МИНИСТЕРСТВО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

**Московский государственный университет
экономики, статистики и информатики
Институт "Московская высшая банковская школа"**

Кондратьев В.К.
Головина О.С.

ОПЕРАЦИОННЫЕ СИСТЕМЫ И ОБОЛОЧКИ

Москва 2002 г.

УДК 681.3.06

Головина О.С., Кондратьев В.К. Операционные системы и оболочки. /Моск. гос. ун-т экономики, статистики и информатики. – М., 2002. – xx с.

Книга содержит сведения для первоначального изучения с операционных системам, сред и оболочек.

Книга знакомит с понятиями системного программного обеспечения, с операционной системы и среды, вычислительного процесса и ресурса, процесса и потока, механизмом прерываний.

Книга содержит начальные сведения об управлении задачами и памятью в операционных системах, об управлении вводом/выводом и файловых системах, основы архитектур операционных систем.

Книга приводит сравнительные характеристики операционных систем семейства Windows.

Книга может служить основой для первоначального изучения широко распространенной операционной системы UNIX.

В книге рассматриваются вопросы практического использования операционной системы UNIX, дается объяснение основных понятий: ядро системы, оболочка, процесс, файл, файловая система. Приводится иллюстрация работы основных механизмов управления системой с помощью команд ОС UNIX.

Каждая глава завершается контрольными вопросами.

Учебное пособие предназначено для изучения дисциплин "Операционные системы и оболочки" и "Операционные системы, среды и оболочки". И может быть рекомендована студентам младших курсов, обучающихся по специальности 351400 «Информатика по отраслям» или по специальности 351500 "Математическое обеспечение и администрация информационных систем".

Оглавление

1. Операционные системы и среды

1.1 Введение

Системное программное обеспечение – это программы и комплексы программ, общие для всех, кто совместно использует технические средства компьютера, и применяемые для автоматизации разработки новых программ и выполнения программ существующих.

Системное программное обеспечение состоит из 5-ти групп:

1) операционные системы;
2) системы управления файлами;
3) интерфейсные оболочки для взаимодействия пользователя с ОС и программные среды;
4) системы программирования;
5) утилиты.

ОС – комплекс управляющих и обрабатывающих программ, интерфейс между аппаратурой компьютера и пользователем с его задачами, предназначен для эффективного использования ресурсов вычислительной системы.

Функции ОС:

- прием от пользователя заданий или команд, выданных в виде командной строки или с помощью манипулятора (мыши);
- прием и исполнение программных запросов на запуск, приостановку или остановку других программ;
- загрузка в оперативную память подлежащих исполнению программ;
- инициирование программы (передачи управления на ее выполнение);
- идентификация всех программ и данных;
- обеспечение работы систем управления файлами (СУФ) и/или систем управления базами данных (СУБД);
- обеспечение режима мультипрограммирования, выполнение двух и более задач на одном процессоре;
- организация и управление операциями ввода/вывода;
- обеспечение минимального времени ответа в системах реального времени;
- распределение памяти, организация виртуальной памяти;
- планирование и диспетчеризация заданий в соответствии с заданной дисциплиной обслуживания;
- обмен сообщениями и данными между выполняющимися программами;
- защита одной программы от влияния другой, сохранность данных;
- предоставление услуг на случай частичного сбоя системы;
- обеспечение работы систем программирования.

Система управления файлами предназначена для организации более удобного доступа к данным, организованным в файлы. Все современные ОС имеют соответствующие системы управления файлами, однако, ряд ОС позволяют работать с несколькими файловыми системами (даже одновременно). Эта возможность обеспечивается монтированием файловых систем.

Интерфейсная оболочка предназначена для удобства взаимодействия пользователя с ОС. Назначение – расширить возможности по управлению ОС или изменить встроенные в систему возможности. Примеры: Explorer, X Window, эмуляторы).

Операционная среда – интерфейс, необходимый программам для обращения к ОС с целью получить определенный сервис.

Система программирования включает следующие элементы:
- транслятор с соответствующего языка;
- библиотеки подпрограмм;
- редакторы;
- компоновщики;
- отладчики.

Самостоятельных (вне ОС) систем программирования не бывает.

Утилиты – это специальные системные программы, с помощью которых можно как обслуживать саму ОС, так всю вычислительную систему:
- подготовка для работы носителей данных;
- перекодировка;
- оптимизация размещения данных на диске;
- разбиение накопителя на магнитных дисках на разделы;
- форматирование;
- архивирование данных.

1.2 Основные понятия

1.2.1 Понятие операционной среды

Назначение операционной системы:
- управление вычислительными процессами в вычислительной системе;
- распределение ресурсов вычислительной системы между различными вычислительными процессами;
- образование программной (операционной) среды, в которой выполняются прикладные программы пользователей.

Программная подсистема, при обращении к которой посредством соответствующих вызовов пользователь получает функции и сервисы, называется **операционной системой**.

Операционная среда – набор функций и сервисов ОС и правила обращения к ним. Операционная среда – набор интерфейсов, необходимый программам и пользователям для обращения к ОС с целью получить определенные сервисы.

Операционная система в общем случае может содержать несколько операционных сред.

Операционная среда может включать несколько интерфейсов: пользовательские и программные.

Операционная среда – системное программное окружение, в котором могут выполняться программы, созданные по правилам работы этой среды.

1.2.2. Понятие вычислительного процесса и ресурса

Последовательный процесс («задача») – выполнение отдельной программы с ее данными на последовательном процессоре. Процессор имеет два аспекта:

- носитель данных;
- исполнитель операций, связанных с обработкой данных.

С процессом связано понятие ресурса. Термин **ресурс** относится к используемым, относительно стабильным и часто недостающим объектам, которые запрашиваются, используются и освобождаются процессами в период их активности.

Ресурсы могут быть:

- разделяемыми;
- неделимыми.

Разделяемые ресурсы могут использоваться:

- одновременно (в один и тот же момент времени);
- параллельно (в течение некоторого отрезка времени процессы используют ресурс попеременно).

Мультипрограммный режим работы вычислительной системы заключается в том, что пока одна программа (процесс, задача) ожидает завершения очередной операции ввода/вывода, другая программа (задача) может быть поставлена на выполнение.

При мультипрограммировании повышается пропускная способность системы, но отдельный процесс никогда не может быть выполнен быстрее, чем, если бы он выполнялся в однопрограммном режиме.

ОС поддерживает мультипрограммирование (многопроцессность) и старается эффективно использовать ресурсы путем организации очередей запросов.

При необходимости использовать какой-либо ресурс процесс обращается к супервизору ОС и сообщает ему свои требования (вид ресурса, объем и т.д.). Эта директива переводит процессор в привилегированный режим, если он есть.

Ресурс будет выделен обратившемуся за ним процессу, если:

- он свободен и нет задач с более высоким приоритетом, обратившимся за этим ресурсом;
- текущий запрос и ранее выданные запросы допускают совместное использование ресурсов;
- ресурс используется задачей с более низким приоритетом и может быть временно отобран.

Если ресурс занят, ОС ставит задачу в очередь к ресурсу, переводя ее в состояние ожидания. Очередь к ресурсу может быть организована различными способами, но обычно с помощью списковой структуры.

После завершения работы с ресурсом задача с помощью системного вызова супервизора сообщает ОС об отказе от ресурса.

Супервизор ОС, получив управление, освобождает ресурс и проверяет, есть ли очередь к этому ресурсу. Если очередь есть, то в зависимости от **дисциплины обслуживания** и приоритетов задач, ожидающих данный ресурс, супервизор выбирает задачу и переводит ее в состояние готовности к выполнению. Управление будет передано либо этой выбранной задаче, либо той, которая только что освободила ресурс.

При выдаче запроса задача может указать, хочет ли она владеть ресурсом монопольно или совместно с другими задачами (файл).

При организации управления ресурсами требуется принять решение о том, что в данной ситуации выгоднее:

- быстро обслуживать отдельные наиболее важные запросы;
- предоставлять всем процессам равные возможности;
- обслуживать максимально возможное количество процессов;
- наиболее полно использовать ресурсы.
-

1.2.3 Динамика состояния процесса

Процесс может находиться:

- в **активном состоянии**, может участвовать в конкуренции за использование ресурсов вычислительной системы;
- в **пассивном состоянии**, известен системе, в конкуренции не участвует.

Активный процесс может находиться в одном из следующих состояний:

- **выполнения**, когда все затребованные ресурсы выделены. В состоянии выполнения может находиться только один процесс (для однопроцессорной системы);
- **готовности к выполнению**, когда ресурсы могут быть предоставлены и процесс перейдет в состояние выполнения;
- **блокирования или ожидания**, когда затребованные ресурсы не могут быть предоставлены или не завершена операция ввода/вывода.

За время существования процесс может неоднократно совершать переходы из одного состояния в другое. Это обусловлено:

- обращениями к операционной системе с запросами ресурсов;
- выполнением системных функций;
- взаимодействием с другими процессами;
- появлением сигналов прерывания от таймера, каналов и устройств ввода/вывода и других устройств.

Процесс из **состояния бездействия** в **состояние готовности** может перейти в следующих случаях:

- по команде оператора (пользователя) в диалоговых ОС, где программа может иметь статус задачи, а не просто быть исполняемым файлом и только на время исполнения получит статус задачи (ОС на ПК);
- при выборе из очереди планировщиком;
- по вызову из другой задачи (посредством обращения к супервизору один процесс может создавать, инициировать, приостановить, остановить или уничтожить другой процесс);
- по прерыванию от внешнего «инициативного» устройства;
- при наступлении запланированного времени запуска программы.

Процесс, который может исполняться, как только ему будет предоставлен процессор, находится в **состоянии готовности**, все ресурсы, за исключением процессора, ему выделены.

Из **состояния выполнения** процесс может выйти по одной из следующих причин:

- процесс завешается, при этом он посредством обращения к супервизору передает управление ОС и сообщает о своем завершении. Супервизор уничтожает процесс или переводит его в список **бездействующих** процессов. В состояние **бездействия** процесс может быть переведен принудительно по

команде оператора или путем обращения к супервизору другой задачи, требующей остановить данный процесс;
- процесс переводится супервизором ОС в состояние **готовности к исполнению** в связи с появлением более приоритетной задачи или в связи с окончанием выделенного ему кванта времени;
- процесс **блокируется** из-за невозможности предоставить ему ресурс или вследствие запроса ввода/вывода, а также по команде оператора на приостановку задачи.

Процесс деблокируется и переводится в состояние готовности к исполнению при наступлении соответствующего события:
- завершение операции ввода/вывода;
- освобождение затребованного ресурса;
- загрузка в оперативную память страницы виртуальной памяти и т.д.

Предпосылками изменения состояния процесса являются события. Один из основных видов событий – прерывания.

1.2.4 Реализация понятия последовательного процесса в ОС

Чтобы ОС могла управлять процессами, она должна располагать полной информацией о них. Для этого на каждый процесс заводится специальная информационная структура, называемая **дескриптором процесса** (описателем задач, блоком управления задачей). В общем случае дескриптор процесса содержит следующую информацию:
- идентификатор процесса (PID);
- тип (класс) процесса, по которому супервизор определяет правила обслуживания;
- приоритет процесса, по которому супервизор предоставляет ресурсы;
- переменную состояния (готов к работе, в состоянии выполнения, ожидание устройства ввода/вывода и т.д.);
- защищенную область памяти, в которой хранятся регистры процессора, если процесс прерывается, не закончив работу. Эта область называется **контекстом задачи**;
- информацию о ресурсах, которыми процесс владеет или имеет право пользоваться;
- место (или адрес) памяти для общения с другими процессами;
- параметры времени запуска;
- в случае отсутствия системы управления файлами – адрес задачи на диске в ее исходном состоянии и адрес на диске, куда она выгружается из оперативной памяти.

Описатели задач, как правило, располагаются в оперативной памяти для ускорения работы супервизора. Для каждого состояния (кроме выполнения) ведется список задач.

1.2.5 Процессы и треды

Для реализации «мультизадачности» было введено понятие «легковесных» процессов, которые в настоящее время получили названия **потоки** или **треды** (нити).

Понятие **процесс** в плоскости ОС подчеркивает их обособленность:

- у каждого процесса свое виртуальное адресное пространство;
- каждому процессу выделяются свои ресурсы – файлы, семафоры и т.д.

Обособленность процессов нужна для защиты процессов друг от друга, т.к. они, используя общие ресурсы вычислительной системы, постоянно конкурируют друг с другом. Процессы не связаны между собой и могут принадлежать различным пользователям, разделяющим одну вычислительную систему. ОС считает процессы несвязанными и независимыми.

В самих процессах также имеется внутренний параллелизм, использование которого позволяет повысить производительность вычислительной системы. Например, некоторые операции, выполняемые приложением, могут требовать большого количества процессорного времени, в этом случае пользователь долго будет ждать результата. Если программные модули, выполняющие такие длительные операции, оформить в виде самостоятельных «подпроцессов» (легковесных или облегченных потоков, тредов, «задач»), то у пользователя появляется возможность параллельно выполнять несколько операций в рамках одного процесса. Для этих задач ОС не создает полноценной виртуальной машины:

- задачи не имеют собственных ресурсов;
- находятся в том же виртуальном адресном пространстве, что и данный процесс;
- могут пользоваться теми же файлами, виртуальными устройствами и прочими ресурсами.

Единственный собственный ресурс для **потоков** – процессорный.

В однопроцессорных системах потоки разделяют между собой процессорное время, а в многопроцессорных – могут выполняться параллельно, если нет конкуренции из-за других ресурсов.

Главный **результат** многопоточности – возможность параллельно выполнять несколько видов операций в одной прикладной программе. Параллельное вычисление (более эффективное использование ресурсов центрального процессора и уменьшение суммарного времени выполнения задачи) реализуется на уровне потоков и программа, оформленная в виде нескольких потоков в рамках одного процесса, выполняется быстрее за счет параллельного выполнения отдельных ее частей.

Особенно эффективно можно использовать многопоточность для выполнения распределенных приложений.

С понятием «поток» связано распределение процессорного времени.

С понятием «процесс» связано распределение всех ресурсов, при диспетчеризации следует учитывать все ресурсы, закрепленные за процессом.

Каждый процесс всегда состоит, по крайней мере, из одного потока, и только при наличии внутреннего параллелизма программист может расщепить один поток на несколько.

1.2.6 Прерывания

Прерывания – механизм, позволяющий координировать параллельное функционирование отдельных устройств вычислительной системы и реагировать на особые ситуации, возникающие при работе процессора. Прерывания – принудительная передача управления от выполняемой программы к системе, происходящее при возникновении определенного события.

Основная цель введения прерываний – реализация асинхронного режима работы и распараллеливание работы отдельных устройств вычислительного комплекса.

Механизм прерываний реализуется аппаратно-программным способом. Прерывание всегда влечет за собой изменение порядка выполнения команд процессором.

Механизм обработки прерываний включает следующие шаги:

1) установление факта прерывания и его идентификация;
2) запоминание состояния прерванного процесса (счетчика команд, содержимого регистров процессора, спецификации режима и др.);
3) аппаратная передача управления подпрограмме обработки прерываний;
4) сохранение информации о прерванной программе, которую не удалось сохранить на шаге 2 с помощью действий аппаратуры, иногда большого объема информации;
5) обработка прерываний;
6) восстановление информации, относящейся к прерванному процессу;
7) возврат в прерванную программу.

Шаги 1 – 3 реализуются аппаратно, а шаги 4 –7 программно.

Главные функции механизма прерываний:

- распознавание или классификация прерываний;
- передача управления на обработку прерываний;
- корректное возвращение к прерванной программе.

Прерывания, возникающие при работе вычислительной системы, можно разделить на два основных класса:

- внешние (асинхронные);
- внутренние (синхронные).

Внешние прерывания вызываются асинхронными событиями, которые происходят вне прерываемого процесса, например:

- прерывания от таймера;
- прерывания от внешнего устройства (прерывания по вводу/выводу);
- прерывания по нарушению питания;
- прерывания с пульта оператора вычислительной системы;
- прерывания от другого процессора или другой вычислительной системы.

Внутренние прерывания вызываются событиями, которые связаны с работой процессора и являются синхронными с его операциями, например:

- нарушение адресации;
- наличие в поле адреса несуществующей инструкции;
- деление на нуль;
- переполнение или исчезновение порядка;
- ошибка четности;
- ошибка в работе различных аппаратных устройств.

Собственно программные прерывания происходят по соответствующей команде прерывания, то есть по этой команде процессор производит те же действия, что и при обычных внутренних прерываниях. Данный механизм введен для того, чтобы переключение на системные программные модули происходило не как переход в подпрограмму, а как обычное прерывание. Этим обеспечивается автоматическое переключение процессора в привилегированный режим с возможностью выполнения всех команд.

Сигналы, вызывающие прерывания, формируются в процессоре или вне его, они могут возникнуть одновременно. Выбор одного их них происходит на основе приоритетов, установленных для каждого их них. Наивысшим приоритетом обладают прерывания от схем контроля процессора. Учет приоритетов может быть встроен в

технические средства или может определяться операционной системой. Программно-аппаратное управление порядком обработки сигналов прерывания позволяет применять различные дисциплины обслуживания прерываний.

Распределение прерываний по уровню приоритета (от низкого к высокому):

- программные прерывания;
- прерывания от внешних устройств: терминалов;
- прерывания от внешних устройств: сетевого оборудования;
- прерывания от внешних устройств: магнитных дисков;
- прерывания от системного таймера;
- прерывания от средств контроля процессора.

Процессор может обладать средствами защиты от прерываний:

- отключение системы прерываний;
- маскирование (запрет) отдельных видов прерываний.

Программное управление средствами защиты от прерываний позволяет ОС регулировать обработку сигналов прерывания:

- обрабатывать сразу при поступлении;
- откладывать обработку на некоторое время;
- полностью игнорировать.

Обычно операция прерывания выполняется только после завершения выполнения текущей команды.

Сигналы прерывания возникают в произвольные моменты времени, поэтому к моменту обработки может накопиться несколько сигналов. Сигналам прерывания присваиваются приоритеты, в первую очередь обрабатывается сигнал с более высоким приоритетом.

Программное управление специальными регистрами маски (маскирование сигналов прерывания) позволяет реализовать различные дисциплины обслуживания:

- с **относительными приоритетами**, обслуживание не прерывается даже при наличии запросов с более высоким приоритетом. В программе обслуживания данного запроса следует наложить маски на все остальные сигналы прерывания или просто отключить систему прерываний;
- с **абсолютными приоритетами**, обслуживается прерывание с наибольшим приоритетом. В программе обслуживания прерываний следует наложить маски на сигналы прерывания с более низким приоритетом. Возможно многоуровневое прерывание, то есть прерывание программы обработки прерывания, число уровней меняется и зависит от приоритета запроса;
- по **принципу стека** (последним пришел – первым обслужен), запросы с более низким приоритетом могут прервать обработку прерывания с более высоким приоритетом. В программе обслуживания прерываний не следует накладывать маски ни на один сигнал прерывания и отключать систему прерываний.

Управление ходом выполнения задач со стороны ОС заключается:

- в организации реакций на прерывание;
- в организации обмена информацией;
- в предоставлении необходимых ресурсов;
- в динамике выполнения задачи;
- в организации сервиса.

Причины прерываний определяет ОС (супервизор прерываний) и выполняет действия, необходимые при данном прерывании и в данной ситуации.

При появлении запроса на прерывание система прерываний идентифицирует сигнал и, если прерывание разрешено, управление передается на соответствующую подпрограмму обработки прерываний.

Подпрограмма обработки прерываний состоит их трех секций:

1) отключение прерываний, сохранение контекста прерванной программы, установка режима работы системы прерываний;
2) собственно тело программы обработки прерываний;
3) восстановление контекста прерванной ранее программы, установка прежнего режима работы системы прерываний.

1-я и 3-я секции подпрограммы обработки прерываний – служебные, сохраняют и восстанавливают контекст задач. Поскольку эти действия необходимо выполнять практически в каждой подпрограмме обработки прерывания, во многих ОС первые секции подпрограмм обработки прерываний выделяются в специальный системный модуль – **супервизор прерываний**.

Супервизор прерываний выполняет следующие действия:

- сохраняет в дескрипторе текущей задачи рабочие регистры процессора, определяющие контекст прерванной задачи;
- определяет программу, обслуживающую текущий запрос на прерывание;
- устанавливает необходимый режим обработки пребывания;
- передает управление подпрограмме обработки прерывания.

После выполнения подпрограммы обработки прерывания управление передается супервизору в модуль управления диспетчеризацией задач. Диспетчер задач производит:

- выбор готовой к выполнению задачи (в соответствии с дисциплиной обслуживания)
- восстановление контекста задачи;
- установка прежнего режима работы системы прерываний;
- передачу управления выбранной задаче.

Из подпрограммы обработки прерывания нет возврата непосредственно в прерванную программу. Если бы контекст прерванной задачи сохранялся в стеке, а не в дескрипторе задачи, то не было бы возможности гибко выбирать на обслуживание задачу, после завершения подпрограммы обработки прерывания.

В конкретных процессорах и ОС могут быть изменения и дополнения к рассмотренной дисциплине обслуживания прерываний.

1.2.7 Основные виды ресурсов

Одним из важнейших ресурсов является сам **процессор**, точнее **процессорное время**. Имеется множество методов разделения этого ресурса.

Вторым важным ресурсом является **оперативная память**. В оперативной памяти может располагаться одновременно несколько процессов (точнее фрагментов, участвующих в вычислении), а может вся оперативная память предоставляться процессам попеременно.

В конкретный момент времени процессор при выполнении вычислений обращается к очень небольшому числу ячеек оперативной памяти, поэтому память желательно разделить для возможно большего числа параллельно исполняемых процессов. С другой стороны, чем больше оперативной памяти предоставлено процессу, тем условия для его выполнения. Проблема разделения оперативной памяти между параллельно выполняемыми процессами является наиболее актуальной.

Внешняя память, например магнитный диск, является двумя видами ресурсов:

- собственно память;
- доступ к ней.

Каждый из этих ресурсов может предоставляться независимо друг от друга, но для работы с внешней памятью необходимы оба вида ресурсов:

- собственно память используется одновременно;
- доступ к внешней памяти попеременный.

Если обращение к внешнему устройству использует механизм прямого доступа, то такие устройства разделяются параллельно. Если устройство работает с последовательным доступом, то оно не относится к разделяемым ресурсам, например, принтер или накопитель на магнитной ленте.

Важным видом ресурсов являются **программные модули**. Системные программные модули рассматриваются как ресурсы, которые могут быть разделены между параллельно выполняемыми процессами.

Программные модули могут использоваться:

- однократно;
- многократно.

Однократно исполняемые модули, как правило, могут быть выполнены только один раз, поскольку в процессе своего исполнения они могут:

- повредить часть кода;
- повредить исходные данные, от которых зависит ход вычислений.

Однократно исполняемые программные модули вообще не распределяются как ресурс системы, они, как правило, используются только на этапе загрузки системы.

Повторно используемые программные модули делятся на следующие виды:

- непривилегированные;
- привилегированные;
- реентерабельные.

Привилегированные программные модули работают в привилегированном режиме, при отключенной системе прерываний, т.е. никакие внешние события не могут нарушить естественный порядок вычислений. Привилегированный программный модуль всегда выполняется до конца и представляет собой попеременно разделяемый ресурс.

Структура привилегированного программного модуля включает следующие секции:

- отключение прерываний;
- собственно тело программного модуля;
- включение прерываний.

Непривилегированные программные модули – это обычные программные модули, которые могут быть прерваны во время своей работы. В общем случае их нельзя считать разделяемыми, потому что если его прервать в рамках одного процесса и запустить еще раз в рамках другого процесса, то промежуточные результаты для первого процесса могут быть потеряны.

Реентерабельные программные модули допускают повторное многократное прерывание своего исполнения и повторный их запуск при обращении из других задач, т.е. реентерабельные программные модули должны сохранять промежуточные значения для прерываемых вычислений и их восстановление, когда вычислительный процесс возобновляется с прерванной точки. Это можно реализовать двумя способами:

- с помощью статических методов выделения памяти под сохраняемые значения;
- с помощью динамических методов выделения памяти под сохраняемые значения. Этот метод используется чаще.

Реентерабельный программный модуль делится на следующие секции:

- привилегированный модуль, заказывающий в системной области памяти блок ячеек для хранения текущих (промежуточных) данных;
- основное тело реентерабельного модуля, которое и может быть прервано. Работает в непривилегированном режиме;

- привилегированный модуль, освобождающий в системной области памяти блок памяти, использованный для хранения промежуточных результатов.

При помещении всех промежуточных данных в системную область, на вершину стека помещается указатель на начало области данных и ее объем. Во время исполнения центральной секции реентерабельного программного модуля возможно ее прерывание. Если прерывание не возникло, то в последней секции производится освобождение использованного блока системной области памяти. Если во время исполнения центральной части произошло прерывание и другой вычислительный процесс обращается к тому же реентерабельному модулю, то для этого нового процесса выделяется новый блок памяти в системной области и на вершину стека записывается новый указатель. Повторное вхождение возможно, пока не израсходуется область системной памяти, выделенной специально для реентерабельной обработки.

