отдельно от него.

Ряд рабочих столов использует собственные менеджеры окон, например:

• Xfce — xfwm;

• KDE — Kwin;

• Gnome — Metacity.

* **X-сервер UNIX.**

Важной особенностью Х-сервера является возможность его работать на стеке протоколов TCP/IP с программами, запущенными на удаленных компьютерах.

Обычно, для этих целей используется асинхронная связь (протокол UDP), но возможна и синхронная связь по протоколу TCP, которая работает в 30 раз медленнее, чем асинхронная связь по UDP.

Кроме того, на компьютере может быть запущено несколько Х-серверов, которые выводят графическую информацию на разные дисплеи с разными номерами: 0, 1, 3 и так далее**.**

X-сервер отображает не все программы, пытающиеся к нему подключиться, а только те, которым разрешен доступ. Чтобы разрешить Х-серверу подключение программ, используется утилита xhost.

* **Архитектура шины D-Bus.**

Системы IPC для Unix/Linux обладают развитыми средствами IPC низкого уровня: сокеты, каналы, семафоры, сегменты разделяемой памяти, передача сообщений.

D-Bus – еще одна система межпроцессного взаимодействия (Interprocess Communication или IPC). Суть проблемы состоит в том, что приложения одного рабочего стола должны тесно взаимодействовать между собой.

D-Bus - это система межпроцессного взаимодействия, предоставляющая приложениям несколько шин для передачи сообщений и обеспечивающая беспроблемную связь десктопных приложений. Причем, поддерживается не только широковещательная рассылка сообщений (сигналов), но и удаленный вызов методов.

Некоторые из основных понятий шины D-Bus:

• Системная шина (System bus) — данная шина создается при старте сервера D-Bus. На этой шине работают сервисы, которые нельзя отнести к какой-то определенной пользовательской сессии, и которые относятся к системе в целом.

• Сессионная шина (Session bus) — данная шина создается на каждый вход (login) пользователя в систему. К данной шине подключаются приложения пользователя, прошедшего процедуру login. Также через нее проходит общение программ, запущенных в данной рабочей сессии пользователя.

• Имя на шине (Bus name). Каждая программа, подключенная к шине, получает свое уникальное имя, которое начинается с двоеточия (":") и представлено двумя числами, разделенными точками. Например, «:1.5».

* **Библиотека libdbus.**

Низкоуровневое взаимодействие приложений через шину D-Bus обеспечивается с помощью библиотеки libdbus.

В настоящее время, проект D-Bus реализован как версия 1.х.

Поэтому, для его идентификации используется суффикс -1.

Например, для ОС УПК АСУ:

• /etc/dbus-1 — директория с файлами конфигурации D-Bus;

• /lib/libdbus-1.so — ссылка на библиотеку libdbus-1.so.3.14.8

* **Проекции ПО D-Bus на языки программирования.**

Для эффективного использования возможностей шины применяются проекции libdbus, реализованные на различные языки.

Широко известны следующие проекции:

• GLib API — библиотеки проекта GNU

• Python API — который широко используется во всех дистрибутивах Linux

• Qt API — которая составляет постоянную альтернативу библиотекам Glib API

Для доступа к шине D-Bus из языка shell используется набор утилит:

dbus-cleanup-sockets - используется для очистки директорий от остатков сокетов;

dbus-daemon - является демоном шины сообщений D-BUS;

dbus-launch - используется для запуска dbus-daemon из скрипта командной оболочки. Как правило, вызывается из скриптов, регистрирующих вход пользователей в систему;

dbus-monitor - используется для мониторинга сообщений, поступающих через шину сообщений D-BUS;

dbus-send - используется для отправки сообщения в шину сообщений D-BUS;