При статическом способе выделения памяти резервируется область памяти для фиксированного числа вычислительных процессов, в которых будут располагаться переменные реентерабельных программных модулей, для каждого процесса своя область памяти. К таким процессам относятся драйверы ввода/вывода.

Кроме реентерабельных программных модулей еще имеются **повторно входимые модули**. Повторно входимые программные модули допускают многократное параллельное исполнение, но их нельзя прерывать. Повторно входимые программные модули состоят из привилегированных секций и повторное обращение к ним возможно только при завершении работы какой-либо секции. После выполнения какой-либо секции управление передается супервизору, который определит, какой процесс будет использовать этот модуль и с какой точки. В повторно входимых программных модулях определены все допустимые (возможные) точки входа. Повторно входимые модули встречают гораздо чаще, чем реентерабельные.

К ресурсам относятся также информационные ресурсы, т.е. данные. Информационные ресурсы включают в себя:

- переменные, находящиеся в оперативной памяти;
- файлы.

Если процессы используют данные только для чтения, то такие информационные ресурсы можно разделять.

Если процессы могут изменять данные, то работы с такими данными должна быть организована специальным образом.

1.3. Классификация операционных систем

Вариантов классификации ОС может быт очень много, они зависят от признака, по которому одна ОС отличается от другой:

- по назначению;
- по режиму обработки;
- по способу взаимодействия с системой;
- по способу построения.

Основным предназначением ОС является:

- **организация эффективных и надежных вычислений;**
- **создание различных интерфейсов для взаимодействия с этими вычислениями и самой вычислительной системой.**

ОС разделяют **по назначению**:

- ОС общего назначения;
- ОС специально назначения.

ОС специального назначения подразделяются на следующие:

- для переносимых компьютеров и встроенных систем;
- для организации и ведения баз данных;
- для решения задач реального времени и т.д.

ОС разделяют **по режиму обработки задач**:

- однопрограммный режим;
- мультипрограммный режим.

Мультипрограммирование – способ организации вычислений, когда на однопроцессной вычислительной системе создается видимость одновременного выполнения нескольких задач. Любая задержка в выполнении одной программы используется для выполнения других программ.

Мультипрограммный и многозадачный режимы близки по смыслу, но синонимами не являются.

Мультипрограммный режим обеспечивает параллельное выполнение нескольких приложений, а программисты, создающие эти приложения, не должны заботиться о механизме организации их параллельной работы. Эти функции выполняет ОС, которая распределяет между выполняющимися приложениями ресурсы вычислительной системы, обеспечивает необходимую синхронизацию вычислений и взаимодействие.

Мультизадачный режим предполагает, что забота о параллельном выполнении и взаимодействии приложений ложится на прикладных программистов.

Современные ОС для ПК реализуют и мультипрограммный, и многозадачный режимы.

По организации работы в **диалоговом режиме** ОС делятся на следующие:

- однопользовательские (однотерминальные);
- мультитерминальные.

В мультитерминальных ОС с одной вычислительной системой одновременно могут работать несколько пользователей, каждый со своего терминала, при этом у пользователей возникает иллюзия, что у него имеется своя собственная вычислительная система. Для организации мультитерминального доступа необходим мультипрограммный режим работы вычислительной системы.

Основная особенность **операционных систем реального времени (ОСРВ)** – обеспечение обработки поступающих заданий в течение заданных интервалов времени, которые нельзя превышать. Поток заданий не является планомерным и не регулируется оператором, т.е. задания поступают в непредсказуемые моменты времени и без всякой очередности. В ОСРВ в общем случае отсутствуют накладные расходы процессорного времени на этап инициирования (загрузку программы, выделение ресурсов), так как набор задач обычно фиксирован и вся информация о задаче известна до поступления запроса. Для реализации режима реального времени необходим режим мультипрограммирования, который является основным средством повышения производительности вычислительной системы, а для задач реального времени производительность – решающий фактор. Лучшие по производительности характеристики для систем реального времени обеспечивают однотерминальные ОСРВ.

По **способам построения (архитектуре)** ОС подразделяются на следующие:

- микроядерные;
- монолитные.

Это деление условно. К микроядерным ОС относится ОСРВ QNX, а к монолитным – Windows 9x и Linux. Для ОС Windows 9x пользователь не может изменить ядро, так как не располагает исходными кодами и программой сборки ядра. Для ОС Linux такая возможность предоставлена, пользователь может сам собрать ядро, включив в него необходимые программные модули и драйверы.

1.4. Вопросы к главе 1

1) В чем заключается различие понятий процесс и задача?
2) Для чего каждая задача получает дескриптор? Какие поля содержатся в дескрипторе? Что такое контекст задачи?
3) Объясните понятие ресурса. Почему понятие ресурса является одним из фундаментальных пи рассмотрении ОС? Какие виды и типы ресурсов вы знаете?
4) Сколько и каких списков дескрипторов задач может быть в системе? От чего зависит это число?
5) Перечислите дисциплины обслуживания прерываний, как можно реализовать каждую из этих дисциплин?
6) С какой целью в ОС вводится специальный программный модуль, называемый супервизором прерываний?
7) В чем заключается различие между повторно входимыми и повторно прерываемыми программными модулями? Как они реализуются?
8) Что такое привилегированный программный модуль? Почему нельзя создать ОС, в которой не было бы привилегированных программных модулей?

2. Управление задачами и памятью в операционных системах

Оперативная память – это важнейший ресурс любой вычислительной системы, поскольку без нее, как и без центрального процессора, невозможно выполнение ни одной программы. Память является разделяемым ресурсом. Способы разделения памяти и времени центрального процессора сильно влияют на скорость выполнения отдельных вычислений и на общую эффективность вычислительной системы.

ОС выполняет следующие основные функции, связанные с управлением задачами:

- создание и удаление задач;
- планирование процессов и диспетчеризация задач;
- синхронизация задач, обеспечение их средствами коммуникации.

Система управления задачами обеспечивает похождение их через компьютер. В зависимости от состояния процесса ему должен быть предоставлен тот или иной ресурс.

Создание и удаление задач производится по соответствующим запросам от пользователей или самих задач.

Основным подходом к организации того или иного метода управления процессами является организация очередей процессов и ресурсов.

На распределение ресурсов влияют конкретные потребности тех задач, которые должны выполняться параллельно.

Задачи динамического планирования, т.е. наиболее эффективного распределения ресурсов, возникающие практически при каждом событии, называются **диспетчеризацией**. Планирование осуществляется реже, чем задача текущего распределения ресурсов между уже выполняющимися процессами и потоками. Различие между долгосрочным и краткосрочным планированием заключается в частоте запуска.

Долгосрочный планировщик решает, какой из процессов, находящихся во входной очереди, должен быть переведен в очередь готовых к выполнению процессов в случае освобождения ресурсов памяти. В очереди готовых к выполнению процессов должны находиться в равной пропорции процессы, ориентированные на ввод/вывод, и процессы, ориентированные на работу центрального процессора.

Краткосрочный планировщик решает, какая из задач, находящихся в очереди готовых к выполнению, должна быть передана на выполнение. В большинстве современных ОС долгосрочный планировщик отсутствует.

2.1. Планирование и диспетчеризация процессов и задач

2.1.1. Стратегия планирования

Стратегия планирования (краткосрочное планирование, диспетчеризация) определяет, какие процессы планируются на выполнение для того, чтобы достигнуть поставленной цели. Стратегий планирования много, но основные из них следующие:

- по возможности заканчивать вычисления в том же порядке, в котором он были начаты;
- отдавать предпочтение более коротким задачам;
- предоставлять всем пользователям одинаковые услуги, в том числе и одинаковое время ожидания.

Стратегия планирования связана с понятием процесс, а не задача, так как процесс может состоять из нескольких задач (потоков).

2.1.2. Дисциплины диспетчеризации

Диспетчеризация связана с понятием задачи (потока). Если ОС не поддерживает механизма потоков, то понятие задачи можно заменить на понятие процесса.

Известно большое количество правил, в соответствии с которыми формируется очередь (список) готовых к выполнению задач. Имеются два больших класса дисциплин обслуживания:

- бесприоритетные:
- приоритетные.

При бесприоритетном обслуживании выбор задачи производится в некотором порядке без учета их важности и времени обслуживания.

При реализации приоритетных дисциплин обслуживания отдельным задачам предоставляется преимущественное право на исполнение.

Бесприоритетные дисциплины обслуживания делятся на следующие:

- линейные:
  - в порядке очереди;
  - случайный выбор процесса;
- циклический:
  - циклический алгоритм;
  - многоприоритетный циклический алгоритм.

Приоритетные дисциплины обслуживания делятся на следующие:

- с фиксированным приоритетом:
  - с относительным приоритетом;
  - с абсолютным приоритетом;
  - адаптивное обслуживание;
  - приоритет зависит от времени ожидания;
- с динамическим приоритетом:
  - приоритет зависит от времени ожидания;
  - приоритет зависит от времени обслуживания.

Свойства приоритетов:

- приоритет, присвоенный задаче, может являться величиной постоянной;

- приоритет задачи может изменяться в процессе ее решения.

Диспетчеризация с динамическими приоритетами требует дополнительных расходов на вычисление значений приоритетов, поэтому многие ОС реального времени используют методы диспетчеризации на основе статических (постоянных) приоритетов.

Самой простой в реализации является **дисциплина FCFS (first come – first served)**, задачи обслуживаются в порядке очереди, т.е. в порядке их появления. Задачи, приостановленные для ожидания какого-либо ресурса, после перехода в состояние готовности становятся в эту очередь перед задачами, которые еще не выполнялись. Образуются две очереди:

- новые задачи;
- ранее выполнявшиеся, но попавшие в состояние ожидания.

Дисциплина FCFS реализует стратегию обслуживания «по возможности заканчивать вычисления в порядке их появления». Эта дисциплина не требует внешнего вмешательства в ход вычислений и перераспределения процессорного времени. По классу диспетчеризации (вытесняющие и не вытесняющие) дисциплина FCFS относится к не вытесняющим. Достоинства дисциплины FCFS:

- простота реализации;
- малые расходы системных ресурсов на формирование очереди задач.

Основной недостаток – при увеличении загрузки вычислительной системы растет среднее время ожидания обслуживания, короткие задачи ожидают столько же времени, как и трудоемкие.

Дисциплина обслуживания SJN (shortest job next) требует, чтобы для каждого задания была известна оценка в потребностях процессорного времени. Пользователи должны были указывать предположительное время выполнения. Диспетчер задач сравнивал указанное время с реальным временем выполнения и, если время выполнения превышало указанное, то помещал это задание в конец очереди.

Дисциплина обслуживания SJN предполагает, что имеется только одна очередь заданий, готовых к выполнению. Если задание было временно заблокировано из-за занятости какого-либо ресурса, то оно помещается в конец очереди готовых к выполнению заданий наравне с вновь поступившими. Задания, которым требуется совсем немного времени для завершения, попадают в конец очереди.

Для устранения этого недостатка была предложена **дисциплина SRT (shortest remaining time**, следующее задание требует меньше всего времени для своего завершения).

Перечисленные три дисциплины обслуживания могут использоваться для пакетных режимов работы, когда не важно время отклика.

Для интерактивной работы надо обеспечить приемлемое время реакции системы и равенство в обслуживании, если система мультитерминальная. Интерактивные задания должны иметь преимущество перед фоновыми. Эти условия решены в дисциплине RR (round robin – круговая, карусельная).

Дисциплина обслуживания RR предполагает, что каждая задача получает процессорное время порциями (квантами). После окончания выделенного кванта времени задача снимается с исполнения и на выполнение выбирается следующая задача. Снятая задача помещается в конец очереди готовых к выполнению задач.

Величина кванта времени выбирается как компромисс между приемлемым временем реакции системы на запросы пользователей и накладными расходами на частоту смены контекста задач.

2.1.3. Вытесняющие и не вытесняющие алгоритмы диспетчеризации

Диспетчеризация без перераспределения процессорного времени, то есть **не вытесняющая многозадачность** – это такой способ диспетчеризации процессов, при котором активный процесс выполняется до тех пор, пока он сам, по своей инициативе, не отдаст управление диспетчеру задач для выбора из очереди другого, готового к исполнению процесса. Дисциплины обслуживания FCFS, SJN, SRT относятся к не вытесняющим.

Диспетчеризация с перераспределением процессорного времени между задачами, то есть вытесняющая многозадачность – это такой способ, при котором решение о переключении процессора с выполнения одного процесса на выполнение другого процесса принимается диспетчером задач, а не самой активной задачей. Механизм диспетчеризации сосредоточен в самой ОС и программист не должен заботиться о параллельном выполнении своего приложения с другими приложениями. Операционная система выполняет следующие функции:

- определяет момент снятия с выполнения текущей задачи;
- сохраняет контекст текущей задачи в дескрипторе задачи;
- выбирает из очереди готовых к выполнению задач следующую;
- загружает контекст выбранной задачи;
- запускает выбранную задачу на исполнение.

Дисциплина RR и аналогичные ей относятся к вытесняющим.

При не вытесняющей многозадачности механизм распределения процессорного времени распределен между ОС и прикладной программой. Прикладная программа должна быть разделена на кванты, по окончанию которых с помощью системного вызова управление передается супервизору ОС. Диспетчер задач формирует очереди и выбирает задачу на исполнение.

2.1.4. Качество диспетчеризации и гарантии обслуживания

Одна из проблем при выборе дисциплины обслуживания – гарантия обслуживания. При некоторых дисциплинах обслуживания, например, с абсолютными приоритетами, низкоприоритетные задачи долго могут не получать процессорное время.

Требование к системе – не только завершить процесс, но завершить его к указанному времени или в течение указанного времени.

Наиболее рациональное решение – выделять процессорное время квантами.

Гарантировать обслуживание можно тремя способами:

- выделять минимальную долю процессорного времени некоторому классу процессов, если, по крайней мере, один из них готов к исполнению;
- выделять минимальную долю процессорного времени некоторому конкретному процессу, готовому к исполнению;
- выделить столько процессорного времени некоторому процессу, чтобы он мог выполнить свои вычисления к сроку.

Для сравнения алгоритмов диспетчеризации используются следующие критерии:

- использование (загруженность) центрального процессора;
- пропускная способность – количество процессов, выполняющихся в единицу времени;
- время оборота – интервал времени от момента появления процесса во входной очереди до момента его завершения (время ожидания во входной очереди + время ожидания в очереди готовых к выполнению процессов + время

ожидания в очередях к оборудованию + время выполнения в процессоре + время ввода/вывода);

- время ожидания – суммарное время нахождения процесса в очереди готовых к выполнению процессов;
- время отклика – время от момента попадания процесса во входную очередь до момента первого обращения к терминалу.

Главные причины уменьшения производительности системы:

- накладные расходы на переключение процессора (переключения контекстов задач, перемещения страниц виртуальной памяти, обновление данных в кэш-области);
- переключение на другой процесс в тот момент, когда текущий процесс выполняет критическую секцию, а другие процессы активно ожидают входа в свою критическую секцию.

Методы повышения производительности системы в мультипроцессорных системах:

- совместное планирование, все потоки одного приложения одновременно выбираются для выполнения процессорами и одновременно снимаются с них;
- находящиеся в критической секции задачи не прерываются, а активно ожидающие входа в критическую секцию задачи не выбираются до тех пор, пока вход в секцию не освободится;
- планирование с учетом «советов» программы.

2.1.5. Диспетчеризация задач с использованием динамических приоритетов

При выполнении программ может случиться ситуация, когда одна или несколько задач не могут быть выполнены в течение значительного времени из-за высокой нагрузки в вычислительной системе. Введение механизма динамических приоритетов позволяет реализовать быстрое выполнение коротких задач и гарантировать выполнение любых запросов. Эта дисциплина используется в ОС UNIX.

Каждый процесс имеет два атрибута приоритета, с учетом которого распределяется процессорное время между исполняющимися задачами:

- текущий приоритет, на основе которого осуществляется планирование;
- заказанный относительный приоритет (nice number).

Более высокому значению текущего приоритета может соответствовать более низкий фактический приоритет планирования.

Рассмотрим частный случай, когда текущий приоритет процесса варьируется в диапазоне от 0 (низкий приоритет) до 127 (высокий приоритет). Процессы, выполняющиеся в режиме задачи, имеют более низкий приоритет (0 – 65), чем в режиме ядра (66 – 95, системный диапазон). Приоритеты в диапазоне 96 – 127 относятся к процессам с фиксированным приоритетом.

Процессу, ожидающему недоступный в данный момент ресурс, система присваивает приоритет сна (sleep) из диапазона системных приоритетов и связанное с событием, вызвавшим это состояние. Когда процесс пробуждается, ему присваивается приоритет, равный приоритету сна, который находится в системной области, поэтому вероятность выбора такого процесса на выполнение велика. Такой подход позволяет быстро завершить системный вызов.

После завершения системного вызова восстанавливается приоритет режима задачи, сохраненный перед выполнением системного вызова. Это может привести к снижению приоритета и переключению контекста.

Для принятия решения о выборе следующего запускаемого процесса планировщику необходима информация об использовании процессора. Эта составляющая приоритета

уменьшается обработчиком прерываний по таймеру по каждому тику таймера. Когда процесс выполняется в режиме задачи, его текущий приоритет линейно уменьшается.

Каждую секунду процессор пересчитывает приоритеты процессов, готовых к выполнению, что приводит к перемещению процессов в более приоритетные очереди и повышает вероятность их последующего запуска.

Данный алгоритм планирования обеспечивает:

- интересы низкоприоритетных процессов, так как в результате длительного ожидания их приоритет и вероятность выполнения увеличиваются;
- более вероятный выбор интерактивных процессов по сравнению с вычислительными.

2.2. Память и отображение, виртуальное адресное пространство

Программист обращается к памяти с помощью некоторого набора логических имен. Имена переменных и входных точек модулей составляют область имен.

Физическая память представляет собой множество ячеек, которые пронумерованы, к каждой ячейке можно обратиться, указав ее порядковый номер (адрес). Количество ячеек физической памяти ограничено и фиксировано.

Системное программное обеспечение должно связать каждое указанное пользователем имя с физической ячейкой памяти, т.е. осуществить отображение пространства имен на физическую память компьютера. Это происходит в два этапа;

- посредством системы программирования;
- посредством операционной системы (с помощью специальных программных модулей управления памятью и использования соответствующих аппаратных средств вычислительной системы).

Между этими этапами обращение к памяти имеет форму **виртуального** или логического адреса. Множество всех допустимых значений виртуального адреса для некоторой программы определяет ее виртуальное адресное пространство или виртуальную память.

Виртуальное адресное пространство **зависит от**:

- архитектуры процессора;
- системы программирования.

Виртуальное адресное пространство **не зависит от**:

- объема реальной физической памяти, установленной в компьютере.

Адреса команд и переменных в готовой машинной программе, подготовленной к выполнению системой программирования, как раз и являются **виртуальными адресами**.

В результате работы системы программирования полученные виртуальные адреса могут иметь как двоичную, так и символьно-двоичную форму, т.е. привязка к физическим адресам производится на этапе загрузки программы в память перед ее непосредственным выполнением.

Если система программирования генерирует **абсолютную двоичную программу**, то виртуальные адреса точно соответствуют физическим. Часть программ любой ОС должны быть абсолютными двоичными программами, размещаться по фиксированным физическим адресам и обеспечивать размещение остальных программ на различных физических адресах.

В простейших компьютерных системах используется тождественность виртуального адресного пространства исходному пространству имен. Отображение выполняется самой ОС, которая во время исполнения использует таблицу символьных

имен. Данная схема используется в трансляторах, в которых стадии исполнения и трансляции практически неразличимы.

В простейшем случае транслятор-компилятор генерирует относительные адреса, которые являются виртуальными и впоследствии настраиваются на один из непрерывных разделов. Второе отображение осуществляется перемещающим загрузчиком. После загрузки виртуальный адрес теряется, и доступ выполняется непосредственно к физическим адресам.

Термин **виртуальная память** фактически относится к системам, которые сохраняют виртуальные адреса во время исполнения. Второе отображение осуществляется в процессе выполнения задачи, поэтому адреса физических ячеек могут изменяться.

Простое непрерывное распределение – это самая простая схема, согласно которой вся память условно может быть разделена на три части:

- область, занимаемая операционной системой;
- область, в которой размещается исполняемая задача;
- незанятая ничем (свободная) область памяти.

Эта схема предполагает, что ОС не поддерживает мультипрограммирования, поэтому не возникает проблемы распределения памяти между несколькими задачами. Чтобы предоставить задачам максимальный объем памяти, ОС строится таким образом, чтобы постоянно в памяти располагалась только самая нужная ее часть – ядро ОС, остальные модули загружаются при необходимости.

Эта схема влечет два вида потерь:

- потери процессорного времени из-за простоя в связи с вводом/выводом;
- потери самой оперативной памяти, так как она не всегда используется полностью.

Если программа должна будет использовать логическое (и виртуальное) адресное пространство, которое превышает свободную область памяти, или больше всей памяти компьютера, то используется **распределение с перекрытием**, **оверлейная структура**. Этот метод предполагает, что вся программа может быть разбита на части – сегменты. Каждая оверлейная программа имеет одну главную часть (main) и несколько сегментов (segment), причем, в памяти одновременно могут находиться только главная часть и один или несколько сегментов.

Для организации мультипрограммного режима необходимо обеспечить одновременное расположение в оперативной памяти нескольких задач (целиком или частично). Самая простая схема распределения памяти между несколькими задачами предполагает, что память, незанятая ядром ОС, может быть разбита на несколько непрерывных частей (зон, разделов). Разделы характеризуются:

- именем;
- типом;
- границами (начало раздела и его длина).

Разбиение памяти на несколько разделов может быть:

- **фиксированным** (статическим);
- **динамическим** (выделение нового раздела памяти происходит непосредственно при появлении новой задачи).

Методы распределения памяти, при которых задаче уже не предоставляется сплошная (непрерывная) область памяти, называются **разрывными**. Для реализации этого метода нужно иметь соответствующую аппаратную поддержку – относительную адресацию: если указать адрес начала текущего фрагмента программы и величину

смещения относительно этого начального адреса, то можно указать необходимую переменную или команду. Виртуальный адрес можно представить состоящим из двух полей:

- указатель на часть программы (с которой идет работы) для определения местоположения этой части;
- относительный адрес нужной ячейки памяти (по отношению к найденному адресу).

Программист может самостоятельно разбивать программу на фрагменты или возложить эту задачу на систему программирования.

Для **сегментного способа организации виртуальной памяти** программу нужно разбить на части и уже каждой части выделить физическую память. Каждый программный модуль или их совокупность могут быть восприняты как отдельные сегменты. Каждый сегмент размещается в оперативной памяти как самостоятельная единица. Логически обращение к элементам программы производится как указание имени сегмента и смещения относительно его начала. Физически имя (или порядковый номер) сегмента соответствует некоторому адресу, с которого этот сегмент начинается при его размещении в памяти, и смещение должно прибавляться к этому адресу.

Страничный способ организации виртуальной памяти – способ разрывного размещения задач в памяти, при котором все фрагменты задачи имеют одинаковый размер, кратный степени двойки (чтобы вместо операции сложения для получения физического адреса можно было использовать операцию конкатенации). При таком способе все фрагменты программы, на которые она разбивается (кроме последней части) получаются одинаковыми. Одинаковыми должны быть и единицы памяти, предоставляемые для размещения фрагментов программы. Эти одинаковые части называются страницами:

- оперативная память разбивается на физические страницы;
- программа разбивается на виртуальные страницы.

Часть виртуальных страниц располагается в оперативной памяти, а часть – во внешней (файл подкачки, страничный файл, swap-файл).

При сегментно-страничном способе организации виртуальной памяти программа разбивается на логически законченные части – сегменты, виртуальный адрес содержит указание на номер соответствующего сегмента. Вторая составляющая виртуального адреса – смещение относительно начала сегмента, может состоять из двух полей:

- виртуальной страницы;
- индекса.

Виртуальный адрес состоит из трех компонентов:

- сегмента;
- страницы;
- индекса.

2.3. Вопросы к главе 2

1) Какие дисциплины диспетчеризации вы знаете?
2) Что такое гарантия обслуживания?
3) Опишите механизмы диспетчеризации. В чем их различия?
4) Что такое виртуальный адрес и виртуальное адресное пространство?
5) Сравните сегментный и страничный способы организации памяти.

3. Управление вводом/выводом и файловые системы

Программирование задач управления вводом/выводом является наиболее сложным, требующим высокой квалификации, поэтому подпрограммы ввода/вывода:

- оформляли в виде системных библиотечных процедур;
- включили в операционную систему, чтобы не включать этот код в каждую программу, а только оформить обращение к нему.

Системы программирования вставляют в машинный код необходимые библиотечные подпрограммы ввода/вывода и обращения к тем системным программным модулям, которые управляют операциями обмена между оперативной памятью и внешними устройствами.

Управление вводом/выводом – одна из основных функций любой операционной системы.

Организация ввода/вывода в различных ОС имеет много общего, а реализация сильно отличается от системы к системе.

3.1. Основные понятия и концепции организации ввода/вывода в ОС

Сложность проектирование ввода/вывода возникает из-за огромного числа устройств различной природы и назначения. Разработчик ввода/вывода должен решить две задачи:

- обеспечить эффективное управление устройствами ввода/вывода;
- создать удобный и эффективный интерфейс устройств ввода/вывода, позволяющий прикладным программистам просто считывать или сохранять данные.

Система ввода/вывода должна быть универсальной.

Главный принцип ввода/вывода – любые операции по управлению вводом/выводом объявляются привилегированными и могут выполняться только самой ОС. Для обеспечения этого принципа в большинстве процессоров вводятся два режима:

- **режим пользователя**, выполнение команд ввода/вывода запрещено;
- **режим супервизора**, выполнение команд ввода/вывода разрешено.

Использование команд ввода/вывода в пользовательском режиме вызывает исключение (прерывание) и управление передается ОС.

Для мультипрограммных ОС одним из основных видов ресурсов являются устройства ввода/вывода и обслуживающие их программы. ОС должны управлять разделяемыми и неразделяемыми устройствами и позволять параллельно выполняющимися задачам использовать различные устройства ввода/вывода.

Непосредственное обращение к внешним устройствам из пользовательских программ не разрешено по трем причинам:

- возможные конфликты при доступе к устройствам ввода/вывода;
- повышение эффективности использование этих ресурсов;
- ошибки в программах ввода/вывода могут привести к разрушению системы.

Компонента ОС, выполняющая ввод/вывод называется **супервизором ввода/вывода**. Основные задачи супервизора следующие:

- получение, проверка на корректность и выполнение запросов на ввод/вывод от прикладных задач и от модулей самой системы;
- планирование ввода/вывода: выполнение или постановка в очередь;

- инициирование ввода/вывода – передача управления драйверам;
- при получении сигналов прерывания передача управления соответствующей программе обработки прерывания;
- передача сообщений об ошибках, если они появляются;
- передача сигнала о завершении операции ввода/вывода.

Если устройство ввода/вывода является **инициативным**, управление со стороны супервизора ввода/вывода заключается в активизации соответствующего вычислительного процесса. **Инициативное устройство** – устройство, по сигналу прерывания от которого запускается соответствующая ему программа.

Имеются два основных режима ввода/вывода:

- **режим обмена с опросом готовности**;
- **режим обмена с прерываниями**.

Центральный процессор посылает устройству управления команду для ввода/вывода. Устройство ввода/вывода исполняет команду, преобразуя ее в сигнал, понятный устройству ввода/вывода. Устройства ввода/вывода намного медленнее центрального процессора, поэтому сигнал готовности приходится очень долго ждать, постоянно опрашивая устройство интерфейса. В режиме опроса готовности драйвер, управляющий процессом обмена данными с внешним устройством, выполняет в цикле команду «поверить готовность устройства». Центральный процессор в таком режиме используется нерационально.

Режим обмена с прерываниями является режимом асинхронного управления. Драйверы, работающие в режиме прерываний, представляют собой сложный комплекс программ и могут иметь несколько секций:

- секцию запуска, которая инициирует операцию ввода/вывода, включает устройство или инициирует очередь ввода/вывода;
- одну или несколько секций продолжения, которые являются обработчиками прерываний;
- секцию завершения, которая выключает устройство и завершает операцию.

Для организации использования многими параллельно выполняющимися задачами устройств ввода/вывода, которые не могут быть разделяемыми, введено понятие виртуального устройства (спулинга). Главная задача **спулинга** – создать видимость параллельного разделения устройства ввода/вывода с последовательным доступом, которое должно быть монопольным и быть закрепленным. Например, каждому вычислительному процессу можно предоставить не реальный, а виртуальный принтер, и поток выводимых символов сначала направить в файл на диске. По окончанию виртуальной печати в соответствии с дисциплиной обслуживания и приоритетами приложений содержимое спул файла выводится на принтер. Системный процесс, управляющий спул файлом, называется **спулером**.

Синхронный ввод/вывод характеризуется тем, что задача, выдавшая запрос на операцию ввода/вывода, переводится супервизором в состояние ожидания завершения указанной операции. Когда супервизор получает от секции завершения сообщение о завершении, он переводит задачу в состоянье готовности к выполнению, и она продолжает свою работу. Синхронный ввод/вывод является стандартным для большинства ОС.

Простейший вариант **асинхронного вывода** – буферизованный вывод данных на внешнее устройство, при котором данные из приложения передаются не непосредственно на устройство ввода/вывода, а в специальный системный буфер. В этом случае логически операция вывода считается законченной, и задача может не ожидать реального процесса вывода данных на устройство. Процессом реального вывода занимается супервизор ввода/вывода. Асинхронный вывод возможен при наличии двух условий:

- в запросе на вывод было указано на необходимость буферизации данных;
- устройство вывода допускает асинхронные операции.

Для организации асинхронного ввода необходимо:

- выделить область памяти для временного хранения считываемых с устройства данных;
- связать выделенный буфер с задачей, заказавшей операцию ввода;
- запрос на операцию ввода разбить на две части (два запроса).

В первом запросе указывается операция на ввод данных, как при асинхронном вводе, и имя буфера для вводимых данных. После этого задача продолжает выполнение или переводится в режим ожидания выполнения, но не переводится в ожидания завершения операции ввода/вывода, как при асинхронном вводе. После выполнения некоторого объема программного кода задача выдает второй запрос на завершение операции ввода и, если операция ввода данных завершена к этому времени, то выбирает данные из системного буфера, если операция ввода не завершена, то задача приостанавливается до завершения ввода, как при асинхронном вводе.

Накопители на магнитных дисках обладают крайне низкой скоростью по сравнению с быстродействием центральной части процессора. С учетом того, что операции чтения/записи на диск производятся несколькими большими буферами, средняя скорость работы процессора с оперативной памятью на 2 – 3 порядка выше, чем скорость передачи данных из внешней памяти на магнитных дисках в оперативную память. Чтобы сгладить такое несоответствие в производительности основных подсистем, используется **буферирование** и/или **кэширование** данных.

Простейший вариант – использование двойного буферирования: пока в один буфер заносятся данные с магнитного диска, из второго буфера ранее считанные данные могут быть прочитаны запросившей их программой. Аналогичный процесс происходит при записи. Буферирование используется во всех ОС.

Кэширование полезно в том случае, когда программа неоднократно читает с диска одни и те же данные. После того как они один раз будут помещены в кэш, обращение к диску больше не потребуется, и скорость работы программы значительно возрастет. Под КЭШем понимается некий пул буферов, управление которым производится с помощью системного процесса.

3.2. Функции файловой системы ОС и иерархия данных

Файл – набор данных, организованных в виде совокупности записей одинаковой структуры. **Файловая система** – это набор спецификаций и соответствующее им программное обеспечение, которое отвечает за создание, удаление, организацию, чтение, запись, модификацию и перемещение файлов информации, а также за управление доступом к файлам и за управление ресурсами, которые используются файлами. Файловая система определяет способ организации данных на диске (или на другом носителе).

Все современные ОС имеют соответствующие **системы управления файлами**. Система управления файлами (СУФ) является основной подсистемой в абсолютном большинстве современных операционных систем:

- через систему управления файлами связываются по данным все системные обрабатывающие программы;
- с помощью СУФ решаются проблемы централизованного распределения дискового пространства и управления данными;
- с помощью СУФ пользователям предоставляются возможности работать с файлами.

СУФ предоставляет пользователям следующие возможности:

- создание, удаление, переименование и другие операции над именованными наборами данных (файлами) из своих программ или посредством специальных управляющих программ, реализующих функции интерфейса пользователя;
- работа с не дисковыми периферийными устройствами как с обычными файлами;
- обмен данными между файлами, файлом и устройством, между устройствами;
- работа с файлами с помощью обращений к программным модулям СУФ;
- защита файлов от несанкционированного доступа.

В некоторых ОС может быть несколько систем управления файлами, что обеспечивает возможность работы с несколькими файловыми системами. СУФ, являясь компонентой системы, зависит от нее. Основное назначение файловой системы и соответствующей ей системы управления файлами – организация удобного доступа к данным, организованным в файлы. Файловая система определяет принципы доступа к данным. Любая СУФ не разрабатывалась сама по себе, а для конкретной ОС.

Для того чтобы можно было загрузить с магнитного диска собственно саму ОС, а уже с ее помощью и организовывать работу той или иной СУФ, были приняты специальные системные соглашения о структуре диска. Первый сектор магнитного диска содержит информацию о логической организации диска и простейшую программу, с помощью которой можно найти и вызвать программу загрузки самой ОС.

Информация на магнитных дисках размещается и передается блоками. Каждый блок называется **сектором** и располагается на концентрических дорожках поверхности диска. Группа дорожек одного радиуса, расположенных на поверхностях магнитных дисков, образуют **цилиндры**. Каждый сектор состоит из **поля данных** и **поля служебной информации**, ограничивающей и идентифицирующей его. Размер сектор (объем поля данных) устанавливается контроллером или драйвером. Физический адрес сектора на диске определяется с помощью трех «координат»:

- номер цилиндра;
- номер рабочей поверхности диска;
- номер сектора на дорожке.

Обмен информацией между ОЗУ и дисками физически осуществляется только секторами.

Диск может быть разбит на несколько **разделов**, которые могут использоваться как одной ОС, так и различными. На каждом разделе может быть организована своя файловая система. Для организации хотя бы одной файловой системы должен быть определен, по крайней мере, один раздел.

Разделы могут быть двух типов:

- первичный;
- расширенный.

Максимальное число первичных разделов – четыре, но обязательно должен быть хотя бы один. Если первичных разделов больше одного, то один должен быть **активным**, в нем находится загрузчик ОС.

На одном диске может быть только один расширенный раздел, который в свою очередь может содержать большое количество подразделов – **логических дисков**.

3.3. Файловые системы FAT, FAT32, NTFS и s5

3.3.1. Файловая система FAT

Аббревиатура FAT (file allocation table) означает «таблица размещения файлов». Этот термин относится к линейной табличной структуре со сведениями о файлах – именами файлов, их атрибутами и другими данными, определяющими местоположение файлов или их фрагментов в среде FAT. Элемент FAT определяет фактическую область диска, в котором хранится начало физического файла.

В файловой системе FAT логическое дисковое пространство любого логического диска делится на две области:

- системную область;
- область данных.

Системная область создается при форматировании и обновляется при манипулировании файловой структурой. Область данных содержит файлы и каталоги, подчиненные корневому, и доступна через пользовательский интерфейс. Системная область состоит из следующих компонентов:

- загрузочной записи;
- зарезервированных секторов;
- таблицы размещения файлов (FAT);
- корневого каталога.

Таблица размещения файлов представляет собой карту (образ) области данных, в которой описывается состояние каждого участка области данных. Область данных разбивается на кластеры. **Кластер** – один или несколько смежных секторов в логическом дисковом адресном пространстве (только в области данных). В таблице FAT кластеры, принадлежащие одному файлу (некорневому каталогу), связываются в цепочки. Для указания номера кластера в системе управления файлами FAT16 используется 16-битовое слово, следовательно, можно иметь до 65536 кластеров.

Кластер – минимальная адресуемая единица дисковой памяти, выделяемая файлу или некорневому каталогу. Файл или каталог занимает целое число кластеров. Последний кластер при этом может быть задействован не полностью, что приведет к заметной потере дискового пространства при большом размере кластера.

Так как FAT используется при доступе к диску очень интенсивно, она загружается в ОЗУ и находится там максимально долго.

Корневой каталог отличается от обычного каталога тем, что он размещается в фиксированном месте логического диска и имеет фиксированное число элементов. Для каждого файла и каталога в файловой системе хранится информация в соответствии со следующей структурой:

- имя файла или каталога – 11 байт;
- атрибуты файла – 1 байт;
- резервное поле – 1 байт;
- время создания – 3 байта;
- дата создания – 2 байта;
- дата последнего доступа – 2 байта;
- зарезервировано – 2 байта;
- время последней модификации – 2 байта;
- номер начального кластера в FAT – 2 байта;
- размер файла – 4 байта.

Структура системы файлов является иерархической.

3.3.2. Файловая система FAT32

FAT32 является полностью независимой 32-разрядной файловой системой и содержит многочисленные усовершенствования и дополнения по сравнению с FAT16.

Принципиальное отличие FAT32 заключается в более эффективном использовании дискового пространства: FAT32 использует кластеры меньшего размера, что приводит к экономии дискового пространства.

FAT32 может перемещать корневой каталог и использовать резервную копию FAT вместо стандартной. Расширенная загрузочная запись FAT32 позволяет создавать копии критических структур данных, что повышает устойчивость дисков к нарушениям структуры FAT по сравнению с предыдущими версиями. Корневой каталог представляет собой обычную цепочку кластеров, поэтому может находиться в произвольном месте диска, что снимает ограничение на размер корневого каталога.

3.3.3. Файловая система NTFS

Файловая система NTFS (New Technology File System) содержит ряд значительных усовершенствований и изменений, существенно отличающих ее от других файловых систем. С точки зрения пользователей файлы по-прежнему хранятся в каталогах, но работа на дисках большого объема в NTFS происходит намного эффективнее:

- имеются средства для ограничения доступа к файлам и каталогам;
- введены механизмы, существенно повышающие надежность файловой системы;
- сняты многие ограничения на максимальное количество дисковых секторов и/или кластеров.

Основные характеристики файловой системы NTFS:

- **надежность**. Высокопроизводительные компьютеры и системы совместного использования должны обладать повышенной надежностью, для этой цели введен механизм транзакций, при котором ведется **журналирование** файловых операций;
- **расширенная функциональность**. В NTFS введены новые возможности: усовершенствованная отказоустойчивость, эмуляция других файловых систем, мощная модель безопасности, параллельная обработка потоков данных, создание файловых атрибутов, определенных пользователем;
- **поддержка стандарта POSIX**. К числу базовых средств относятся необязательное использование имен файлов с учетом регистра, хранение времени последнего обращения к файлу и механизм альтернативных имен, позволяющий ссылаться на один и тот же файл по нескольким именам;
- **гибкость**. Распределение дискового пространства отличается большой гибкостью: размер кластера может изменяться от 512 байт до 64 Кбайт.

NTFS хорошо работает с большими массивами данных и большими томами. Максимальный размер тома (и файла) – 16 Эбайт. (1 Эбайт равен 2**64 или 16000 млрд. гигабайт.) Количество файлов в корневом и некорневом каталогах не ограничено. Поскольку в основу структуры каталогов NTFS заложена эффективная структура данных, называемая «бинарным деревом», время поиска файлов в NTFS не связано линейной зависимостью с их количеством.

Система NTFS обладает некоторыми средствами для самовосстановления и поддерживает различные механизмы проверки целостности системы, включая ведение журнала транзакций, позволяющий отследить по системному журналу файловые операции записи.

Файловая система NTFS поддерживает объектную модель безопасности и рассматривает все тома, каталоги и файлы как самостоятельные объекты NTFS. Права доступа к томам, каталогам и файлам зависит от учетной записи пользователя и той группы, к которой он принадлежит.

Файловая система NTFS обладает встроенными средствами сжатия, которые можно применять к томам, каталогам и файлам.

3.3.4. Файловая система s5 операционной системы UNIX System V

Файловая система *s5* занимает слайс диска и состоит из трех основных компонентов:

- суперблок;
- массив индексных дескрипторов;
- блоки данных.

Суперблок содержит общую информацию о файловой системе:

- тип файловой системы;
- размер файловой системы в логических блоках, включая сам суперблок, массив дескрипторов и блоки данных;
- размер массива индексных дескрипторов;
- число свободных блоков, доступных для размещения;
- число свободных блоков для размещения дескрипторов;
- размер логического блока;
- список номеров свободных дескрипторов;
- список адресов свободных блоков.

Массив индексных дескрипторов. Индексный дескриптор **(inode)** содержит информацию о файле, т.е. метаданные файла. Каждый файл связан с одним **inode**, хотя может иметь несколько имен в файловой системе, каждое из которых будет указывать на один и тот же **inode**. Поля индексного дескриптора содержат следующую информацию:

- тип файла и права доступа;
- число ссылок на файл, т.е. количество имен, которые имеет файл в файловой системе;
- идентификаторы владельца и группы;
- размер файла в байтах; для специальных файлов это поле содержит старший и младший номера устройств;
- время последнего доступа к файлу;
- время последней модификации файла;
- время последней модификации **inode**;
- массив адресов дисковых блоков, где хранятся данные файла.

Массив адресов дисковых блоков содержит информацию о расположении данных файла. Поскольку дисковые блоки хранения данных файла могут располагаться не последовательно, **inode** должен хранить физические адреса блоков, принадлежащих данному файлу. В индексном дескрипторе эта информация хранится в виде массива, каждый элемент которого содержит физический адрес дискового блока, а индексом массива является номер логического блока файла. Массив имеет фиксированный размер и состоит из 13 элементов. Первые 10 элементов адресуют непосредственно блоки хранения данных файла. Одиннадцатый элемент адресует блок, в свою очередь содержащий адреса блоков хранения данных. Двенадцатый элемент указывает на дисковый блок, тоже хранящий адреса блоков, каждый из которых адресует блок хранения данных файла. Тринадцатый элемент используется для тройной косвенной адресации, когда для нахождения адреса блока хранения данных файла используется три дополнительных блока.

Такой подход позволяет при относительно небольшом фиксированном размере индексного дескриптора поддерживать работу с файлами, размер которых может

изменяться от нескольких байтов до десятка мегабайтов. Для относительно небольших файлов (до 10 Кбайт при размере блока 1024 байта) используется прямая индексация, обеспечивающая максимальную производительность. Для файлов, размер которых не превышает 266 Кбайт (10Кбайт + 256*1024) достаточно простой косвенной адресации. Наконец, при использовании тройной косвенной адресации можно обеспечить доступ к 16777216 блокам (256*256*256).

Как и во многих современных операционных системах, в ОС UNIX файлы организованы в виде древовидной структуры, называемой файловой системой (*file system*). Каждый файл имеет имя, определяющее его расположение в дереве файловой системы. Корнем этого дерева является корневой каталог (*root directory*), имеющий имя "/". Пример файлового дерева приведен на рис. 5.2.

Для ОС UNIX характерно, что в системе может присутствовать несколько файловых систем, которые могут иметь различную внутреннюю структуру, а файлы, принадлежащие этим файловым системам, могут быть расположены на различных устройствах. Следует отметить, что имя файла является атрибутом файловой системы, а не набора данных на диске. Каждый файл в системе связан со своими метаданными, хранящимися в индексных дескрипторах (*inode*), которые содержат все характеристики файла, в том числе и указатели на дисковые блоки, где хранятся данные файла. Имя файла в файловой системе является указателем на его метаданные, в то время как метаданные не содержат указателя на имя файла.

3.4. Вопросы к главе 3

1) Почему создание подсистемы ввода/вывода считается одной из самых сложных областей проектирования операционных систем?
2) Почему операции ввода/вывода объявляются привилегированными?
3) Перечислите основные задачи, возлагаемые на супервизор ввода/вывода.
4) В каких случаях устройство ввода/вывода называется инициативным?
5) Какие режимы управления вводом/выводом вы знаете, опишите каждый из них.
6) Что означают термины «spooling» и «swapping»?
7) Чем обеспечивается независимость пользовательских программ от устройств ввода/вывода, подключенных к компьютеру?
8) Что такое синхронный и асинхронный ввод/вывод?
9) Что такое кэширование операций ввода/вывода при работе с накопителями на магнитных дисках?
10) Что такое «файловая система»? Что обеспечивает использование той или иной файловой системы?
11) Объясните общие принципы файловой системы FAT. Что такое кластер и от чего зависит его размер?
12) Сравните файловые системы FAT16 и FAT32. В чем заключаются их достоинства и недостатки?
13) Расскажите о правилах, которые определяют состояние прав доступа при перемещении или копировании объектов, если используется NTFS.
14) Объясните структуру файловой системы s5. Что хранится в каталогах? Где хранятся права доступа к файлам и каталогам?

4. Архитектура операционных систем.

В настоящее время уже никто не разрабатывает ОС, кроме специализирующихся на этом фирм, а все являются только пользователями.

4.1. Основные принципы построения операционных систем

4.1.1. Принцип модульности

Модуль – функционально законченный элемент системы, отвечающий требованиям межмодульного интерфейса. Из определения следует, что один модуль можно заменить на другой. Способы обособления отдельных частей ОС могут различаться, но чаще всего разделение происходит по функциональному принципу.

Особенно важное значение при построении ОС имеют модули, позволяющие более эффективно использовать ресурсы вычислительной системы:

- привилегированные;
- повторно входимые;
- реентерабельные.

В некоторых ОС реентерабельность достигается автоматически:

- при неизменяемости кодовых частей программы при исполнении;
- при автоматическом распределении регистров;
- при автоматическом отделении кодовых частей программ от данных и помещении данных в системную область памяти.

Принцип модульности отражает технологические и эксплуатационные свойства ОС. Наибольший эффект достигается при распространении принципа модульности на ОС, прикладные программы и аппаратуру.

4.1.2. Принцип функциональной избирательности

Часть модулей, которые должны постоянно находиться в оперативной памяти для более эффективной организации вычислительного процесса, называется **ядром** ОС. При формировании состава ядра следует учитывать два противоречивых требования:

- в состав ядра должны войти наиболее часто используемые системные модули;
- количество модулей должно быть таковым, чтобы объем памяти, занимаемый ядром, не был слишком большим.

В состав ядра входят, как правило, следующие модули:

- модули по управлению системой прерываний;
- средства по переводу программ из состояния выполнения в состояние ожидания, готовности и обратно;
- средства по распределению основных ресурсов: оперативной памяти и процессорного времени.

Транзитные программные модули загружаются в память только при необходимости и в случае отсутствия свободного дискового пространства могут быть замещены другими транзитными модулями.

4.1.3. Принцип генерируемости ОС

Принцип генерируемости - возможность настраивать системную супервизорную часть (ядро и основные компоненты), исходя из конкретной конфигурации

вычислительного комплекса и класса решаемых задач. Процедура генерации производится с помощью программы-генератора и языка описания входных данных для этой программы. В результате генерации получается полная версия ОС – совокупность системных наборов модулей и данных.

Принцип модульности положительно проявляется при генерации ОС. Он упрощает настройку ОС на требуемую конфигурацию вычислительной системы. Принцип генерируемости реализован в ОС, типа UNIX.

4.1.4. Принцип функциональной избыточности

Этот принцип дает возможность проведения одной и той же работы различными способами. В состав ОС может входить:

- несколько типов планировщиков (модулей супервизора, управляющих тем или иным видом ресурсов);
- различные средства организации связи между вычислительными процессами.

Это дает возможность пользователям:

- быстро и наиболее адекватно адаптировать ОС к определенной конфигурации вычислительной системы;
- обеспечить максимально эффективную загрузку технических средств при решении конкретного класса задач;
- получить максимальную производительность при решении заданного класса задач.

4.1.5. Принцип виртуализации

Построение виртуальных ресурсов, их распределение и использование в настоящее время имеет место почти в каждой ОС. Этот принцип позволяет представить структуру системы в виде определенного набора планировщиков процессов и распределителей ресурсов и использовать единую централизованную схему распределения ресурсов.

Понятие **виртуальная машина** является итогом концепции виртуальности. Любая ОС, являясь средством распределения ресурсов и организуя по определенным правилам управление процессами, скрывает от пользователя и его приложений реальные аппаратные и иные ресурсы, заменяя их абстракциями. Пользователь видит виртуальную машину как некое устройство, способное воспринимать его программы и команды. Пользователя не интересует реальная конфигурация вычислительной системы и способы управления ее компонентами. Он оперирует с теми ресурсами, которые ему предоставлены в рамках виртуальной машины.

Виртуальная машина, предоставляемая пользователю, воспроизводит архитектуру реальной машины, но архитектурные элементы в таком представлении имеют новые или улучшенные характеристики, часто упрощающие работу с системой. Характеристики могут быть произвольными, но обычно пользователи хотят видеть идеальную по своим архитектурным характеристикам машину:

- единообразная по логике работы память практически неограниченного объема. Среднее время доступа соизмеримо со временем доступа к оперативной памяти. Организация работы с информацией в такой памяти производится в терминах обработки данных на уровне выбранного пользователем языка программирования;
- произвольное количество (виртуальных) процессоров, способных работать параллельно и взаимодействовать во время работы. Способы управления процессорами (синхронизация и информационные взаимодействия)

реализованы и доступны пользователям на уровне используемого языка в терминах управления процессами;

- произвольное количество (виртуальных) внешних устройств, способных работать с памятью виртуальной машины параллельно или последовательно, асинхронно или синхронно по отношению к работе того или иного виртуального процессора, которые инициируют работу этих устройств. Информация, хранимая или передаваемая на виртуальные устройства, не ограничена допустимыми размерами. Доступ к такой информации осуществляется на основе либо последовательного, либо прямого способа доступа в терминах соответствующей системы управления файлами. Предусмотрено расширение информационных структур данных, хранимых на виртуальных устройствах.

Степень приближения к «идеальной» виртуальной машине может быть большей или меньшей в каждом конкретном случае. Чем больше виртуальная машина, реализуемая средствам ОС на базе конкретной аппаратуры, приближена к идеальной по характеристикам машине, чем больше ее архитектурно-логические характеристики отличны от реально существующих, тем больше степень виртуальности у полученной пользователем машины.

4.1.6. Принцип независимости программ от внешних устройств

Этот принцип в настоящее время реализуется в подавляющем большинстве современных ОС общего назначения. Принцип независимости заключается в том, что связь программ с конкретными устройствами производится не на уровне трансляции программ, а в период планирования ее исполнения. При работе с новым устройством для хранения данных перекомпиляция не требуется.

Принцип независимости позволяет одинаково осуществлять операции управления внешними устройствами независимо от конкретных физических характеристик. Смена носителя и данных, размещенных на нем, не принесет каких-либо изменений в программу, если в системе реализован принцип независимости.

4.1.7. Принцип совместимости

Одним из аспектов совместимости является способность ОС выполнять программы, написанные:

- для других ОС;
- для более ранних версий данной операционной системы;
- для другой аппаратной платформы.

Совместимость подразделяется на два аспекта:

- двоичная совместимость;
- совместимость на уровне исходных текстов приложений.

При **двоичной совместимости** можно взять исполняемую программу и выполнить ее в среде другой ОС. Для этого необходимы:

- совместимость на уровне команд процессора;
- совместимость на уровне системных вызовов;
- совместимость на уровне библиотечных вызовов, если они являются динамически связываемыми.

Совместимость на уровне исходных текстов требует:

- наличия соответствующего транслятора в составе системного программного обеспечения;

- совместимости на уровне библиотек и системных вызовов.

Необходимо перекомпилировать имеющиеся исходные тексты в новый выполняемый модуль.

Одним из средств обеспечения совместимости программных и пользовательских интерфейсов является соответствие стандартам POSIX. Использование стандарта POSIX позволяет создавать программы в стиле UNIX, которые могут легко переноситься из одной ОС в другую.

4.1.8. Принцип открытой и наращиваемой ОС

Открытая ОС доступна для анализа как системным специалистам, обслуживающим вычислительную систему, так и пользователям. Наращиваемая ОС позволяет не только использовать возможности генерации, но и вводить в состав ОС новые модули, совершенствовать старые и т.д.

Этот принцип требует, чтобы можно было легко внести дополнения и изменения в ОС, если потребуется, и не нарушить целостность ОС.

К открытым системам в первую очередь относятся UNIX-подобные системы.

4.1.9. Принцип модульности (переносимости)

Операционная система должна относительно легко переноситься:

- с процессора одного типа на процессор другого типа;
- с аппаратной платформы (архитектуры вычислительной системы) одного типа на аппаратную платформу другого типа.

Принцип переносимости близок принципу совместимости, но это не одно и то же.

Написание переносимой ОС, как и любой переносимой программы, должно следовать определенным правилам:

- большая часть операционной системы должна быть написана на языке, который имеется во всех вычислительных системах, на которые планируется в дальнейшем ее переносить. Это должен быть стандартизованный язык высокого уровня, например, язык С. программы, написанные на ассемблере, в общем случае не являются переносимыми;
- минимизировать или исключить ту часть кода, которая непосредственно взаимодействует с аппаратурой. Если аппаратный код не может быть исключен, он должен быть изолирован в нескольких модулях.

4.1.10. Принцип обеспечения безопасности вычислений

Обеспечение безопасности при выполнении вычислений является желательным свойством для любой многопользовательской системы. Правила безопасности определяют следующие свойства:

- защита ресурсов одного пользователя от других;
- установка квот по ресурсам для предотвращения захвата одним пользователем всех системных ресурсов.

Обеспечение безопасности информации от несанкционированного доступа является обязательной функцией сетевых операционных систем. Во многих современных ОС гарантируется степень безопасности данных, соответствующих уровню С2 в системе стандартов США.

В соответствии с требованиями Оранжевой книги безопасной считается система, которая «посредством специальных механизмов защиты контролирует доступ к

информации таким образом, что только имеющие соответствующие полномочия лица или процессы, выполняющиеся от их имени, могут получить доступ на чтение, запись, создание или удаление информации». Низший класс – D, высший – A.

Основные свойства для систем класса C:

- наличие подсистемы учета событий, связанных с безопасностью;
- избирательный контроль доступа.

Класс C делится на два подуровня:

- C1 обеспечивает защиту данных от ошибок пользователей, но не злоумышленников;
- C2 более строгий уровень контроля.

На уровне C2 должны присутствовать:

- средства секретного входа, обеспечивающие идентификацию пользователей путем ввода уникального имени и пароля перед тем, как им будет разрешен доступ к системе;
- избирательный контроль доступа, позволяющий владельцу ресурса определить, кто имеет доступ к ресурсу и что он может с ним делать. Владелец делает это путем предоставления прав доступа пользователю или группе пользователей;
- средства учета и наблюдения, обеспечивающие возможность обнаружить и зафиксировать важные события, связанные с безопасностью, или любые попытки создать, получить доступ или удалить системные ресурсы;
- защита памяти, память повторно инициируется перед повторным использованием.

На уровне C2 система не защищена от ошибок пользователя, но поведение его может быть проконтролировано по протоколам, оставленным средствами наблюдения и аудита.

4.2. Микроядерные операционные системы

Микроядро – минимальная стержневая часть операционной системы, служащая основой модульных и переносимых расширений.

Основная идея, заложенная в технологию микроядра (ОС или графический интерфейс), заключается в том, чтобы конструировать необходимую среду верхнего уровня, из которой можно легко получить доступ ко всем функциональным возможностям уровня аппаратного обеспечения, ядро служит стартовой точкой для создания системы.

Искусство разработки микроядра заключается в выборе базовых примитивов, которые должны в нем находиться для обеспечения необходимого и достаточного сервиса. В микроядре содержится и исполняется минимальное количество кода, необходимое для реализации основных системных вызовов:

- передача сообщений;
- организация взаимодействия между внешними по отношению к микроядру процессами;
- поддержка управления прерываниями и др.

Микроядро – маленький модуль системного программного обеспечения, работающий в наиболее приоритетном состоянии компьютера и поддерживающий остальную часть операционной системы, рассматриваемую как набор серверных приложений.

Исполняемые микроядром функции ограничены в целях сокращения его размера и максимизации количества кода, работающего как прикладная программа.

Микроядро включает только те функции, которые требуются для определения набора абстрактных сред обработки для прикладных программ и для организации совместной работы приложений. Микроядро обеспечивает пять различных типов сервисов:

- управление виртуальной памятью;
- задания и потоки;
- межпроцессные коммуникации;
- управление вводом/выводом и прерываниями;
- сервисы набора Хоста и процессора.

В качестве приложения ядра работают следующие подсистемы и функции операционной системы:

- система управления файлами;
- поддержка внешних устройств;
- традиционные программные интерфейсы.

Микроядра проще, чем ядра монолитных или модульных операционных систем. Функция операционной системы разбивается на модульные части, которые могут быть сконфигурированы целым рядом способов.

Поскольку микроядра малы и имеют сравнительно мало требуемого к исполнению кода уровня ядра, они обеспечивают удобный способ поддержки характеристик реального времени

4.3. Монолитные операционные системы

Монолитные операционные системы являются прямой противоположностью микроядерным. В монолитной ОС очень трудно удалить один из уровней многоуровневой модульной структуры. Добавление новых функций и изменение существующих для монолитных ОС требует очень хорошего знания всей архитектуры ОС и чрезвычайно больших усилий. Для преодоления этих трудностей используется технология «сервер – клиент».

Модель сервер – клиент предполагает наличие программного компонента, являющегося потребителем какого-либо сервиса – **клиента**, и программного компонента, поставщика этого сервиса – **сервера**. Взаимодействие между сервером и клиентом стандартизуется, сервер может обслуживать клиентов, реализованных различными способами. Главное требование – использование единообразного интерфейса. Инициатором обмена является клиент, который посылает запрос серверу. Один и тот же программный компонент может быть и клиентом, и сервером.

При поддержке монолитных ОС возникает ряд проблем, связанных с тем, что все функции микроядра работают в едином адресном пространстве:

- опасность возникновения конфликта между различными частями ядра;
- сложность подключения к ядру новых драйверов.

Преимущество микроядерной архитектуры перед монолитной заключается в том, что каждый компонент системы представляет собой самостоятельный процесс, запуск и останов которого не отражается на работоспособности остальных процессов.

4.4. Требования, предъявляемые к ОС реального времени

Система реального времени должна давать отклик на непредсказуемые внешние воздействия в течение предсказуемого интервала времени. Свойства операционных систем реального времени:

- **ограниченное время отклика**, реакция на событие гарантированно последует до наступления крайнего срока;
- одновременность обработки. Даже если возникает более одного события одновременно, временные ограничения для всех событий должны быть выдержаны. В системе реального времени должен присутствовать параллелизм, который достигается использованием нескольких процессоров и/или многозадачного подхода.

Примеры систем реального времени:

- системы управления атомными электростанциями;
- системы управления технологическими процессами;
- системы медицинского мониторинга;
- системы управления вооружением;
- системы космической навигации;
- системы разведки;
- системы управления лабораторными экспериментами;
- системы управления автомобильными двигателями;
- робототехника;
- телеметрические системы управления;
- системы антиблокировки тормозов;
- системы сигнализации и т.д.

Различают системы «мягкого» и «жесткого» реального времени. Различия зависят от требований к системе:

- в «жесткой» системе нарушение временных ограничений не допустимо;
- в «мягкой» системе нарушение временных ограничений нежелательно.

Основные требования к операционной системе реального времени:

1) **мультипрограммность и многозадачность (многопоточность)**. ОС должна активно использовать прерывания для диспетчеризации. Максимальное время выполнения того или иного действия должно быть известно заранее и соответствовать требованиям приложения;
2) **приоритеты задач (потоков).** Проблема, какой задаче ресурс требуется больше всего. В идеальной ситуации ОСРВ отдает ресурс потоку или драйверу с ближайшим крайнем сроком завершения. Чтобы реализовать этот принцип ОС должна знать, сколько времени требуется каждому процессу для его завершения. Таких ОС нет, так как их очень сложно реализовать, поэтому вводится понятие уровня приоритета для задачи и временные ограничения сводятся к приоритетам;
3) **наследование приоритетов**. ОСРВ должна допускать наследование приоритета, то есть повышение уровня приоритета потока до уровня приоритета потока, который его вызывает. Наследование означает, что блокирующий ресурс поток наследует приоритет потока, который он блокирует;
4) **синхронизация процессов и задач**. Так как задачи разделяют данные (ресурсы) и должны сообщаться друг с другом, то должны существовать механизмы блокирования и коммуникации. Эти системные механизмы должны быть всегда доступны процессам, требующим реального времени;
5) **предсказуемость**. Времена выполнения системных вызовов и временные характеристики поведения системы в различных обстоятельствах должны быть известны разработчику.

Разработчик ОСРВ должен привести следующие характеристики:

- задержку прерывания, время от момента прерывания до момента запуска задачи;
- максимальное время выполнения каждого системного вызова;
- максимальное время маскирования прерываний драйверами и ОС.

4.5. Принципы построения интерфейсов операционных систем

Операционная система – интерфейс между аппаратурой компьютера и пользователем с его задачей. Интерфейс операционных систем – специальные интерфейсы системного и прикладного программирования, предназначенные для выполнения следующих задач:

- управление процессами, которое включает в себя следующий набор основных функций:
  - запуск, приостанов и снятие задачи с выполнения;
  - задание или изменение приоритета задачи;
  - взаимодействие задач между собой (сигналы, семафоры, очереди, конвейеры, почтовые ящики);
  - удаленный вызов подпрограмм;
- управление памятью:
  - запрос на выделение блока памяти;
  - освобождение памяти;
  - изменение параметров блока памяти;
  - отображение файлов на память;
- управление вводом/выводом:
  - запрос на управление виртуальными устройствами;
  - файловые операции.

Пользовательский интерфейс ОС реализуется с помощью специальных программных модулей, которые принимают его команды на соответствующем языке и транслируют их в обычные вызовы в соответствии с основным интерфейсом системы. Обычно эти модули называются интерпретатором команд.

Имеются два основных подхода к управлению задачами:

1) порождаемая задача наследует все ресурсы задачи-родителя;
2) при порождении нового процесса ресурсы для него запрашиваются у операционной системы.

Обращение к операционной системе в соответствии с имеющимися API может осуществляться:

- посредством вызова подпрограммы с передачей ей необходимых параметров;
- через механизм программных прерываний.

Интерфейс прикладного программирования предназначен для использования прикладными программами системных ресурсов ОС и реализуемых ею функций.

Термин API (application program interface, интерфейс прикладного программирования):

- API как интерфейс высокого уровня, принадлежащий к библиотекам RTL (run time library, библиотека во время выполнения);
- API прикладных и системных программ, входящих в поставку операционной системы;
- прочие API.

API представляет собой набор функций, предоставляемых системой программирования разработчику прикладной программы и ориентированных на организацию взаимодействия результирующей программы с целевой вычислительной системой (совокупность аппаратных и программных средств, в окружении которых выполняется результирующая программа).

API используется не только прикладными, но и многими системными программами как в составе ОС, так и в составе системы программирования.

Программный интерфейс API включает в себя не только сами функции, но и соглашения об их использовании, которые зависят от:

- операционной системы;
- архитектуры целевой вычислительной системы;
- системы программирования.

Варианты реализации API:

- на уровне ОС;
- на уровне системы программирования;
- на уровне внешней библиотеки процедур и функций.

В каждом из этих вариантов разработчику предоставляется возможность подключить функции API к исходному коду программы и организовать их вызов.

Возможности API можно оценить со следующих позиций:

- эффективность выполнения функций API (скорость выполнения, объем вычислительных ресурсов);
- широта предоставляемых возможностей;
- зависимость прикладной программы от архитектуры целевой вычислительной системы.

В идеале набор функций API должен:

- выполняться с наивысшей эффективностью;
- предоставлять пользователю все возможности современных ОС;
- иметь минимальную зависимость от архитектуры вычислительной системы.

4.6. Вопросы к главе 4

1) Перечислите и поясните основные принципы построения операционных систем.
2) Сравните микроядерные и монолитные операционные системы.
3) Перечислите основные требования, предъявляемые к операционным системам реального времени.
4) Какие задачи возлагаются на интерфейс прикладного программирования API.

5. Операционные системы Windows

Недостатки MS-DOS:

- трудности с многозадачностью;
- трудности с защитой памяти;
- трудности с большим адресным пространством.

Отсюда:

Мало пригоден для работы в сети с крупными приложениями.

В главе 5 рассматриваются три главные операционные системы Windows и основы их архитектур.

5.1. Операционные системы Windows

5.1.1. Перечень ОС Windows и их основных характеристик

Windows – это операционная система с графическим интерфейсом, использующая изображения, значки, меню и прочие визуальные элементы.

Windows 3.1 - 16-разрядная оболочка, работающая поверх ОС MS-DOS. В версии 3.11 добавлены средства сетевой поддержки. Версии 3.x работают под управлением ОС MS-DOS.

Windows 9x - 32-разрядная ОС не требует ОС MS-DOS. Поддерживает все существующие 16-разрядные приложения и служит платформой для разработки 32-разрядных приложений. Обладает встроенными сетевыми средствами.

Поддержка аппаратуры Plug-and-Play позволяет изменять конфигурацию персонального компьютера без вмешательства пользователя.

Файловая система FAT32 – расширение FAT. FAT32 более эффективно использует дисковое пространство. В FAT16 таблица размещения файлов разделена на блоки по 16 Кб. Если блок используется не полностью, оставшийся объем пропадает. В FAT32 объем блоков – 4 Кб, поэтому потери меньше. FAT16 поддерживает диски до 2 Гб, а FAT32 – до 2 Тб. Большинство приложений модифицировать не надо, кроме тех, которые обращаются к диску на физическом уровне.

Windows NT - 32-разрядные ОС, применяются для одновременного выполнения множества приложений в многопроцессорных системах.

Программы выполняются в отдельных областях памяти, поэтому неисправное приложение не влияет на работу других. Архитектура Windows NT защищает ОС от приложений, пытающихся монополизировать ресурсы процессора или воспользоваться адресным пространством ОС.

ОС Windows NT Workstation – мощная ОС для компьютеров, объединенных в одноранговую сеть или рабочая станция в составе домена Windows NT Server.

ОС Windows NT Server – основа для серверных приложений, позволяет реализовать стандартные функции серверов файлов и печати.

В настоящее время на смену Windows NT пришла версия **Windows 2000**. Имеется несколько разновидностей Windows 2000, но основные из них Professional и Server:

- **Windows 2000 Professional** содержит все средства, необходимые для операционной системы, предоставляя возможность использовать компьютер и программное обеспечение одному или нескольким пользователям;
- **Windows 2000 Server** – операционная системы, специально предназначенная для управления сетью. Существует несколько разновидностей Windows 2000 Server, каждая из которых создана для выполнения специфических задач по работу с сетями Internet, базами данных и т.д. Данная версия не предназначена для индивидуального использования.

Многопроцессные возможности и файловая система NTFS делают Windows NT и Windows 2000 наиболее защищенными и стабильными операционными системами семейства Windows.

Симметричная многопроцессорная обработка обеспечивает автоматическое использование всех доступных процессоров многопроцессорного компьютера и равномерное распределение потребностей системы и приложений между процессорами.

Слой абстрагирования от аппаратуры позволяет, не теряя производительности на любой платформе, оставаться независимой от аппаратуры.

Файловая система NTFS предпочтительна для ОС Windows NT, могут использоваться и другие файловые системы, но хотя бы один раздел должен быть отформатирован как NTFS. Файловая система NTFS поддерживается только для Windows NT и Windows 2000.

Преимущество – возможность работы с разделами большого размера.

При использовании NTFS следует учитывать, что

- в NTFS встроены средства восстановления после сбоя, не нужно запускать утилиту восстановления;
- на разделе NTFS нельзя восстановить удаленный файл;
- при использовании NTFS значительно уменьшаются потери из-за фрагментации диска;
- NTFS поддерживает модель защиты Windows NT, поэтому все файлы и каталоги можно защитить правами доступа и подвергать аудиту;
- не рекомендуется использовать NTFS на томах размером менее 400 Мб.

5.1.2. Выбор платформы Windows

Выбор Windows-платформы зависит от класса задач, Windows 9x и Windows NT, дополняя друг друга, дает возможность решить большинство задач пользователей.

Офисные задачи – создание документов, запросы к базам данных или анализ электронных таблиц (Microsoft Office). Лучший выбор - Windows 9x.

Удаленные пользователи – мобильные пользователи, работающие вне офиса, у клиента, требовательны к совместимости приложений и не требовательны к ресурсам. Лучший выбор - Windows 9x.

Высокая производительность требуется пользователям научной и технической сферы при интенсивных вычислениях и анализе данных. Windows NT поддерживает симметричную многопроцессорную обработку, а для офисных приложений не нужен отдельный компьютер. Лучший выбор - Windows NT или Windows 2000.

Защита уровня С-2 обеспечивает работу многих пользователей на одном компьютере под управлением Windows NT или Windows 2000 с гарантированной защитой всех файлов в системе. Файловая система NTFS предотвращает неавторизованный доступ к системе и данным. Лучший выбор - Windows NT или Windows 2000.

Высокая надежность – уровень доступности и производительность системы выше среднего, например, управление производством. Windows NT выполняет 16-разрядные приложения в раздельных адресных пространствах, что обеспечивает продолжение выполнения при отказе одного из приложений. Windows NT обеспечивает полную защиту 32-разрядных приложений и может автоматически восстанавливаться после сбоя. Лучший выбор - Windows NT или Windows 2000.

5.1.3. Термины

В системах Windows приняты следующие термины:

- **вход в систему (log-in)** – процедура регистрации в системе, получение доступа к ней;
- **панель управления** – средство для управления видом и функционированием Windows, а также различных элементов оборудования ПК;
- **мой компьютер** – средство, позволяющее видеть все содержимое вашего компьютера;
- **рабочий стол** – то, что появляется на экране сразу после запуска Windows;
- **путь** – полный путь к файлу;
- **подменю** – меню, вложенное в главное меню. Подменю также могут быть вложены в другие подменю. В меню Пуск признаком подменю служит небольшая стрелка рядом с его именем;
- **<F1>** - клавиша для вызова системы справки;
- **система справки** – средство отображения справочной информации о Windows;
- **раздел** – раздел с информацией в системе справки;
- Windows Update – пункт в меню Пуск, позволяющий подключиться к Internet и осуществить оттуда обновление средств Windows;
- **мастер** - средство Windows, облегчающее трудную задачу;
- **значок** – небольшое изображение, являющееся визуальным представлением программы, файла, папки или команды;
- **минимизировать** – способ свернуть приложение в кнопку на панели задач, не закрывая его;
- **панель задач** – один из наиболее полезных элементов рабочего стола Windows. Именно здесь находится кнопка Пуск и появляются кнопки, представляющие все открытые программы;
- **твердая копия** – реальное, из бумаги и чернил, доказательство выполнения на компьютере некоторой работы;
- **операционная система** – это Windows. Операционная система – это то, что управляет компьютером и работой его программ;
- **экранная подсказка** – информация о том, что выполняет та или иная кнопка панели инструментов;
- **мои рисунки** – папка, которую Windows предоставляет для хранения фотографий и других графических файлов;
- **родительский каталог** – каталог, расположенный одним уровнем выше каталога, содержимое которого оператор в данное время просматривает;
- **вставить** – удобная команда, используемая для перемещения выделенной информации, а иногда файлов или папок;
- **отмена удаления** – процесс возвращения удаленного файла на место откуда его удалили;
- **удаленный доступ** – предоставляет соединение между вашим компьютером и Internet, сетью или другим компьютером;
- **домашняя страница** – страница, которую Web-броузер автоматически отображает при подключении к Internet. В процессе работы можно вернуться на домашнюю страницу, щелкнув по кнопке Домой;
- **модем** – небольшое устройство, позволяющее вашему компьютеру подключаться к Internet;
- **средство поиска – Web-страница особого типа, позволяющая (*Uniform Resource Locator – унифицированный локатор ресурсов) – адрес Web-страницы***

- **адресная книга** – список имен и адресов электронной почты, поддерживаемый Outlook Express, облегчает создание новых приложений e-mail;
- **вложения** – файлы, пересылаемые вместе с сообщениями электронной почты;
- **группа новостей** – большая группа людей, которым вы можете одновременно послать сообщение;
- **архив** – набор файлов, сжатых и собранных в один файл для облегчения передачи. Наиболее популярный формат архива – ZIP-файл;
- **загрузка** – процесс передачи файла с другого компьютера на ваш компьютер. Обратным процессом является выгрузка файла;
- **FTP (File Transfer Protocol – протокол передачи файлов)** – примитивный способ передачи файлов между UNIX-компьютерами, а также метод передачи файлов через Internet;
- **GIF (Graphics Interchange Format – формат обмена графическими данными)** – специальный формат файлов для хранения графических изображений, пригодный для использования на всех компьютерах;
- **изображение** – графика на Web-странице;
- **JPEG (Joint Photographic Expert Group – объединенная экспертная группа по фотографии)** – универсальный графический стандарт, созданный на замену устаревшему CIF (до 16 миллионов цветов в одном графическом изображении);
- **концентратор** – центральное оборудование в сети. Все компьютеры подключаются к концентратору, используя специальные сетевые кабели;
- **подключение диска** – процесс добавления сетевого жесткого диска в вашу систему, чтобы ОС Windows сочла его диском вашего ПК. Жесткий диск **подключается** к вашему ПК, ему присваивается значок и буква в окне Мой компьютер;
- **сеть** – оборудование и программное обеспечение, которые позволяют многим компьютерам совместно использовать свои ресурсы;
- **сетевая карта** – карта расширения, которая вставляется в ваш компьютер и позволяет ему получить доступ к сети. Ваш компьютер соединяется с другими компьютерами в сети посредством сетевых кабелей;
- **администратор** – лицо, организующее совместное использование компьютера. Только администратор может добавлять или удалять пользователей, менять пароли, изменять какие-либо системные установки, имеющие отношение к пользователям, и осуществлять общую поддержку системы;
- **пароль** – секретное слово, без которого доступ в систему закрыт;
- **пользователь** – лицо с правом использования компьютера. Права и привилегии каждого пользователя определяет администратор;
- **Active Desktop (рабочий стол похож на Web-страницу)** – периодически взаимодействует с определенными Web-узлами с целью обновления информации;
- **Active Desktop Gallery** – витрина Internet-информации, можно найти конкретную информацию для рабочего стола;
- **команда копирования** – используется для создания копии файла, папки, изображения и т.д.;
- **ярлык** – средство для быстрого доступа к программе, папке или файлу;
- **локальный принтер** – принтер, подключенный непосредственно к вашему компьютеру;

- **модем** – элемент оборудования, позволяющий вашему компьютеру обмениваться информацией с использованием телефонной линии (пересылка факс-сообщений, работа в Internet);
- **сетевой принтер** – принтер, к которому могут иметь доступ другие компьютеры в сети;
- **резервное копирование** – процедура копирования файлов с жесткого диска на магнитную ленту (или какой-либо диск) для безопасности. Если с оригинальными файлами что-то случается, всегда можно воспользоваться резервной копией;
- **дисковое пространство** – место на жестком диске (либо свободное, либо в целом);
- **восстановление** – процедура, обратная резервному копированию. Восстановление файлов с помощью резервной копии – процесс копирования их назад на жесткий диск. Это делается в том случае, если файлы были удалены, либо восстановление всего диска, если с ним что-нибудь произошло;
- **инсталляционный диск** – гибкий диск или компакт-диск, с которого новая программа инсталлируется на жесткий диск вашего компьютера. Иногда с помощью инсталляционного диска можно и деинсталлировать компоненты Windows;
- **мастер компонентов Windows** – средство, позволяющее инсталлировать и деинсталлировать компоненты Windows;
- **фрагментация** – процесс разделения файла на небольшие части для более эффективного размещения на диске;
- **схема** – набор установок Windows, включающий цвета, шрифты, размеры и другую информацию, описывающую элементы экрана. Можно использовать уже готовые схемы, предлагаемые Windows, или создавать свои собственные;
- **программа сохранения экрана** – программа, запускающаяся после периода отсутствия активности и гасящая экран. На сегодня эти средства – дань традиции;
- шрифт – стиль, определяющий вид текста на экране или в печатном документе;
- **пункт** – мера высоты шрифта. В одном дюйме 72 пункта. Большинство пользователей используют шрифты размером 10 или 12 пунктов;
- **засечка** – декоративный элемент шрифта. Шрифт без засечек (sans serif) используется преимущественно для заголовков. Шрифт с засечками (serif) читается легче, поэтому обычно используется для текста;
- **драйвер устройства** – программа, позволяющая устройству (например, модему или принтеру) взаимодействовать с Windows;
- **оборудование**
- – устройства, которые можно подключить к компьютеру, чтобы расширить его возможности;
- **Plug and Play (включи и работай)** – технология, позволяющая компьютеру выявлять вновь подключенные устройства и автоматически их конфигурировать;
- **USB (Universal Serial Bus – универсальная последовательная шина)** – помимо поддержки стандарта Plug and Play, порты USB позволяют подключить до 127 устройств без необходимости завершать работу Windows и выключать компьютер;
- **системное табло** – область на правом конце панели задач, где отображаются индикатор времени, а также небольшие значки, позволяющие управлять различными программами или аспектами функционирования Windows.

5.2. Архитектура Windows

5.2.1. Режимы выполнения программного кода

Два режима: пользователя и ядра, четыре уровня привилегий (кольца) для защиты от менее привилегированного кода (модель защиты Intel).

Уровень привилегий 0, режим ядра, максимальный.

Уровень привилегий 3, режим пользователя, минимальный.

ОС Windows используют только 0 и 3 уровни.

Режим ядра (кольцо 0) – наиболее привилегированный режим:

- имеет прямой доступ к аппаратному обеспечению;
- имеет доступ ко всей памяти компьютера;
- не может быть вытеснен в страничный файл на жестком диске;
- выполняется с большим приоритетом, чем процессы режима пользователя.

Компоненты режима ядра защищены архитектурно, процессор предотвращает их изменение другой программой.

Процесс **режима пользователя** характеризуется следующим:

- не имеет прямого доступа к аппаратуре, это защищает систему от неисправных приложений или неавторизированного доступа;
- ограничен выделенным им адресным пространством. Этим обеспечивается целостность ОС;
- может быть вытеснен из физической памяти в виртуальную память на жестком диске. Пространство на диске используется как дополнительное ОЗУ;
- выполняется с меньшим приоритетом, чем ядро.

Процессы режима пользователя получают меньший доступ к процессу, чем процессы режима ядра. ОС не ожидает окончания выполнения приложения. Неисправный программный компонент не вызывает разрушения системы.

5.2.2. Многозадачность

Многозадачность – способность операционной системы обеспечить совместное использование процессора несколькими программами, т.е. выполнять более одной программы (задачи) одновременно. Рабочие программы можно назвать задачами.

Однозадачность – один процесс должен завершиться прежде, чем может начаться другой.

Процесс – выполняемая программа, ему принадлежит адресное пространство и выделенные ресурсы, а также один или более потоков, выполняющихся в его контексте. В Windows 2000 и UNIX загруженная в память программа называется **процессом**. В Windows 95 также применяется термин процесс. Термины процесс и задача можно считать синонимами.

Поток – основная единица, которой ОС выделяет процессное время, и минимальный квант кода, который может быть запланирован для выполнения. Поток – это часть процесса, выполняющаяся в данный момент времени. Поток работает в адресном пространстве процесса и использует ресурсы, выделенные процессу.

Любой процесс содержит хотя бы один поток, 16-разрядные приложения имеют один поток, 32-разрядные могут включать несколько потоков.

Ресурсами владеют процессы, а не потоки.

Корпоративная многозадачность – контроль над процессором никогда не отбирается у задачи, приложение должно самостоятельно отказаться от контроля над процессором, чтобы другое приложение заработало. Программа должна учитывать необходимость возврата управления процессором операционной системе, иначе ОС будет заблокирована.

Вытесняющая многозадачность – ОС получает контроль над процессором без согласия выполняющегося приложения.

С помощью планирования ОС определяет, какой поток использует процессор в данный момент времени. Каждому потоку присваивается приоритет. **Планирование** основано на заранее заданной единице времени – **кванте** (продолжительность кванта зависит от конфигурации системы). Уровни приоритетов – от 0 (наименьший) до 31 (наибольший). Поток с наибольшим приоритетом получает процессор в свое распоряжение.

Приоритет каждого потока определяется по:

- классу приоритета процесса, которому принадлежит поток;
- уровню приоритета потока внутри класса приоритета его процесса.

Уровни приоритетов Windows разделены на два **класса**:

- реального времени (приоритеты от 16 до 31) используются для выполнения основных функций ОС и обычно не применяются для приложений;
- переменного приоритета (от 0 до 15) – определяет процессорный приоритет приложения; приоритет 0 – для бесстраничного системного потока.

Базовые уровни приоритетов:

- низкий – запускает приложение с уровнем приоритета 4;
- обычный – запускает приложение с уровнем приоритета 7;
- высокий – запускает приложение с уровнем приоритета 13;
- реального времени – запускает приложение с уровнем приоритета 24.

5.2.3. Управление памятью

В Windows 9x и Windows 2000 каждый процесс имеет свое адресное пространство до 4 Гб памяти (не физическое ОЗУ). Физическая память ограничена системными ресурсами: ОЗУ и дисковым пространством. Windows выделяет приложению 2 Гб памяти, а остальные 2 Гб резервируются для нужд ядра.

Если объем ОЗУ меньше, чем 4 Гб, то Windows использует механизм **виртуальной памяти**: когда объем ОЗУ будет исчерпан, часть содержимого физической памяти переносится на жесткий диск. Этот механизм называется **подкачкой**.

Для каждого процесса ядро поддерживает **таблицу страниц** – структуру, позволяющую преобразовать виртуальные адреса в физические.

Виртуальная память Windows использует механизм отображения области физической памяти на любую область 32-разрядных адресов для того, чтобы любая программа как бы обладала своим собственным физическим ОЗУ.

Каждая программа имеет собственное виртуальное адресное пространство, которое диспетчер виртуальной памяти преобразует в адреса физического ОЗУ или в файлы на жестком диске.

Физическое и виртуальное (логическое) адресное пространство каждого процесса разделено на страницы – кванты памяти, размер которых зависит от компьютера. Ядро может перемещать страницы памяти в страничный файл на диске и обратно. Когда страница перемещается в физическую память, ядро обновляет таблицу страниц соответствующего процесса. Когда ядру требуется место в физической памяти, оно

вытесняет самые старые страницы физической памяти в страничный файл. Все это происходит незаметно для приложения.

5.2.4. Выполнение приложений

Windows 9x и Windows NT выполняют приложения по-разному, особенно 16-разрядные.

Механизм сообщений Windows используется для управления приложениями. Сообщение генерируется всякий раз, когда происходит событие, например, перемещение мыши. Сообщения помещаются в очередь сообщений. Активное приложение постоянно просматривает свою очередь и извлекает из нее поступившие сообщения.

Обмен сообщениями в Windows основан на том, что у каждого приложения своя очередь сообщений, т.е. каждый поток имеет собственную очередь сообщений и не влияет на поведение других работающих приложений. Если одно из приложений разрушится, то остальные будут выполняться на основе вытесняющей многозадачности (для Win32-приложений). 16-разрядные приложения используют общую очередь сообщений и в случае сбоя, пока проблема не будет решена, остальные приложения не получат доступа к очереди.

Windows NT выполняет приложение в рамках **виртуальной машины**. Фактически ВМ – среда для выполнения приложения, которая эмулирует все ресурсы компьютера. Для приложения ВМ – полноценный компьютер.

Каждая **виртуальная машина** включает в себя следующие компоненты:

- карту памяти, определяющую объем виртуальной памяти, доступный этой виртуальной машине;
- контекст выполнения, определяемый состоянием регистров виртуальной машины;
- набор ресурсов, доступных приложению.

Основные достоинства **виртуальной м**ашины Windows:

- виртуальная память, выделенная отдельной ВМ, изолирована от виртуальной памяти, выделенной другой ВМ;
- средства защиты памяти и портов ввода-вывода позволяют защитить каждое из входящих в систему устройств.

Драйвер устройства – программный компонент, получающий команды из ОС и преобразующий их в команды конкретным устройствам. Часто драйверы разрабатываются производителем устройства. Драйверы позволяют разрабатывать аппаратно независимые приложения. Компоненты, для которых используются драйверы:

- дисплеи;
- звуковые карты;
- устройства связи;
- принтеры;
- сетевые адаптеры.

Драйвер может принадлежать к одной из групп: **защищенного режима и реального режима**.

Драйверы реального режима созданы для работы в реальном режиме ОС MS-DOS. Они не так безопасны и устойчивы, как драйверы защищенного режима. Драйвер защищенного режима (виртуальный драйвер) обеспечивает быстрый разделяемый доступ к устройству. Код защищенного режима выполняется более эффективно.

Windows 9x поддерживает оба типа драйверов, а Windows NT только драйверы защищенного режима.

5.2.5. Интерфейс прикладного программирования Win32 (API Win32)

API Win32 обеспечивает доступ ко всем функциям ОС, позволяет разрабатывать приложения, работающие на всех платформах.

Основной код API Win32 содержится в трех библиотеках динамической загрузки:

USER32 (User32.dll и User.exe) создают и контролируют окна на экране.

GDI32 (Gdi32.dll и Gdi.exe) контролируют интерфейс графических устройств:
- вывод на экран;
- вывод на принтер;
- включение/отключение пикселов.

KERNEL32 (Kernel32.dll) выполняет базовые функции ОС:
- управление памятью;
- файловый ввод/вывод;
- загрузку программы;
- выполнение программы.

Операция **шлюзования** выполняется, когда ОС преобразует вызов 16-разрядной функции в вызов 32-разрядной. Процессы в Windows 9x и Windows NT не могут одновременно содержать 16-разрядный и 32-разрядный код.

5.2.6. Реестр Windows

Реестр – унифицированная база данных, содержащая информацию об аппаратной и программной конфигурации локального компьютера.

Редактор реестра REGEDIT.EXE позволяет просматривать и редактировать реестр Windows 9x и Windows NT. При ручном редактировании следует быть осторожным: редактор не распознает синтаксические и семантические ошибки и не предупреждает о создании некорректного элемента. Большинство параметров системы можно модифицировать через диспетчер устройств и др. панели управления.

Реестр – древовидная иерархическая база данных, хранится в двух файлах: USER.DAT – настройки для пользователя и SYSTEM.DAT – настройки для компьютера. Узел иерархического дерева называется **ключом**. Любой ключ может содержать вложенные ключи. В ключе хранится произвольное число значений данного типа, каждое значение называется **элементом реестра**. Компоненты ключей следующие:
- имя (уникально среди ключей того же уровня иерархии);
- класс (имя класса объекта);
- дескриптор защиты (для Windows NT и Windows 2000);
- время последней записи;
- элементы.

Список ключей:

HKEY_CLASSES_ROOT – сведения о встраивании и связывании объектов и ассоциации файлов с приложениями;

HKEY_LOCAL_MACHINE – спецификации рабочей станции, драйверов и другие системные настройки;

HKEY_CURRENT_CONFIG – информация о текущей конфигурации компьютера;

HKEY_USERS – информация обо всех пользователях данной рабочей станции;

HKEY_CURRENT_USER – настройки системы и программ, относящиеся к текущему пользователю;

HKEY_DYN_FATA – динамическая информация о состоянии различных устройств.

5.3. Вопросы к главе 5

1. Перечислите основные возможности Windows 3.1, Windows 9x, Windows NT и Windows 2000.
2. Как выбрать ОС, оптимальную для конкретной ситуации.
3. Как Windows 9x и Windows NT выполняют программный код?
4. Как Windows 9x и Windows NT реализуют вытесняющую многозадачность?
5. Как Windows 9x и Windows NT 9x и Windows NT управляют памятью?
6. Сходства и различия выполнения приложений в Windows 9x и Windows NT.
7. Как драйверы устройств обеспечивают независимость от аппаратуры?
8. Назначение и структуру реестра Windows.

6. Операционные системы типа UNIX

6.1. Общая характеристика операционных систем UNIX, особенности архитектуры семейства ОС UNIX

ОС UNIX – исключительно удачная реализация простой мультипрограммной и многопользовательской операционной системы. Первоначально ОС предназначалась для разработки программного обеспечения. ОС UNIX обладает простым, но очень мощным командным языком и независимой от устройств файловой системой. При создании ОС UNIX использовался язык высокого уровня С, поэтому системные и прикладные программы получились легко переносимыми (мобильными). Компилятор с языка С для всех оттранслированных программ дает реентерабельный и разделяемый код, что позволяет эффективно использовать имеющиеся в системе ресурсы.

При разработке ОС UNIX преследовались следующие цели:

- сохранить простоту и обойтись минимальным количеством функций;
- общность – одни и те же методы и механизмы должны были использоваться во многих случаях;
- создать операционную среду, в которой большие задачи можно решать, комбинируя небольшие программы, а не создавая программы заново.

Общность в ОС UNIX проявляется во многих аспектах:

- обращение к файлам, устройствам ввода/вывода и буферам межпроцессных сообщений выполняется с помощью одних и тех же средств;
- одни и те же механизмы именования, присвоения альтернативных имен и защиты от несанкционированного доступа применяются к файлам с данными, к каталогам и устройствам;
- одни и те же механизмы обслуживают программные и аппаратные прерывания.

6.2. Основные понятия системы UNIX

Основным достоинством ОС UNIX является то, что система базируется на небольшом числе понятий.

6.2.1. Виртуальная машина

ОС UNIX – многопользовательская система. Каждому пользователю после регистрации предоставляется виртуальный процессор, в котором есть все необходимые ресурсы:

- процессор (карусельная диспетчеризация RR, динамические приоритеты);
- память;
- устройства;
- файлы.

Текущее состояние такого виртуального компьютера называется образом. Процесс – выполнение образа. Образ состоит из следующих элементов:

- образа памяти;
- значений общих регистров процессора;
- состояния открытых файлов;
- текущего каталога и др.

Образ процесса во время его выполнения размещается в основной памяти. В современных реализациях, поддерживающих страничный механизм виртуальной памяти, прежде всего выгружаются неиспользуемые страницы.

Образ памяти делится на три логических сегмента:

1) сегмент реентерабельных процедур;
2) сегмент данных;
3) сегмент стека.

6.2.2. Пользователь

ОС UNIX предназначена для мультитерминальной работы. Чтобы начать работу пользователь должен «войти» в систему:

- ввести учетное имя;
- ввести пароль.

Пользователь называется зарегистрированным, если на него заведена соответствующая учетная запись в файле /etc/passwd. Регистрацию новых пользователей выполняет администратор системы. Пользователь не может изменить свою регистрационное имя, но может изменить пароль. Пароли хранятся в закодированном виде в файле /etc/shadow.

Файловая система ОС UNIX имеет древовидную структуру. Каждому зарегистрированному пользователю устанавливается некоторый каталог файловой системы, который называется «домашним» для данного пользователя. При удачной регистрации пользователя в системе, он попадает в свой «домашний» каталог.

Доступ пользователя к «чужим» файлам и каталогам ограничен установленными правами доступа к этим файлам и каталогам.

6.2.3. Интерфейс пользователя

После регистрации пользователя в ОС UNIX для его запускается один из командных интерпретаторов, который прописан в файле /etc/passwd. В системах UNIX поддерживаются несколько командных интерпретаторов с похожими, но различающимися возможностями. Общее название для любого командного интерпретатора – shell (оболочка).

Вызванный командный интерпретатор приглашает пользователя ввести команду, после выполнения которой снова выводится приглашение.

Командные языки достаточно просты и в то же время мощны.

6.2.4. Привилегированный пользователь

При регистрации пользователя в системе ему присваивается уникальный идентификатор (UID). Каждый пользователь относится к той или иной группе пользователей с идентификатором группы (GID). Сведения о группах пользователей и значения идентификаторов хранятся в файле /etc/group.

Значения идентификаторов UID и GID наследуются процессами, порожденными текущим пользователем.

Администратор системы также является зарегистрированным пользователем, но он должен обладать большими возможностями, чем обычный пользователь. В ОС UNIX суперпользователю root выделяется нулевое значение идентификатора. Пользователь с таким идентификатором имеет неограниченные права доступа к любому файлу и на выполнение любой программы. Суперпользователь имеет возможность полного контроля над системой.

На суперпользователя не действуют ограничения на использование ресурсов, такие как:

- максимальный размер файла;
- максимальное число сегментов разделяемой памяти;
- максимальное допустимое пространство на диске и т.д.

6.2.5. Команды и командный интерпретатор

Оболочкой shell в ОС UNIX называется механизм взаимодействия между пользователем и системой.

Командная строка состоит из имени команды, ключей (или опций) и аргументов, разделенных пробелами. Оболочка разбивает командную строку на компоненты.

Любой командный язык семейства shell состоит из трех частей:

1) служебных конструкций, позволяющих манипулировать с текстовыми строками и строить сложные команды на основе простых;
2) встроенных команд, выполняемых непосредственно интерпретатором командного языка;
3) команд, представляемых отдельными выполняемыми файлами.

6.2.6. Процессы

В ОС UNIX процесс – программа, выполняемая в собственном адресном пространстве. При удачной регистрации пользователя в системе автоматически создается процесс, в котором выполняется программа командного интерпретатора. Если командному интерпретатору встречается команда, соответствующая выполняемому файлу, то он создает новый процесс и запускает в нем эту команду.

6.3. Функционирование системы UNIX

6.3.1 Выполнение процессов

Процесс может выполняться в одном из двух состояний:

- пользовательском. Процесс выполняет пользовательскую программу и имеет доступ к пользовательскому сегменту данных;
- системном. Процесс выполняет программы ядра и имеет доступ к системному сегменту данных.

Когда пользовательскому процессу требуется выполнить системную функцию, он создает системный вызов. Фактически происходит вызов ядра системы как подпрограммы. С момента появления системного вызова процесс считается системным. Пользовательский и системный процессы являются двумя фазами одного и того же процесса, но они никогда не пересекаются между собой. Каждая фаза пользуется своим собственным стеком. Стек задачи содержит:

- аргументы;
- локальные переменные;
- другую информацию относительно функций, выполняемых в режиме задачи.

Диспетчерский процесс не имеет пользовательской фазы.

В ОС UNIX используется разделение времени, каждому процессу выделяется квант времени:

- процесс завершается сам до истечения отведенного ему кванта времени;
- процесс откладывается по истечении кванта времени.

Пользовательским процессам устанавливаются приоритеты в зависимости от количества получаемого ими процессорного времени:

- процессам, которые получили больше процессорного времени, назначаются более низкие приоритеты;
- процессам, которые получили небольшое количество процессорного времени, приоритет повышают.

Такой метод диспетчеризации обеспечивает хорошее время реакции для всех пользователей системы.

Все системные процессы имеют более высокие приоритеты по сравнению с пользовательскими и поэтому всегда обслуживаются в первую очередь.

6.3.2. Подсистема ввода/вывода

В ОС UNIX команды ввода/вывода, применяемые к файлам и физическим устройствам, одни и те же. Физические устройства представлены специальными файлами в единой структуре файловой системы. Пользователь не может написать зависящую от устройства программу. Стандартные файлы ввода и вывода, приписываемые пользовательскому терминалу, открывать не требуется.

Система ввода/вывода ОС UNIX, в отличие от большинства систем, ориентирована на работу с потоками, а не с записями. **Поток** – последовательность байтов, заканчивающаяся разделителем. Понятие потока позволяет проще добиться независимости от устройств и унификации файлов с физическими устройствами и конвейерами.

6.3.3. Перенаправление ввода/вывода

Механизм перенаправления ввода/вывода является одним из наиболее элегантных, мощных и одновременно простых механизмов в ОС UNIX. Для того чтобы обеспечить более гибкое использование программ ввода и вывода, желательно обеспечить им ввод из файла или из вывода других программ и направить их вывод в файл или на ввод других программ.

Реализация этого механизма основана на следующих свойствах ОС UNIX:

- любой ввод/вывод трактуется как ввод из некоторого файла и вывод в некоторый файл. Клавиатура и монитор тоже интерпретируются как файлы;
- доступ к любому файлу производится через его дескриптор:
  - файл с дескриптором 1 называется файлом стандартного ввода (stdin);
  - файл с дескриптором 2 называется файлом стандартного вывода (stdout);
  - файл с дескриптором 3 называется файлом стандартного вывода диагностических сообщений (stderr);
- программа, запущенная в некотором процессе, наследует от породившего процесса все дескрипторы открытых файлов.

Файл стандартного ввода – клавиатура, файлы стандартного вывода и вывода диагностических сообщений – экран терминала.

При запуске любой команды можно сообщить интерпретатору:

- какой файл или вывод какой программы должен служить файлом стандартного ввода для запускаемой программы:
- какой файл или ввод какой программы должен служить файлом стандартного вывода;
- какой файл или ввод какой программы должен служить файлом вывода диагностических сообщений.

Все, что требуется для нормального функционирования механизма перенаправления ввода/вывода, - придерживаться соглашения об использовании дескрипторов stdin, stdout и stderr.

6.4. Файловая система

Файл в ОС UNIX – множество символов с произвольным доступом. В файле содержатся произвольные данные, помещенные туда пользователем и ничего более.

6.4.1. Структура файловой системы

Информация на дисках размещается поблочно, по 512 байт в каждом блоке, блок равен сектору. Диск разбивается на следующие области:

- неиспользуемый блок;
- управляющий блок или суперблок, в котором содержится размер диска и границы других областей;
- i-список, состоящий из описаний файлов, называемых i-узлами;
- область для хранения содержимого файлов.

Каждый i-узел содержит:

- идентификационный номер владельца;
- идентификационный номер группы владельцев;
- права доступа;
- физические адреса на диске, где находится содержимое файла;

- размер файла;
- время создания файла;
- время последней модификации файла;
- время последнего изменения атрибутов;
- число ссылок на файл;
- тип файла: каталог, обычный файл или специальный файл.

Следом за i-списком идут блоки, предназначенные для хранения файлов. Пространство на диске, оставшееся свободным от файлов, образует связанный список свободных блоков.

6.4.2. Защита файлов

Защита файлов осуществляется при помощи идентификатора пользователя и десяти битов защиты – прав доступа. Права доступа подразделяются на три типа:

- чтение (**r**ead);
- запись (**w**rite);
- исполнение (e**x**ecute).

Эти права доступа могут быть предоставлены трем классам пользователей:

- владельцу файла;
- группе, в которую входит владелец;
- всем прочим пользователям.

Атрибуты доступа определяют, что разрешено делать с данным файлом данной категории пользователей.

При создании файла модифицируется не сам файл, а каталог, в котором появляются новые ссылки на узлы. Удаление файла заключается в удалении ссылки. Право на создание и удаление файла – это право на запись в каталог.

Право на выполнение каталога интерпретируется как право на поиск в нем, прохождение через него. Оно позволяет обратиться к файлу по пути, содержащему данный каталог, даже тогда, когда каталог не разрешено читать и список всех его файлов недоступен.

6.5. Межпроцессные коммуникации в UNIX

ОС UNIX в основе своей наиболее полно отвечает требованиям технологии «север – клиент». Для построения программных систем, работающих по принципу модели «сервер – клиент» в ОС UNIX существуют специальные механизмы, описанные ниже.

6.5.1. Сигналы

ОС UNIX, предоставляющая каждому пользователю виртуальный компьютер, поддерживает систему прерываний, отвечающую стандартным требованиям:

- обработка исключительных ситуаций;
- средства обработки внешних и внутренних прерываний;
- средства управления системой прерываний.

Всем этим требованиям в ОС UNIX отвечает техника сигналов, которая не только воспринимает и обрабатывает сигналы, но может их порождать и посылать на другие машины (процессы). Сигналы могут быть:

- синхронными, когда сам процесс инициирует сигнал;

- асинхронными, когда интерактивный пользователь за терминалом инициирует возникновение сигнала. Источником асинхронных прерываний может быть ядро, когда оно контролирует состояние аппаратуры.

Сигнал – простейшая форма межпроцессого взаимодействия, которое используется для передачи от одного процесса другому или ядра системы какому-либо процессу уведомления о возникновении определенного события.

6.5.2. Семафоры

Семафор – переменная определенного типа, которая доступна параллельным процессам для проведения над ней только двух операций:

- закрытия (P-операция);
- открытия (V-операция).

Семафор играет роль вспомогательного критического ресурса, так как операции P и V неделимы при своем выполнении и взаимно исключают друг друга.

Семафорный механизм работает по схеме, в которой сначала исследуется состояние критического ресурса, а затем уже осуществляется допуск к критическому ресурсу или отказ от него на некоторое время. При отказе доступа к критическому ресурсу используется режим «пассивного ожидания», поэтому в состав механизма включаются средства формирования и обслуживания очереди ожидающих процессов.

Основным достоинством семафорных операций является отсутствие состояния «активного ожидания», что может существенно повысить эффективность работы мультипрограммной вычислительной системы.

Операция P(S) проверяет текущее значение семафора S, и если оно меньше нуля, то осуществляется переход к следующей за примитивом операции, иначе процесс снимается на некоторое время с выполнения и переводится в состояние «пассивного ожидания». В этом состоянии ожидающий процесс не проверяет семафор непрерывно, поэтому на процессоре может выполняться другой полезный процесс.

Операция V(S) связана с увеличением значения семафора на единицу и переводом одного или нескольких процессов в состояние готовности к выполнению.

Операции P(S) и V(S) выполняются операционной системой в ответ на запрос, выданный некоторым процессом и содержащий имя семафора в качестве параметра.

Механизм семафоров, реализованный в ОС UNIX, является обобщением классического механизма семафоров общего вида. Семафор в ОС UNIX состоит из следующих элементов:

- значение семафора;
- идентификатор процесса, который хронологически последним работал с семафором;
- число процессов, ожидающих увеличения значения семафора;
- число процессов, ожидающих нулевого значения семафора.

Для работы с семафорами имеются следующие три системные вызова:

- создание и получение доступа к набору семафоров;
- манипулирование значениями семафоров (синхронизация процессов на основе использования семафоров);
- выполнение разнообразных управляющих операций над набором семафоров.

6.5.3. Программные каналы

Программный канал (конвейер, транспортер) является средством, с помощью которого можно производить обмен данными между процессами. Принцип работы

конвейера основан на механизме ввода/вывода, который используется для работы с файлами в ОС UNIX. Задача, передающая информацию, действует так, как будто она записывает данные в файл, а задача, для которой эта информация предназначена, читает ее из этого файла. Операции записи и чтения осуществляются не записями, а потоком байтов. Программный канал представляет собой поток данных между двумя (или более) процессами. Конвейеры не являются файлами на диске, а представляют собой буферную область.

Программные каналы (pipes) в ОС UNIX являются очень важным средством взаимодействия и синхронизации процессов. Теоретически программный канал позволяет взаимодействовать любому числу процессов, обеспечивая дисциплину FIFO (first-in-first-out). Процесс, читающий из программного канала, прочитает самые давние данные, записанные в программный канал. Традиционно для хранения данных использовались файлы, в современных версиях ОС UNIX применяются и другие средства, например, очереди сообщений.

В ОС UNIX различают два вида программных каналов:

- **именованный программный канал** может служить для общения и синхронизации произвольных процессов, знающих имя данного программного канала и имеющих соответствующие права доступа;
- **неименованным программным каналом** могут пользоваться только создавший его процесс и его потомки.

6.5.4. Очереди сообщений

Очереди сообщений (Queue) являются более сложным методом связи взаимодействующих процессов по сравнению с программными каналами. С помощью очередей также можно из одной или нескольких задач независимым образом посылать сообщения некоторой задаче-приемнику. При этом только процесс-приемник может читать и удалять сообщения из очереди, а процессы-клиенты имеют право лишь помещать в очередь свои сообщения. Очередь работает только в одном направлении, если необходима двухсторонняя связь, следует создать две очереди.

Работа с очередями сообщений имеет много отличий от работы с конвейерами:

- очереди сообщений предоставляют возможность использовать несколько дисциплин обработки сообщений:
  - FIFO – сообщение, записанное первым, будет прочитано первым;
  - LIFO – сообщение, записанное последним, будет прочитано первым;
  - приоритетная – сообщения читаются с учетом их приоритетов;
  - произвольный доступ – можно читать любое сообщение, а программный канал обеспечивает только дисциплину FIFO;
- при чтении сообщения из очереди оно не удаляется, а может быть прочитано несколько раз;
- в очередях реально присутствуют не сами сообщения, а только их адреса в памяти и размеры. Эта информация размещается системой в сегменте памяти, доступном для всех задач, общающихся с помощью данной очереди.

Каждый процесс, использующий очередь, должен предварительно получить разрешение на использование общего сегмента памяти с помощью системных запросов, потому что очередь – системный механизм и для работы с ней требуются системные ресурсы и обращение к самой ОС.

Для обеспечения возможности обмена сообщениями между процессами механизм очередей сообщений поддерживается следующими системными вызовами:

- образование новой очереди сообщений или получения дескриптора существующей очереди;
- посылка сообщения, т.е. постановка его в указанную очередь сообщений;
- прием сообщения, т.е. выборка сообщения из очереди сообщений;
- выполнение ряда управляющих действий.

Ядро хранит сообщения в виде связанного списка (очереди), а дескриптор очереди сообщений является индексом в массиве заголовков очередей сообщений.

6.5.5. Разделяемая память

Для работы с разделяемой памятью используются четыре системных вызов:

- создание нового сегмента разделяемой памяти или нахождение существующего сегмента с тем же ключом;
- подключение сегмента с указанным дескриптором к виртуальной памяти обращающегося процесса;
- отключение от виртуальной памяти ранее подключенного к ней сегмента с указанным виртуальным адресом начала;
- управление разнообразными параметрами, связанными с существующим сегментом.

После подключения сегмента разделяемой памяти к виртуальной памяти процесса этот процесс может обращаться к соответствующим элементам памяти с использованием обычных машинных команд чтения и записи, не прибегая к использованию дополнительных системных вызовов.

6.5.6. Вызовы удаленных процедур (RPC)

Во многих случаях взаимодействие процессов имеет характер «клиент – сервер». Один из процессов «клиент» запрашивает у другого процесса сервера» некоторую услугу и не продолжает свое выполнение, пока эта услуга не будет выполнена. По смыслу такой режим взаимодействия эквивалентен вызову процедуры, поэтому он так и назван. По идеологии ОС UNIX идеально соответствует требованиям сетевой операционной системы, поэтому на ее основе можно создавать распределенные системы и организовывать распределенное вычисление. Одной из основных требований к RPC является автоматическое преобразование форматов данных при взаимодействии процессов, выполняющихся на разнородных компьютерах.

Технология вызовов удаленных процедур (RPC – remote procedure call) должна обеспечить работу взаимодействующих процессов, находящихся на разных компьютерах.

В случае удаленного вызова передача параметров процедуре превращается в передачу запроса по сети.

Вызов удаленных процедур включает следующие шаги:

- процесс «клиент» производит локальный вызов процедуры, которую называют «заглушкой». Задачи «заглушки» следующие:
  - принять аргументы;
  - преобразовать аргументы в стандартную форму;
  - сформировать сетевой запрос. Упаковка аргументов и создание сетевого запроса называется сборкой;
- сетевой запрос пересылается на удаленную систему, где соответствующий модуль ожидает такой запрос и при его получении извлекает параметры вызова процедуры, а затем передает их серверу удаленной процедуры. После выполнения осуществляется обратная передача.

6.6 Основы работы в ОС UNIX

В этом разделе описывается, как использовать систему UNIX. Уделено внимание тому, как использовать клавиатуру, получить регистрационное имя, войти в систему и выйти из нее, ввести команды.

6.6.1 Доступ к системе UNIX

Чтобы установить контакт с системой UNIX необходимо иметь:

- терминал;
- регистрационное имя, которое идентифицирует вас как полномочного пользователя;
- пароль, который проверяет вас на идентичность;
- инструкции для диалога и доступа к системе UNIX, если ваш терминал напрямую не связан с компьютером.

Регистрационное имя - это имя, с помощью которого система UNIX проверяет, являетесь ли вы полномочным пользователем системы, во время запроса доступа к ней. Регистрационное имя вы должны вводить каждый раз, когда вы хотите войти в систему.

Чтобы получить регистрационное имя, обратитесь к администратору системы UNIX. Существует несколько правил выбора регистрационного имени. Обычно длина имени составляет от 3 до 8 символов. Оно может состоять из больших или маленьких букв, цифр, символа подчеркивания, но не может начинаться с цифры.

Однако ваше регистрационное имя, возможно, будет определяться конкретным применением. Примеры допустимых имен:

```
dko30101
mary2
jmrs
```

Если терминал напрямую связан с компьютером, то при включении в верхнем левом углу немедленно появится подсказка:

```
login:
```

Если в качестве терминала используется персональный компьютер, настроенный на работу в сети с помощью семейства протоколов TCP/IP, необходимо установить связь с компьютером, на котором установлена ОС UNIX. Это можно осуществлять несколькими способами, например, с помощью сетевого приложения **telnet**, которое имеется в операционных системах Windows 95/98//2000/NT или с помощью средств доступа к Internet. В каждом конкретном случае следует обратиться к соответствующей инструкции или получить информацию у преподавателя.

Например, в качестве терминала используется персональный компьютер с ОС Windows NT, сконфигурированной для работы в сети. Необходимо **зарегистрироваться** в UNIX-системе.

Требуемые исходные данные и действия:

- для установления связи с ОС UNIX необходимо знать сетевое имя удаленного компьютера или его IP-адрес (например, 192.168.2.19);

- найти в персональном компьютере приложение **telnet** и запустить его на исполнение;
- в открывшемся окне приложения выбрать пункт меню **Подключение**;
- ввести IP-адрес удаленной UNIX-системы (например, 192.168.2.19);
- установить по желанию характеристики терминала.

Когда появится подсказка login:, введите **регистрационное имя** и нажмите клавишу <RETURN>. Например, если ваше регистрационное имя dko30123, то строка регистрации будет выглядеть следующим образом:

```
login: dko30123<CR>
```

Замечание: здесь и далее по тексту – обычным шрифтом печатается информация, выводимая на экран терминала, а полужирным шрифтом – вводимая информация.

Если вы сделаете ошибку при вводе вашего регистрационного имени, то вы можете исправить ее с помощью символа @ или клавиши <BACKSPACE>.

Примечание. Помните, что вы должны вводить текст маленькими буквами. Если при вводе будете использовать большие буквы, то система UNIX будет ожидать при приеме только большие буквы, и посылать сообщения будет только большими буквами.

Теперь система выдает вам подсказку для ввода **пароля**. Введите пароль и нажмите клавишу <RETURN>. Если при вводе вы сделаете ошибку, то можете исправить ее с помощью клавиши <BACKSPACE> или символа @. Система UNIX не отображает ваш пароль на экране с целью безопасности.

Если регистрационное имя и пароль допустимы в системе UNIX, то система может вывести текущую информацию и затем подсказку команды.

Когда вы войдете в систему, то экран терминала будет выглядеть следующим образом:

```
login: dko30123
password:
. . . . . .
. . . . . .
$
```

Если вы сделаете ошибку при входе в систему, UNIX выведет сообщение:

```
login incorrect
```

Затем предоставит вам второй шанс войти в систему, выдав подсказку login:

Экран будет выглядеть следующим образом:

```
login: dko30123<CR>
password:
login incorrect
login:
```

Если вы никогда не были зарегистрированы в системе UNIX, то ваша процедура регистрации может отличаться от описанной выше. Это может произойти в том случае, если администратор системы предусмотрел процедуру назначения временных паролей

новым пользователям. Если вы имеете временный пароль, то система заставит выбрать новый пароль прежде, чем позволит зарегистрироваться.

Вынуждая вас выбирать новый пароль исключительно для вашего использования, система заботится о большей безопасности.

В общем, процедура входа в систему будет подобна следующей:

Вы устанавливаете контакт; система UNIX отображает **подсказку login:**. Введите ваше регистрационное имя и нажмите клавишу <RETURN>.

Система UNIX выводит **подсказку password:**. Введите ваш временный пароль и нажмите клавишу <RETURN>.

Система сообщит, что ваш временный пароль больше не действителен и предложит выбрать новый пароль.

Система предложит ввести ваш старый пароль. Введите временный пароль.

Система предложит ввести ваш новый пароль. Введите выбранный вами пароль.

Пароль должен соответствовать следующим требованиям:

- каждый пароль должен иметь, по крайней мере, 6 символов. Только первые 8 символов являются значимыми;
- каждый пароль должен содержать, по крайней мере, 2 буквенных символа и одну цифру или специальный символ. Буквенный символ может быть набран либо на регистре больших символов либо малых;
- каждый пароль должен отличаться от вашего регистрационного имени. Большие буквы и соответствующие им маленькие буквы эквивалентны;
- новый пароль должен отличаться от старого, по крайней мере, на три символа.

Примеры допустимых паролей:

```
bak84ch
dki40102
IGOR/,3S
```

Для проверки система просит вас заново ввести пароль. Введите снова пароль.

Если вы введете новый пароль второй раз не так как в первый раз, то система сообщит вам, что пароль не совпадает и предложит повторить процедуру регистрации снова. Когда пароли совпадут, система отобразит подсказку.

Следующий экран отображает описанную процедуру:

```
login: dko30123<CR>
password:<CR>
Your password has expired
Choose a new one
Old password:<CR>
New password:<CR>
Re-enter new password:<CR>
$
```

Когда системный администратор регистрирует пользователя в системе, с регистрационным именем связываются **две компоненты идентификации**:

- **идентификатор пользователя** (user ID - UID);
- **идентификатор группы**, к которой он относится (group ID -GID).

Имя пользователя связывается с **уникальным** числом. Система использует его как инструмент в различных механизмах защиты в ОС UNIX, например, при защите файлов или при выполнении привилегированных команд.

Примечание: В любой ОС UNIX имеется одно специальное имя **root**, принадлежащее т.н. суперпользователю, которое связано с UID= 0. Это означает, что пользователь имеет все системные привилегии.

Имя группы также связано с числом, которое обычно относится к группе пользователей, объединенных общими задачами, например, сотрудники отдела, студенты одного потока и т.д. Это число также используется механизмами защиты в системе. Если пользователь должен работать данными других групп, этот идентификатор связывается с именами других групп.

Вся регистрационная информация о пользователях системы хранится в файле **/etc/passwd**.

Структура и назначение полей файла /etc/passwd:

- регистрационное имя;
- не используется;
- идентификатор пользователя (UID);
- идентификатор группы пользователей (GID);
- комментарий;
- «домашний каталог»;
- стартовая программа.

Записи данных о каждой установленной группе содержатся в файле **/etc/group**, структура и назначение полей которого следующие:

- имя группы;
- идентификатор группы пользователей (GID);
- список дополнительных членов группы.

В современных версиях ОС UNIX зашифрованные пароли и относящаяся к ним системная информация хранятся в файле **/etc/shadow**, структура и назначение полей следующие:

- регистрационное имя;
- закодированный пароль;
- количество дней от 1.1.1970 до последнего изменения пароля;
- минимальный срок действия пароля (в днях);
- максимальный срок действия пароля (в днях);
- дополнительные данные.

В отличие от файлов, содержимое которого, как правило, доступно любому пользователю системы, содержимое файла в целях безопасности доступно только привилегированному пользователю (root).

Для определения собственных UID и GID, воспользуйтесь **командой id**.

Терминал является устройством ввода/вывода: вы используете его для ввода запросов системе UNIX, а система - для выдачи ответов вам. Видеотерминал отображает ввод и осуществляет вывод информации на экране дисплея.

При взаимодействии с системой UNIX вы должны быть осведомлены о **соглашениях по вводу**. Система UNIX требует, чтобы вы вводили команды маленькими буквами (за исключением некоторых команд, в которых присутствуют большие буквы). Другие соглашения позволяют вам выполнять задачи, такие как стереть буквы или удалить строку, нажав одну или две клавиши. Обратите внимание, что клавиши, связанные с каждой функцией, являются значениями по умолчанию. В большинстве случаев различные клавиши могут быть выбраны для выполнения этих функций. Подробное описание значения некоторых клавиш дано в последующих подразделах.

Соглашения по вводу следующие:

$ - подсказка системной оболочки (предлагает вам ввести вашу команду);

BACKSPACE> или <^**h**> - стереть символ;

<**BREAK**> -остановить выполнение программы или команды;

<**Del**> - удалить текущую командную строку;

<**Esc**> - когда используется с другим символом, то выполнить специальную функцию (называемую последовательностью переключения кода). Когда используется в режиме редактирования редактора vi, то означает конец режима ввода текста и возврат в командный режим;

< **RETURN** > - клавиша <RETURN>. Означает конец строки ввода и помещает курсор на новую строку;

<^**d**> - остановить ввод в систему или выйти из системы (завершить работу);

<^**h**> - возвратиться на один символ (для терминалов, у которых нет клавиши BACKSPACE;

<^**s**> - временный останов вывода на экран;

<^**g**> - продолжает вывод на экран информации, которая была остановлена при помощи <^s>.

Символ ^ означает управляющий символ <Ctrl>, то есть вы должны в этом случае нажать две клавиши одновременно: клавишу управляющего символа и указанную букву.

Что получится, если вы захотите использовать буквенное значение специальных символов? Так как по умолчанию система UNIX интерпретирует специальные символы как команды, то вы должны сказать системе, что нужно игнорировать специальные значения символов, если хотите использовать их как буквенные символы. Обратная косая черта (\) позволяет вам сделать это. Введите \ перед любым специальным символом, с которым вы хотите обращаться в его неизменном виде. Например, вы хотите вывести с помощью команды echo символ *. Чтобы предотвратить интерпретацию системой UNIX знака * как запрос на вывод имен файлов текущего каталога, поставьте обратную косую черту перед знаком *.

Появление на экране подсказки означает, что система UNIX распознала вас как полномочного пользователя и ждет от вас ввода команды.

В общем виде командная строка имеет следующую структуру:

имя [опции] [аргументы] < RETURN >

Имя команды, опции и аргументы должны отделяться друг от друга пробелом или знаком табуляции. Обработка командной строки интерпретатором начинается только после нажатия клавиши < RETURN >.

В дальнейшем изложении функций команд предполагается:

имя обязательный параметр команды

шаблон выбираемый параметр

[…] необязательный параметр команды

Опции:

- являются признаком модификации команды и, как правило, состоят из одного символа;
- большие и маленькие буквы означают разные модификации;
- как правило, опции начинаются с символа – (минус) который не отделяется пробелом от остальных символов;
- опции могут быть скомбинированы любым образом, при этом знак минус можно использовать только один раз.

Аргументы указывают объекты, которые должны обрабатываться командой, например:

- имя файла;
- номер процесса;
- данные.

Если вы знаете, что должна сделать вызываемая программа, но не уверены в правильном использовании синтаксиса, можно в командной строке после имени команды указать опцию -?:

Для того чтобы иметь возможность получения подробной информации, в ОС UNIX имеется встроенное руководство (on-line), доступ к которому обеспечивают **команды man** и **apropos**.

В простейшем случае, для получения информации о любой команде, необходимо указать ее имя в качестве аргумента **команды man**:

```
man имя_команды
```

Команда apropos выводит список команд в соответствии с ключевым словом (шаблоном), указанным в качестве аргумента команды:

```
apropos  шаблон
```

В тех случаях, когда истек минимальный срок действия пароля, с помощью **команды passwd** можно изменить **пароль**:

```
$ passwd
Setting password for user: dko30102
Old password:
New password:
Re-enter password:
$
```

Чтобы **завершить работу** с системой UNIX, введите **<^d**> в ответ на подсказку системы. Через несколько секунд система UNIX может отобразить подсказку login: вновь:

```
$ <^d>
login:
```

Это означает, что вы успешно завершили работу с системой, и она готова зарегистрировать нового пользователя.

Если вы регистрировались с удаленного терминала, произойдет разрыв связи, о чем вас уведомит, например, приложение telnet. Прежде чем вы отключите терминал, обязательно завершите работу с системой.

6.6.2. Файлы и каталоги

Как и во многих современных операционных системах, в ОС UNIX файлы организованы в виде древовидной структуры, называемой **файловой системой** (*file system*). Каждый файл имеет имя, определяющее его расположение в дереве файловой

системы. Корнем этого дерева является **корневой каталог** (*root directory*), имеющий имя "/".

Для ОС UNIX характерно, что в системе может присутствовать несколько файловых систем, которые могут иметь различную внутреннюю структуру, а файлы, принадлежащие этим файловым системам, могут быть расположены на различных устройствах. Имя файла является атрибутом файловой системы, а не набора данных на диске. Каждый файл в системе связан со своими метаданными, хранящимися в **индексных дескрипторах** (*inode*), которые содержат все характеристики файла, в том числе и указатели на дисковые блоки, где хранятся данные файла. Имя файла в файловой системе является указателем на его метаданные, в то время как метаданные не содержат указателя на имя файла.

В ОС UNIX используется несколько различных типов файлов, различающихся по функциональному назначению и действиям операционной системы при выполнении тех или иных операций над файлами.

Обычный файл (ordinary files) представляет собой наиболее общий тип файлов, содержащих данные. Для ОС они представляют собой неструктурированный набор данных. Интерпретация содержимого файла производится программой, обрабатывающей файл. К этим файлам относятся текстовые файлы, исполняемые программы, бинарные файлы и т.д.

Каталог (directory). С помощью каталогов формируется логическое дерево файловой системы. Каталог - это файл, содержащий имена находящихся в нем файлов и указатели на дополнительную информацию (метаданные), позволяющие операционной системе производить операции над этими файлами.

Специальный файл (special file) устройства обеспечивает доступ к физическому устройству. В UNIX различают символьные и блочные файлы устройств.

Символическая ссылка *(symbolic link).* Как было сказано выше, каталог содержит имена файлов и указатели на их метаданные. Сами метаданные не содержат ни имени файла, ни указателя на это имя. Это позволяет одному файлу иметь несколько имен в файловой системе.

Имена жестко связаны с метаданными и, соответственно с данными файла, в то время как сам файл существует независимо от того, как его называют в файловой системе. Такую ссылку можно создать с помощью команды ln (см. дальше).

К файловой системе применимы следующие понятия:

- **место расположения на жестком диске файловой системы (filesystem)** - область на диске, которая специально отформатирована, чтобы предоставить операционной системе осуществлять быструю адресацию и доступ к блокам диска, находящимся внутри этой области;
- **дерево файлов (file system)** - набор из одной или нескольких файловых систем, которые логически объединены в единую иерархическую древовидную структуру;
- **файл (file) -** именованный объект внутри файловой системы, в котором хранятся данные. Файл может быть пустым (т.е. не содержать данных), однако он предоставляет определенную информацию операционной системе;
- **каталог (directory) -** файл, который содержащий имена находящихся в нем файлов. Каталоги определяют положение файла в дереве файловой системы, поскольку сам файл не содержит информации о своем местонахождении;
- **Подкаталог (subdirectory) -** каталог, содержащийся внутри другого каталога. Каталоги, которые содержат подкаталоги, называются родительскими каталогами. Все каталоги имеют родительские подкаталоги, за исключением корневого каталога, который является сам себе родителем;

- **имя файла (filename) -** строка символов, которая создается в каталоге для идентификации файла;
- **путь (pathname) -** строка из одного или более имен файлов, объединенных символом "/". Путь специфицирует место расположения файла внутри древовидной файловой структуры (т.е. внутри файлового дерева).

ОС UNIX для работы с файлами не использует их **имя**. Вся информация, необходимая операционной системе располагается в массиве дескрипторов, в котором метаданные файла однозначно связаны с порядковым номером соответствующего дескриптора.

Имя файла хранится вместе с номером его дескриптора в файлах специального типа – каталогах.

Имена файлов ОС UNIX состоят из комбинации символов ASCII. Длина имени может достигать 255 символов (некоторые файловые системы ограничивают длину имени до 14 символов).

В именах не разрешается использовать символы, имеющие специальное назначение:

; | < > ` " ' $! % & * ? \ ( ) [ ]

Имена файлов, которые начинаются с символа (.) относятся к **скрытым (hidden) файлам**, которые не выводятся по умолчанию командой `ls`. Для того чтобы вывести имена скрытых файлов, необходимо использовать опцию **`-a`** в команде `ls`.

В именах файлов различаются заглавные и прописные буквы, поэтому имена Test и test относятся к различным файлам.

Файловая система ОС UNIX поддерживает несколько **типов файлов**:

- обычные файлы;
- каталоги;
- символические связи;
- специальные файлы устройств.

Если выполнить команду `ls -l`, которая выводит информацию о содержимом каталога, то можно увидеть, что первый символ первого слова каждой строки указывает на тип соответствующего файла. Значение символов следующее:

`-` - обычный файл;
`d` – каталог;
`l` – символическая связь;
`c,b` - файл устройства.

Обычный файл	**-** r w - r w - - - -	. . .
Каталог	**d** r w x r w x - - -	. . .
Символическая связь	**l** r w x r w x r w x	. . .
Файл устройства	**c** r w - r w - - - -	. . .

Обычные файлы содержат неструктурированную последовательность байтов. Приложение, работающее с таким файлом, определяет его структуру и содержание. Обычно такие файлы можно отнести к одной из следующих категорий:

- **текстовые, содержащие набор символов**. Например, письма, отчеты, командные файлы, используемые интерпретатором shell;
- **файлы, содержащие наборы числовых или текстовых данных какого-либо приложения,** например, электронные таблицы, базы данных или документы текстовых процессоров;

- **исполняемые программы в двоичном виде, содержащие машинные команды и данные,** например, программы, связанные с выполнением команд ОС UNIX или программы приложений.

Используя **команду file**, можно определить **тип файла**. Команда допускает использование нескольких аргументов, т.е. возможность определения типа нескольких разных файлов:

```
file имя_файла [ имя_файла …]
```

Каталоги являются специальными файлами, которые предназначены для организации иерархической структуры файловой системы.

Каталоги определяют положение файла в дереве файловой системы, поскольку сам файл не содержит информации о своем местонахождении. Подобно обычным файлам, каталоги содержат данные, однако, в отличие от обычных файлов, ядро накладывает ограничения на структуру этих данных: каталоги содержат для каждого файла данные в виде связки "**номер индексного дескриптора – имя файла**":

- номер индексного дескриптора используется в качестве индекса блока таблицы индексов, где содержится вся информация о файле;
- имя файла является текстовой информацией (ASCII). Каталог не может содержать одинаковые имена, относящиеся к нескольким файлам.

В качестве первого имени каждого каталога используется "точка" (**.**), это – синоним собственного имени каталога, в качестве второго имени используется "две точки" (**..**), это – синоним имени вышестоящего ("родительского") каталога.

Следует напомнить, что имена файлов, начинающиеся с символа "точка", являются скрытыми, они могут быть выведены командой `ls` только с опцией **–a**.

Когда добавляется новое имя файла, размер каталога автоматически увеличивается, однако, при удалении имени файла из каталога, размер каталога не уменьшается; ядро системы использует освободившуюся часть каталога для размещения вновь создаваемых записей имен файлов (точнее связок "номер индексного дескриптора – имя файла").

Специальный файл устройства обеспечивает доступ к устройству. В UNIX различают **символьные (character)** и **блочные (block)** файлы устройств.

Символьные файлы связаны с драйверами устройств, использующими небуферизированный обмен данными с устройствами, к которым относятся терминал, принтер, мышь.

Блочные файлы связаны с драйверами устройств, которые позволяют производить обмен данными в виде пакетов фиксированной длины – блоков.

Доступ к некоторым устройствам может осуществляться как через символьные, так и через блочные специальные файлы. Имена файлов устройств содержатся в каталоге /dev.

Например, чтобы вывести имя специального файла терминала, с которого Вы вводите команды, можно воспользоваться командой

```
$ tty
/dev/tty01
$
```

Символические связи – это специальные файлы, которые содержат в качестве данных символическую ссылку на другой файл. Эти данные указывают путь (*path*) к файлу внутри файловой системы. Ядро системы автоматически определяет по содержимому файла символической связи место расположения файла в файловом дереве.

Следует различать **жесткую связь** (hard link), которая является естественной формой связи, и **символическую связь** (soft link), связанную с особым типом файла и позволяющую косвенно адресовать файл.

Как было сказано выше, каталог содержит имена файлов и указатели на их метаданные. Сами метаданные не содержат ни имени файла, ни указателя на это имя. Это позволяет создавать в каталоге записи, в которых несколько различных имен будут связаны с одним и тем же номером индексного дескриптора. Такая связь называется жесткой связью.

Например, если в каталоге содержится имя файла file1, то создание еще одного имени file2, связанного с тем же номером дескриптора, что и file1, можно с помощью **команды ln**:

```
$ ln file1 file2
```

Наличие у файла нескольких имен можно определить для текущего каталога командой:

```
$ ls –l
. . .
. . .
-rw-r--r--  2  dmo202-02 dmo202 6344 Now 22 10:33  file1
-rw-r--r--  2  dmo202-02 dmo202 6344 Now 22 10:40  file2
. . .
$
```

Во второй колонке строки указано число жестких связей данного файла.

В отличие от жесткой связи, **символическая связь** адресует файл, который, в свою очередь, ссылается на другой файл. Данные файла, являющегося символической связью, содержат имя целевого файла. Такой файл создается командой ln с опцией -s:

```
$ pwd
/home/dk401/dk40101
$ ln –s file1 /home/dk401/dk40101/symfile1
$ cd /home/dk401/dk40101
$ ls –l
. . . . . .
. . . . . .
lrwxrwxrwx 1 dk40101 dk401 15 Now 22 10:33
file1 symfile1  -> /../../dk401/dk40101/file1
. . . . . .
$
```

При выводе содержимого файла symfile1 появятся данные файла file1.

В отличие от жесткой связи, действие которой распространяется только в пределах одной файловой системы, символическая связь может адресовать файл, находящийся в другой файловой системы.

6.6.3. Команды обращения к файловой системе

Когда пользователь регистрируется в операционной системе UNIX, его начинает обслуживать процесс (обычно оболочка shell), с помощью которого будут выполняться команды, вводимые пользователем. Кроме того, пользователь получает в свое распоряжение рабочий каталог, обычно называемый **домашним** каталогом.

Домашний каталог обычно содержит подкаталоги, файлы данных и файлы конфигурации рабочей среды пользователя. Домашние каталоги обычно размещаются в каталоге /home или его подкаталогах, например, /home/dk401/dk40102.

Вывод содержимого каталога можно получить с помощью **команды ls**, в качестве аргумента которой должно указываться имя файла:

ls *опция [имя_файла . . .]*

Имя файла, которое используется в качестве аргумента команды, может быть представлено в двух формах: абсолютной и относительной.

Абсолютное (absolute) имя, иногда называемое полным (full) именем, содержит полный путь - перечень всех имен каталогов от корневого до локального, в котором записано имя целевого файла, например, абсолютное имя файла `file2` в каталоге пользователя `dk40101`:

```
/home/dk401/dk40101/file2
```

Формально, полное имя файла представляет собой последовательность слов, разделенных символом /, каждый компонент имени, кроме последнего, является именем каталога. Абсолютное имя всегда начинается с символа /, представляющего корневой каталог общего логического дерева файловой системы.

Команда pwd выводит полное имя пути текущего каталога, например:

```
$ pwd
/home/dk401/dk40101
```

Относительное (relative) имя пути файла всегда указывает путь к файлу относительно текущего каталога. Для доступа к файлу или подкаталогу текущего каталога удобно использовать относительное имя пути. Например, если имя текущего каталога /home/dp401/dp40110, то вывод содержимого каталога /home/dp401/dp40110/tests/examples можно получить, используя относительное имя:

```
$ ls –l tests/txamples
```

При этом следует помнить:

- относительное имя никогда не начинается символом / (слеш);
- в имени пути можно использовать специальные имена каталогов (.) и (. .), которые являются, соответственно, именами текущего и вышестоящего (родительского) каталога.

Например, если имя текущего каталога /usr/sbin, и необходимо вывести содержимое каталога /usr/lib/mail, надо выполнить команду:

```
$ ls –l  ../lib/mail
```

Для **перехода** из текущего каталога в другой каталог используется **команда cd**, в качестве аргумента которой указывается имя (абсолютное или относительное) целевого каталога. Например, если текущим каталогом является /home/dk401, то для перехода в каталог /home/dk401/dk40101 надо выполнить команду:

```
$ cd dk40101
```

Чтобы перейти в каталог, который находится на два каталога выше текущего, можно использовать имя точка-точка (..):

```
$ cd ../..
```

Для **возвращения в домашний каталог** из любого текущего каталога файлового дерева, достаточно выполнить команду без аргументов:

```
$ cd
```

Если в команде указано имя обычного файла, то выводится сообщение об ошибке:

```
$ cd file1
sh: file1: not a directory
$
```

Текстовые файлы содержат символы, которые могут воспроизводиться устройствами ввода-вывода (печатные символы). Для файлов такого типа в ОС UNIX используются команды, позволяющие **просмотр текстовых файлов**: cat, more, pg, head, tail. Эти команды не используются для просмотра каталогов и бинарных файлов.

Формат **команды cat** следующий:

cat *имя_файла [имя_файла . . .]*

Содержимое каждого указанных файлов выводится в канал стандартного вывода (обычно на дисплей). Команда не имеет возможности остановки вывода по заполнению экрана, если содержимое файла превышает число строк экрана дисплея.

Для прекращения вывода следует использовать комбинацию клавиш: <Ctrl>s или <Ctrl>q.

Нажатие клавиши <Del> завершает команду и возвращает к текущему процессу sh.

Команды head и **tail** позволяют вывести только часть файла. Форматы команд следующие:

```
head [-число] имя_файла  [ имя_файла . . . ]
tail [-число] имя_файла  [ имя_файла . . . ]
```

Команда head выводит первые строки файлов в количестве, определенном опцией -число (по умолчанию 10 строк).

Команда tail выводит последние строки файлов в количестве, определенном опцией –число (по умолчанию 10 строк).

Команда more осуществляет **постраничный вывод текста** файла. Формат команды следующий:

more *имя_файла [имя_файла . . .]*

Подсказка, которая выводится в нижней строке экрана, указывает на часть просмотренного текста, выраженную в процентах:

```
-More- (xx%)
```

На подсказку можно ответить следующим образом:

- нажатием клавиши <Space> осуществляется вывод очередной страницы;
- нажатием клавиши <Enter> осуществляется вывод очередной строки;
- нажатием клавиши h осуществляется вывод окна подсказок;
- нажатием клавиши b осуществляется вывод предыдущей страницы;
- нажатием клавиши q осуществляется завершение команды.

Команда pg осуществляет **постраничный вывод текста** файла. Она с одинаковым успехом позволяет просмотр как предыдущей, так и последующей страницы текста. Формат команды следующий:

pg *имя_файла [имя_файла . . .]*

На подсказку (:) в нижней части экрана можно ответить:

- нажатием клавиши <Enter> - осуществляется вывод очередной страницы;
- нажатием клавиши h - осуществляется вывод окна подсказок;
- нажатием клавиши +n - осуществляется переход вперед на n страниц;
- нажатием клавиши -n - осуществляется переход назад на n страниц.

Команды hd и **od** используются в тех случаях, когда файл содержит информацию, которую нельзя представить в виде печатных символов, такой информацией может быть **набор двоичных данных**, или данных, содержащих управляющие символы.

Команда hd выводит данные в шестнадцатеричном формате.

Команда od выводит данные в восьмеричном формате.

Обе команды позволяют в качестве аргументов указывать несколько имен файлов.

6.6.4. Создание файлов и каталогов

Файлы в ОС UNIX создаются командами и утилитами, такими, например, как текстовый редактор, который позволяет создать файл и поместить вы него данные. В UNIX используется несколько различных типов файлов, различающихся по функциональному назначению и действиям операционной системы при выполнении тех или иных операций над файлами, однако, структура файлового дерева не зависит от типа файла. Существуют специальные команды, позволяющие отличать файлы от каталогов, создавать файлы и каталоги, переносить файлы, а также исследовать их содержимое.

Для создания **пустых** (не содержащих данные) **файлов**, используется **команда touch**:

touch *имя_файла [имя_файл ...]*

Если имя_файла является новым для каталога, то создается пустой файл.

Если имя_файла уже существует в каталоге, то его данные (по умолчанию) будут потеряны.

Для создания **новых каталогов** используется **команда mkdir**:

mkdir *имя_каталога [имя_каталога . . .]*

При создании новых каталогов, старайтесь следовать некоторые соглашения, например, для того, чтобы различать имена файлов и каталогов:

используйте заглавную букву на месте первой буквы имени каталога,

создавайте отдельные каталоги для файлов, связанных одной темой: один каталог для баз данных, другой для электронных таблиц, третий для текстовых файлов и т. д.

Опция –p команды mkdir позволяет создать дочерний каталог во вновь создаваемом каталоге:

mkdir *–p имя_каталога/имя_подкаталога . . .*

Для **копирования файлов** используется **команда cp**:

cp *имя_исходного_файла имя_конечного_файла*

Если конечный файл уже существует, то его содержимое меняется на содержимое исходного файла.

Команда может применяться как для копирования одного исходного файла в конечный файл, так и нескольких файлов:

cp *имя_исходного_файла имя_конечного_файла*
cp *имя_исходного_файла1 [имя_исходного_файла2 …]*
имя_конечного_файла

В первом случае, когда исходный файл копируется в конечный:

- если конечный файл является каталогом, то исходный файл копируется в этот каталог с тем же именем;
- если конечный файл является обычным файлом и файл с таким же именем существует, то содержимое исходного файла заменяет содержимое уже существующего файла.

Во втором случае все исходные файлы копируются с теми же именами в конечный файл, который должен быть каталогом.

В аргументах команды могут указываться как абсолютные, так и относительные имена файлов.

Если в качестве аргумента конечного файла указано имя каталога, то этот каталог должен уже существовать, так как команда cp не создает новых каталогов.

Для копирования файлов в текущий каталог можно использовать имя **.** (точка):

```
$ cp ../dp40101/file .
```

При отсутствии соответствующих прав доступа как к исходному, так и к конечному файлам, копирование не будет выполнено, о чем будет выдано сообщение об ошибке.

Перемещение файлов можно выполнить **командой mv**:

mv *имя_исходного_файла имя_конечного_файла*

Если конечный файл уже существует, то его содержимое меняется на содержимое исходного файла.

Команда может применяться как для перемещения одного исходного файла в конечный файл, так и нескольких файлов:

mv *имя_исходного_файла имя_конечного_файла*
mv *имя_исходного_файла1 [имя_исходного_файла2 …]*
имя_конечного_файла

В первом случае, когда исходный файл перемещается в конечный:

- если конечный файл является каталогом, то исходный файл перемещается в этот каталог с тем же именем;
- если конечный файл является обычным файлом и файл с таким же именем существует, то содержимое исходного файла заменяет содержимое уже существующего файла.

Во втором случае все исходные файлы перемещаются с теми же именами в конечный файл, который должен быть каталогом.

Если имена исходного и конечного файлов относятся к одному каталогу, то команда изменяет имя исходного файла на имя конечного:

```
mv oldname newname
```

Для **удаления файлов** используется **команда rm**:

rm *имя_файла [имя_файла …]*

Команда rm удаляет файлы, имена которых указаны в команде.

Весьма полезной практикой является проверять имена удаляемых файлов, для того, чтобы случайно не был потерян нужный файл. Это особенно полезно в тех случаях, когда удаляется сразу несколько файлов, имена которых явно не указаны в качестве аргументов команды.

Команда rm позволяет делать эту проверку в интерактивном режиме. Для этого используется опция –i:

```
$ rm –i *
file2  ?
$
```

В этом случае требуется отвечать на каждый запрос нажатием клавиш y<Enter>, если файл надо удалить или n<Enter>, если файл удалять не следует.

Команда rm по умолчанию удаляет только файлы, однако использование опции –r разрешает рекурсивно удалять файлы и каталоги, причем, не выдавая при этом никаких сообщений.

```
$ rm –r *
```

```
$
```

Эта команда удаляет все файлы текущего каталога и все файлы нижестоящих каталогов.

ВНИМАНИЕ! Будьте особенно осторожны при использовании приведенной выше команды!

Для **удаления каталогов** используется **команда rmdir**:

```
rmdir имя_каталога  [имя_каталога …]
```

Эта команда удаляет каталоги при условии, что они являются пустыми, т.е. не содержат записей с индексами файлов.

6.6.5. Работа с файлами

В ОС UNIX имеется много команд, которые помогают работать с **файлами**:

- осуществлять их поиск внутри файлового дерева;
- находить информацию внутри файла по шаблону;
- сравнивать содержимое двух файлов и т.д.

Ниже будет рассмотрена группа такого типа команд.

Для **поиска файлов и каталогов** используется **команда find**. Путем использования различных опций (в этой команде опции задаются не отдельными символами, как это делается в большинстве команд UNIX, а словами) можно указать различные критерии поиска, например, поиск по имени (-name), поиск по длине файла (-size), поиск по типу файла (-type) и т.д.

```
find  имя_каталога  расширение [расширение …]
```

Команда осуществляет поиск по маршруту от заданного имени_каталога по всем нижестоящим каталогам. Расширение содержит спецификацию критерия поиска и имен искомых файлов. Этот аргумент команды может содержать, также список действий, выполняемых с каждым найденным файлом. Список расширений анализируется слева направо и, если проверка по критерию дает истину, выполняется следующая проверка. Выражение **расширение** вычисляется как логическая операция **И**. Если условие не выполняется, проверка для текущего файла завершается и анализируется следующий файл.

Наиболее полезными являются –print (ее желательно указывать, если необходимо просмотреть полученные результаты) , -name и -type. Другими опциями пользуются, как правило, опытные пользователи и администраторы.

```
$ find . –name fil.text –print
```

Приведенная выше команда реализует поиск имен файлов file.text, начиная с текущего каталога и вывод результатов на стандартное устройство вывода.

Команда выводит сообщение об ошибке при попытке поиска файла в каталоге, доступ к которому данному процессу ограничен соответствующими правами.

Для **поиска строк информации файлов**, соответствующих шаблону (набору символов), используется **команда grep**:

```
grep [опции] шаблон имя_файла [имя_файла ...]
```

Шаблон, в общем виде, представляет собой регулярное выражение, которое может содержать метасимволы (для более подробного знакомства см. команду ed встроенного руководства). Однако шаблон может быть набором из алфавитно-цифровых символов.

```
$ grep dp40102 /etc/passwd
```

Приведенная выше команда выводит строки файла /etc/passwd, содержащие набор символов dp401-02.

Для **сравнения двух текстовых файла и определения различия между ними** используется **команда diff**:

diff *[опции] имя_файла1 имя_файла2*

Команда всегда предполагает, что необходим переход от *имя_файла1* к *имя_файла2*. Для различающихся строк файлов выводится информация. Строкам из файла *имя_файла1* предшествует символ <, а строкам из файла *имя_файла2* – символ >.

Менее информативная **команда cmp сравнивает данные двух файлов**.

cmp *[опции] имя_файла1 имя_файла2*

При этом файлы могут быть как текстовыми, так и не текстовыми. Команда сообщает о первом отличии между файлами, например:

```
$ cmp file1 file2
file1 file2 differ: char 5, line 1
$
```

Для **нумерации** строк текстового файла используется **команда nl**:

nl *[опция] имя_файла [имя_файла ...]*

Опциями файла можно задать шаг нумерации, начальное значение и т.д.

Для **подсчета** символов, слов и строк файла используется **команда wc**:

wc *[опция] имя_файла*

По умолчанию команда выводит имя файла и разделенные пробелами числа строк, слов и символов.

6.6.6. Управление правами доступа к файлам

Установка прав доступа к файлам в ОС UNIX позволяет эффективно защищать файлы от несанкционированного доступа к нему с целью изменения его содержимого, считывания из него информации или его исполнения. К файлами могут обращаться различные, с точки зрения UNIX, категории пользователей: владельцы файла, члены

группы, к которым принадлежит владелец и члены других групп зарегистрированных пользователей (в этом ряду отсутствует администратор системы, как пользователь имеющий все права доступа к любому файлу).

Рассмотрим результат вывода команды ls –l, с помощью которой имееися возможность вывода всех атрибутов файла. Например, для файла dat, находящегося в текущем каталоге, получен следующий результат:

```
$ ls –l dat
-rw-r--r--  1 dk30101 dk301 24 22 Sep 11:30 dat
```

Рассмотрим подробнее результат вывода:

- первое слово (-rw-r--r--) в первом символе информацию о типе файла, затем девятью символами определяются права доступа к файлу;
- второе слово (1) указывает число ссылок к файлу;
- третье слово (dk30101) определяет имя владельца файла;
- четвертое слово (dk301) определяет имя группы, к которой относится владелец файла;
- пятое слово (24) указывает на размер файла в байтах;
- далее (22 Sep 11:30) показывает дату и время создания файла (11:30) в часах и минутах;
- в конце строки указывается имя файла (dat).

Владельцем файла всегда является пользователь, процесс которого создал файл. Этот пользователь имеет привилегии в работе с созданным файлом. В частности, он может изменить права доступа к файлу, имя владельца и группы.

Изменить имя владельца может только администратор (root) или текущий владелец файла. Это он может сделать с помощью **команды chown**:

```
chown имя_владельца имя_файла [имя_файла ...]
```

Имя_владельца – это имя нового владельца файлов. После изменения имени старый владелец теряет свои привилегии.

Изменить имя группы владельца может администратор (**root**) или владелец файла. Это он может сделать с помощью **команды chgrp**:

```
chown имя_владельца имя_файла [имя_файла ...]
```

Пример

```
$ ls –l dat
-rw-r--r--  1 dk30101 dk301 24 22 Sep 11:30 dat
$ chgrp sys dat
$ chown root dat
$ ls –l dat
-rw-r--r--  1 root sys 24 22 Sep 11:30 dat
$
```

Права доступа, которые устанавливает владелец файла (или администратор) определяют, кто имеет **доступ к файлу** и что он **может сделать с этим файлом**.

Права доступа определяются для трех категорий пользователей:

- владельца (user);
- члена группы владельца (group);

- прочих пользователей (other).

Права доступа определяют три вида доступа:

- разрешение на чтение (read) файла (r);
- разрешение на запись (write) (w);
- разрешение на исполнение (execute) или поиск (search) (x).

Таким образом, девять символов, определяющих права доступа к файлу могут принимать значения r, w или x, если соответствующее право доступа установлено, и символ минус (-), если это запрещено, причем первые три символа слова прав доступа относятся к правам владельца, вторые – к правам членов группы, последние – к прочим:

```
r w x r w – r w –
```

Следует различать назначение прав доступа к файлам и каталогам.

Права доступа к файлу*:*

- **r**ead (r) дает возможность пользователю открыть файл и просмотреть его содержимое. Команды, требующие доступ к содержимому файла, требуют права доступа на чтение, например, команды cat, more, cp;
- **w**rite (w) позволяет изменять содержимое файла. Команды, требующие доступ к изменению файла, требуют права доступа на запись, например, команды vi, mail, cp;
- e**x**ecute (x) дает возможность пользователю выполнить двоичную программу или командный файл подобно любой команде ОС UNIX.

Права доступа к каталогу:

- **r**ead (r) дает возможность пользователю вывести список имен файлов и каталогов (прочитать содержимое каталога);
- **w**rite (w) позволяет изменять содержимое каталога, т.е. добавить или удалить имя (файла или каталога) из каталога. Обычно только владелец каталога имеет разрешение на запись в каталог;
- e**x**ecute (x) дает возможность осуществить поиск по каталогу. Без этого права доступа невозможно использовать имя каталога как компоненту полного имени файла. Отсутствие права на исполнение не позволит сменить каталог при выполнении команды cd.

Права доступа могут **модифицироваться** владельцем файла или пользователем root с помощью **команды chmod**:

```
chmod режим [режим …] имя_файла  [имя_файла …]
```

Полномочия доступа можно задавать **абсолютно** – путем спецификации режима в восьмеричном представлении – или **символически**, задавая отдельные спецификации. Последний способ более популярен и позволяет добавлять и удалять права доступа выборочно.

Изменение прав доступа - **символический режим**

Режим состоит из следующих компонентов:

```
[ugoa] [+|-|=] [[r] [w] [x]]
```

Первая компонента может состоять из одного или более символов – u (user), g (group), o (other) или a (all), указывающих на категорию пользователей, для которой устанавливаются права доступа, владелец (u), группа (g), прочие (o) или все (a).По умолчанию используется a.

Вторая компонента должна быть одним из трех символов +, - или =, означающих, соответственно, добавление полномочия для заданного класса пользователей, удаление полномочия для заданного класса пользователей или установку полномочия для заданного класса пользователей и отмене всех других не специфицированных полномочий.

Третья компонента может быть любой комбинацией символов r, w или x соответствующих разрешениям на чтение, запись или выполнение/поиск объекта файловой системы.

Пробелы между компонентами выражения не допускаются.

Изменение прав доступа - **абсолютный режим**

В абсолютный режиме слово режим команды chmod представляет собой три (или более) восьмеричные цифры, каждая из которых отражает права доступа для соответствующей категории пользователей (владелец, группа, прочие).

Цифра определяет соответствующее полномочие: r=4, w=2, x=1 для каждой из категорий пользователей – u, g, o, т.е. устанавливает соответствующий бит в слове прав доступа файла (см. команду ls –l). Значение 1 бита означает разрешение соответствующего права, 0 – запрещение.

ПРИМЕЧАНИЕ. Абсолютная форма не допускает выборочной установки прав доступа и требует указания о соответствующих правах явно сразу для всех категорий пользователей.

Примеры символического и абсолютного режима установки прав доступа:

Символическая форма:

```
$ ls –l script
-rwxrwxr--  1 dko30101    dko301    23   Okt  16 14:25 script
$
```

Владелец и группа имеют права доступа на чтение, запись и выполнение, а прочие только разрешение на чтение.

```
$ chmod g-w, o+x-r script
-rwxr-xr-x  1 dko30101  dko301 23   Okt  16 14:25  script
$
```

В результате владелец сохранил права доступа на чтение, запись и выполнение, группа утратила право доступа на чтение, а прочие утратили разрешение на чтение, но получили право на исполнение файла.

Абсолютная форма:

```
$ ls –l script
-rwxrwxr-- 1 dko30101 dko301   23 Okt 16 14:25     script
  ↓  ↓  ↓
```

```
  7  7  4

$ chmod 755 script

-rwxr-xr-x 1 dko30101 dko301   23 Okt 16 14:25     script
  ↓  ↓  ↓
  7  5  5
```

Пользователь может управлять **установкой прав доступа к файлам и каталогам**, когда они создаются, с помощью **команды umask**, которая позволяет вывести или изменить текущую маску создания файла, эта маска определяет заданные по умолчанию права доступа, устанавливаемые при создании нового файла или каталога:

umask *[маска]*

Маска представляет собой восьмеричное число из трех цифр, где каждая цифра определяет права доступа владельца, группы и всех прочих пользователей. Действие этой маски для каждой категории пользователей следующее: если рассматривать восьмеричное значение маски как двоичное число, то биты маски, равные 0 будут соответствовать разрешению доступа, а 1 запрещению доступа. Однако действие маски различно по отношению к файлам и каталогам.

Обычно по умолчанию (это определяет администратор системы) вновь создаваемым файлам устанавливаются права доступа 666 (rw-rw-rw-), а каталогам 755 (rwxr-xr-x).

Чтобы вывести на экран установленную маску, можно выполнить команду без аргументов:

```
$ umask
022
$
```

Чтобы у вновь создаваемых файлов для категории прочие было отобрано право на запись, команда должна иметь следующий вид:

```
$ umask 002
$
```

Связь маски команды с правами доступа (значение маски – право доступа к файлу – право доступа к каталогу – описание):

- 0 (rw-) (rwx) предоставляет права чтения и записи файлам, записи и поиска – каталогам;
- 1 (rw-) (rw-) предоставляет права чтения и записи файлам и каталогам;
- 2 (r--) (r-x) предоставляет права чтения файлам, чтения и поиска – каталогам;
- 3 (r--) (r--) предоставляет права только чтения файлам и каталогам;
- 4 (-w-) (-wx) предоставляет права записи файлам, записи и поиска – каталогам;
- 5 (-w-) (-w-) предоставляет права записи файлам, и каталогам;

6 (---) (--x) не предоставляет никаких прав доступа файлам, представляет право поиска каталогам;
7 (---) (---) отменяет все права доступа к файлам и каталогам.

6.6.7. Работа с текстовыми файлами

Текстовые файлы относятся к категории обычных файлов, которые содержат информацию, отображаемую в виде печатных символов. В ОС Unix используется множество подобных файлов, в которых содержится информация о конфигурации системы, процедурах, именах и паролях пользователей и т.д. Любой пользователь имеет возможность создания и модификации (редактирования) текстового файла в своем регистрационном каталоге, который будет содержать собственные данные пользователя. Для этого используются программы – редакторы текстов.

Текстовым редактором является программа, используемая для записи и модификации текстового файла. В отличие от текстового процессора (word processor), текстовый редактор обычно не содержит функций форматирования текста. В этом есть определенный смысл, поскольку редактор становится более быстрым и удобным в использовании и, что особенно важно при написании текстов программ, не вносит в текст файла служебных (управляющих) знаков.

В ОС Unix имеется несколько текстовых редакторов:

- ed;
- ex (extended editor -расширенный редактор);
- семейство редакторов vi.

В любой системе Unix всегда можно найти старый надежный редактор ed (ex), базовые команды которого легко освоить и применять для простых операций создания текстовых файлов и их исправления. Редактор ed (ex) является основой многих других текстовых редакторов, в частности, редакторов семейства vi.

В семейство редакторов vi входят такие редакторы, как viev, vedit и сам vi. Все они выполняют полный вывод текста на экран и позволяют управлять перемещением курсора на экране. Все редакторы используют одну и ту же структуру команд, однако, имеют следующие различия:

view используется только для вывода текстового файла на экран, его просмотра или для режима поиска данных, при этом невозможно внести изменения в файл;

vedit это версия экранного редактора vi, предназначенная для неопытных пользователей; по умолчанию в нее включен режим пояснений;

ex строчно-ориентированный редактор, не имеющий функций управления экраном.

Экранный редактор vi имеет **три** основных рабочих режима:

- режим ввода текста;
- командный режим;
- режим перехода в ex.

В **режиме ввода текста** вводимые с клавиатуры символы поступают во временный файл – буфер редактирования. При этом введенные символы появляются на экране. В этом режиме выполняются четыре функции:

- вставки (например, в любом месте строки);
- добавления (например, после позиции курсора);
- изменения (например, замена одного символа на другой);
- открытия (например, создание пустой строки).

Нажатие клавиши <Esc> приводит к выходу из режима ввода текста.

В **командном режиме** нажимаемые клавиши интерпретируются как команды редактирования vi. Вводимые команды не отражаются на экране. При нажатии <Esc> происходит возвращение в командный режим.

В **режиме перехода в ex** ввод с клавиатуры интерпретируется как команда редактора ex. При этом в начале строки состояния (в нижней части экрана) выводится приглашение на ввод команды (символ двоеточие –":"). После двоеточия выводится курсор. Вводимые команды отображаются на экране сразу, но действие их начинается после нажатия клавиш <Return> или <Esc>.

Синтаксис командной строки **вызова** редактора:

```
vi [имя_файла]
```

Если **создается новый файл**, то на экране (речь идет об экране, размером 80 на 25 символов) будут выводиться 23 символа "~" (тильда), расположенных в первой позиции каждой строки (со 2 по 24), в строке 25 (строка состояния) появляется соответствующее сообщение (например, "text.data" [New file]). Курсор будет находиться в позиции 1 строки 1. Тильды – это просто маркеры, указывающие пустые строки (т.е. строки, не содержащие никакой информации, включая знаки пробела или табуляции). Они не сохраняются в файле и последовательно исчезают по мере ввода данных. Строка состояния используется не только для сообщения о различных условиях, но и для ввода команд редактора ex.

Если **файл существует**, то на экране отобразится начало текста файла.

Для того чтобы выводилась **информация об установленном режиме**, необходимо перейти в командный режим (нажать клавишу <Esc>) и ввести:

```
:set showmode
```

В редакторе vedit эта функция устанавливается автоматически.

Редактор vi создает временную копию файла. Измененный текст, который можно видеть на экране, будет записан в указанный файл только при выполнении команды сохранения файла. Однако можно завершить работу и выйти из редактора без сохранения изменений в тексте файла.

Для выполнения функций **сохранения текста и выхода из редактора**, необходимо перейти из режима ввода текста в командный режим (нажать клавишу <Esc>) и выполнить соответствующую команду:

:q выйти из редактора, если текст не вводился;

:q! выйти из редактора без сохранения изменений, даже если они были сделаны;

:w записать (сохранить) текст в файл;

:wq сохранить текст в файле, затем выйти из редактора;

ZZ сохранить файл, если в нем были сделаны любые изменения, затем выйти из редактора;

:x же, что ZZ;

:w имя_файла сохранить текст в файле с новым именем (имя_файла).

Редактор vi является полноэкранным редактором, поэтому имеется возможность **перемещения курсора** в ту точку экрана, где необходимо поместить текст или сделать

изменения в тексте. Для этого можно воспользоваться клавишами со стрелками или вводом соответствующей команды в режиме ввода (нажатием клавиши <Esc>):

h или (стрелка влево) перемещает курсор на один символ влево;
l или (стрелка влево) перемещает курсор на один символ вправо;
k или (стрелка влево) перемещает курсор на строку вверх;
j или (стрелка влево) перемещает курсор на строку вниз;
w перемещает курсор на одно слово вправо;
b перемещает курсор на одно слово влево;
$ перемещает курсор в конец текущей строки;
0(нуль) или ^ перемещает курсор в начало текущей строки;
<Ctrl>u перемещает курсор вверх на половину экрана;
<Ctrl>d перемещает курсор вниз на половину экрана;
<Ctrl>b перемещает курсор назад на полный экран;
<Ctrl>f перемещает курсор вперед на полный экран.

Для **ввода нового текста**, необходимо установить курсор в позицию на экране дисплея, куда необходимо ввести текст. Затем ввести команду, не завершая ввода нажатием клавиши <Return>:

i ввод текста перед позицией курсора;
a ввод текста после позиции курсора;
o создание новой строки ниже позиции курсора и ввод с начала новой строки;
I вставляет текст в первую позицию строки;
A добавляет текст в конец строки;
O создание новой строки выше позиции курсора и ввод с начала новой.

После завершения ввода необходимо нажать клавишу <Esc> перед выполнением следующей команды перемещения курсора.

Команды **удаления символа, слова или строки** текста выполняются в режиме ввода:

x удаляет символ в позиции курсора;
X удаляет символ перед позицией курсора;
dw удаляет слово;
dd удаляет строку, в которой находится курсор;
d$ или D удаляет текст от позиции курсора до конца строки;
d0 или d^ удаляет текст от начала строки до позиции курсора.

После завершения ввода необходимо нажать клавишу <Esc> перед выполнением следующей команды перемещения курсора.

Следующая группа команд используется для **замены существующего текста**:

r заменяет символ, на который указывает курсор, новым символом; после этого редактор переключается в командный режим без нажатия клавиши <Esc>;
R заменяет текст с позиции курсора до тех пор, пока не будет нажата клавиша <Esc>;
s заменяет символ в позиции курсора одним или несколькими символами;
S заменяет текущую строку новым текстом;
cw заменяет слово;

cc или S заменяет текущую строку новым текстом;
c$ или C заменяет текст от позиции курсора до конца строки;
c0 или c^ заменяет текст от начала строки до позиции курсора/

После завершения ввода необходимо нажать клавишу <Esc> перед выполнением следующей команды перемещения курсора.

Подобно текстовым процессорам, в редакторе vi имеются **функции вырезания, копирования и вставки текста**. При выполнении команд x или d удаленные данные помещаются во временный буфер. Для копирования текста во временный буфер выполняется командой y. Вставка текста из буфера выполняется командой p.

Следует обратить внимание, что содержимое временного буфера сохраняется только до следующей команды редактирования, затем оно заменяется текстом следующей команды. Командой u можно отменить предшествующее редактирование.

yw копирует слово в буфер;
yy или Y копирует строку в буфер;
y$ копирует в буфер текст от позиции курсора до конца строки;
y0 или y^ копирует в буфер текст от начала строки до позиции курсора;
p вставляет текст из буфера в новую строку ниже позиции курсора или после позиции курсора, в зависимости от того, что находится в буфере: полная строка или ее часть;
P вставляет текст из буфера в новую строку выше позиции курсора или перед позицией курсора, в зависимости от того, что находится в буфере: полная строка или ее часть;
u отменяет последнее редактирование;
U восстанавливает текущую строку, отменяя все изменения.

Следует напомнить, что **vi** и **ex** являются **редакторами одного типа**, но используют различные режимы работы:

vi использует экранный режим;
ex – строчный.

Переход в режим редактора ex из vi осуществляется с помощью команды (<Esc>:). Такая возможность была описана выше, при рассмотрении команд :w, :wq, :set showmode и др. Ниже представлено несколько других полезных команд редактора ex:

:! команда команда ОС Unix выполняется без выхода из редактора;
:r имя_файла содержимое указанного файла записывается в текущий файл в строку ниже позиции курсора;
:r команда читает информацию из стандартного канала вывода команды ОС Unix и записывается в текущий файл в строку ниже позиции.

Команды редактора vi используют **ряд опций**, которые **влияют на результаты его работы**. Эти опции могут быть установлены заранее или в процессе работы с редактором. Существует два типа таких опций:

- опции переключения (включить/выключить);
- строчные опции, которым могут присваиваться числовые значения.

Установка опций осуществляется командой set в режиме переключения в ex (<Esc>:).

Опции переключения устанавливаются командой:

```
:set имя_опции
```

Действие опций отменяется добавлением к имени опции префикса no (без пробела):

```
:set no имя_опции
```

Строчные опции или комбинация разных типов опций устанавливаются командами:

```
:set имя_опции=значение
:set имя_опции=значение имя_опции=значение …
```

Для просмотра всех установленных опций используется команда

```
:set all
```

Ниже приведены некоторые часто используемые опции:

number указывает на то, чтобы все строки текста были пронумерованы

showmode устанавливает подсказку о режимах работа редактора в нижней части экрана

tabstop=n устанавливает число пробелов в знаке табуляции (по умолчанию 8)

warpmargin=n определяет числом правую границу текста от края экрана (по умолчанию 0).

Для автоматической установки опций редактор vi использует файл .exrc. Это скрытый файл, который пользователь должен создать в своем регистрационном (домашнем) каталоге, может содержать любые команды set редактора. Эти команды выполняются при каждом вызове редактора. Файл .exrc не может содержать пустых строк.

6.6.8. Система ввода и вывода

Команды и утилиты операционной системы Unix вводятся с клавиатуры, а результаты их выполнения выводятся на дисплей терминала, которые часто называют стандартными устройствами ввода/вывода. Некоторых команды, такие как pr или mail перехватывают вывод на стандартные устройства и посылают данные в файл или на устройство печати. Unix не только обеспечивает эту возможность, но и позволяет соединять команды таким образом, чтобы канал вывода одной команды был связан с каналом ввода другой.

Когда программа вызывается на исполнение, создается процесс, который использует **каналы** ввода и вывода. Эти каналы, называемые **дескрипторами файла**, ссылаются на файлы, которые процесс должен иметь открытыми для использования. Когда процесс запускается, всегда три дескриптора для него являются открытыми - они открываются процессом-родителем текущего процесса (например, процессом shell):

- канал стандартного ввода;
- канал стандартного вывода;
- канал стандартного вывода сообщений об ошибках.

Введите команду cat без аргументов. Обратите внимание на курсор. Наберите несколько строк текста, используя клавишу <Enter>.Что происходит? Нажатие клавиш <Ctrl>**d** завершит выполнение команды и вернет управление процессу shell.

Направление потока вводимых или выводимых данных может быть изменено таким образом, что ввод данных будет осуществляться из обычного файла или любого устройства вместо клавиатуры, а вывод в обычный файл или на другое устройство вместо дисплея терминала.

Оболочки Unix, такие как Bourne shell (sh), Korn shell (ksh) или C shell (csh) обеспечивают возможность перенаправления стандартного ввода, стандартного вывода и стандартного вывода сообщений об ошибках. Это достигается путем использование специальных синтаксических конструкций в командной строке. Важно, что все оболочки устанавливают признак перенаправления ввода/вывода перед выполнением команды. В результате команда даже "не знает", что это имело место, и что stdout будет ссылаться на файл вместо терминала. Здесь следует напомнить, что процесс обычно имеет два дескриптора файла, stdout и stderr и оба они ссылаются на дисплей терминала. Зачем два дескриптора относятся к одному устройству? Смысл этого заключается в том, что stderr обычно не требует перенаправления, поскольку все диагностические сообщения могут быть очень важны для пользователя. Однако если у пользователя есть необходимость в перенаправлении stderr, он также имеет возможность сделать это, с помощью shell.

В командной строке канал stdout можно перенаправить, используя метасимвол **>**, следующим образом:

команда *> имя_файла*
команда *1>имя_файла*

Имя файла должно быть указано непосредственно после символа >, пробел между ними необязателен.

Номер дескриптора указывается непосредственно перед символом >, пробел между ними не допускается. Если номер дескриптора опущен, по умолчанию его номер - 1.

Символ перенаправления и, связанное с ним имя файла, могут быть в любом месте командной строки, однако такую синтаксическую конструкцию принято помещать в конце командной строки.

Например, для решения задачи объединения содержимого файлов /etc/passwd и /etc/group и сохранение результата в файле mix следующие командные строки будут эквивалентны:

```
$ cat /etc/passwd /etc/group >mix
$ cat >mix /etc/passwd /etc/group
```

Если файл, имя которого указано после символа перенаправления не существует, то он сначала создается оболочкой, а затем в него будет перенаправляться вывод.

Если файл уже существует, то его содержимое уничтожается и замещается выводом выполняемой команды.

Перенаправление ввода/вывода в командной строке может быть определено без имени команды. В этом случае shell создаст пустой файл или удалит содержимое существующего файла с указанным именем.

```
$ >filename
```

Перенаправление стандартного вывода сообщений об ошибках stderr реализуется тем же путем, что и для stdout, т.е. использованием символа > в командной строке:

команда *2>имя_файла*

Номер дескриптора указывается непосредственно перед символом >, пробел между ними не допускается. В отличие от stdout, номер дескриптора стандартного вывода сообщений об ошибках – 2, должен обязательно указываться перед символом перенаправления.

Сохраненные в файле сообщения могут быть в дальнейшем использованы для анализа ситуации, вызвавшей ошибку в программе.

В одной командной строке можно указать сразу два символа перенаправления для stdout и stderr:

команда 1> имя_файла1 2> имя_файла2

Имеется возможность перенаправить stdout и stderr в один и тот же файл. Для этого необходимо использовать специальное обозначение:

команда > *имя_файла 2>&1*

2>&1 в данном случае означает следующее: перенаправить stderr туда же, куда был направлен stdout (&1)

Выполните команду:

$ **find / -name fff >fff_file 2>errors**

Где находятся сообщения об ошибках?

Что будет сделано следующей командой?

$ **find . 2>error_file >2 &**

Обычное перенаправление бывает разрушительным для существующего файла. Из предыдущего материала видно, что содержимое файла заменяется или удаляется. Shell обеспечивает возможность дополнения содержимого существующего файла новыми данными.

Символ дополнения (>>) используется для ввода новых данных, которые будут размещаться в конце существующего файла.

Для stdout используется синтаксическая конструкция:

command >> filename
command 1>> filename

Для stderr используется следующий синтаксис:

command 2>> filename

Если имя файла filename не существует, файл создается.

Введите командную строку

```
$ cal   >   cal_out
```

Дополните файл cal_out данными, которые выводит команда date.

Перенаправление ввода позволяет программам получать данные из файла вместо стандартного ввода по умолчанию (клавиатуры терминала).

Для перенаправления stdin в командной строке используется метасимвол (<):

```
команда < имя_файла
команда 0< имя_файла
```

Если номер дескриптора опущен, по умолчанию принимается значение 0.

Перенаправление stdin используется гораздо реже, чем перенаправление stdout или stderr, потому что большинство команд открывают файлы (с дескрипторами, отличными от 0,1 или 2) самостоятельно. Однако если они только читают из stdin, перенаправление ввода может быть особенно полезным.

Приведенные ниже команды дают одинаковый результат, однако, они отличаются друг от друга:

```
$ cat text
$ cat < text
```

Первая команда cat открывает файл text и определяется новый дескриптор файла. Вторая команда cat использует stdin, который был связан оболочкой ссылкой к файлу text.

```
$ more myfile
$ more < myfile
```

В первом примере команда more открывает файл myfile и определяет новый дескриптор файла. Во втором примере more использует stdin, который был связан оболочкой с файлом myfile. Здесь можно увидеть различие в действиях, когда more открывает файл и когда использует stdin: строка состояния в нижней части экрана не отображает процент просмотренной части файла в том случае, когда команда more читает данные из stdin.

Возможность гибкого управления потоком ввода и вывода данных обеспечивается синтаксической конструкцией, используемой shell и командой tee операционной системы Unix

Синтаксическая конструкция shell, которая позволяет создавать возможность перенаправления потока со стандартного вывода одной команды на стандартный ввод другой команды, называется конвейером команд. В отличие от обычного перенаправления ввода/вывода, когда используется файл, конвейер реализует непосредственную передачу данных из выходного потока данных одного процесса во входной поток другого процесса.

Символ | (вертикальная черта) в командной строке указывает на перенаправление стандартного вывода команды1 на стандартный ввод команды2:

```
команда1  |  команда2
```

Команда1 должна записать выводные данные в stdout, а команда2 прочитать данные из своего stdin. Надо заметить, что в конвейере нельзя использовать команды (указываемые после символа вертикальная черта - |), которые не используют stdin, (например, date или ps) и команды, использующие интерактивный режим работы, такие, как vi или mail.

Каждая из команд в конвейере должна быть полностью определена, и содержать все опции и аргументы, которые требуются для ее выполнения.

Следующие команды позволяют вывести на терминал информацию о числе пользователей, работающих в настоящий момент:

```
$ who   >  tmpwho
$ wc   -l   tmpwho
$ rm   tmpwho
```

Та же задача может быть решена, путем использования командной строки следующего вида:

```
$ who  |   wc   -l
```

В общем случае все команды, которые могут читать из stdin и писать в stdout называются фильтрами.

Команды, которые не могут читать из stdin (например, date) или писать в stdout (например, exit), и команды, использующие интерактивный режим работы (например, vi или sh), не являются фильтрами.

Многие команды фильтры разработаны таким образом, что они могут читать по умолчанию из stdin (например, sort или cat), но только в случае, когда в командной строке не указано имени файла. Если имя файла указано, то команда открывает этот файл. Это позволяет использовать такие команды в конвейере.

Команда wc подсчитывает число строк, слов и символов в текстовом файле:

```
wc [ -l | -w | -c ] [ file … ]
```

Строки файла определяются числом символов перевода строки.

Слова являются любым набором символов, они отделяются друг от друга символом перевода строки, пробелом или знаком табуляции.

Следующая командная строка определяет число зарегистрированных пользователей системы:

```
$ cat /etc/passwd | wc -l
174
$
```

Команда sort сортирует строки текстового файла, а затем результат записывает в стандартный вывод. Строки вывода упорядочиваются в соответствии с содержащимися в

них символами (слева направо). Если указывается несколько файлов, то они сортируются и сливаются. Для более подробного знакомства с опциями команды обратитесь к соответствующей странице руководства.

sort *[options] [file …]*

Как и любой другой фильтр sort не изменяет содержимого входного файла, а результат записывает в stdout.

Следующая командная строка сортирует содержимое файл /etc/passwd, а на экран выводит пронумерованные строки файла.

```
$ sort /etc/passwd | nl
```

Команда tr выполняет трансляцию указанных символов файла, заменяя их новыми.

tr *[option] строка1 строка2*

Команда заменяет каждый символ набора строка1 соответствующими символами набора строка2. При этом число символов строка2 (обычно) должно быть равно числу символов строка1.

Тройниками являются команды фильтры, которые могут читать stdin, а писать одновременно в stdout и в один или несколько файлов, указанных в командной строке.

Примером такой команды может служить **команда tee**:

tee *[options] [file …]*

Команда tee не изменяет содержимого потока данных через stdin, а просто выводит две или более копий, одна из которых направляется в stdout, а все остальные в другие файлы; в том случае, если указанные файлы не существуют, команда создает их заново, содержимое уже существующих файлов уничтожается. Опция –a используется для добавления вывода в файлы без замены их содержимого.

6.6.9. Программы и процессы

Обычно **программой** называют исполняемый файл, хранящийся на устройстве внешней памяти. Программы могут быть пользовательскими или системными, т.е. находящимися в составе ОС (команды ОС Unix). Как и все другие файлы, программа имеет своего владельца и группу, к которой он принадлежит. Для всех категорий пользователей программы определены права доступа, в том числе, право исполнять программу. Программы при выполнении могут вызывать другие программы или подпрограммы, часть из которых может представлять системные вызовы – базовые обслуживающие программы. Готовая к исполнению программа (загрузочный модуль), помещенная в оперативную память, называется образом программы.

Когда программа запускается на исполнение, ОС Unix порождает особый объект, называемый **процессом**. Управление процессами в системе осуществляется специальной системной программой, входящей в состав ядра системы. Основные свойства и атрибуты процесса можно определить следующим образом:

Процесс – это выполняемая системой программа, загрузочный модуль которой помещен в оперативную память. При этом ядром выделены требуемые ресурсы системы (необходимая для выполнения процесса память, центральный процессор и т.д.)

Процесс размещается в области памяти, которая используется для загрузки команд программы и требуемых для работы данных.

Процесс всегда имеет три открытых файла (канала) ввода-вывода: stdin, stdout, и stderr, которые имеются в таблице дескрипторов файлов вместе с ограниченным числом других открытых файлов.

Поскольку в системе одновременно может существовать несколько процессов, время центрального процессора (или процессоров, если система многопроцессорная) распределяется между ними. Процесс, занимающий в текущий момент времени процессор, называется активным. Все другие процессы ожидают выделения процессорного времени в очереди.

Каждому процессу в системе присвоено значение приоритета, на основании которого ему планируется выделение времени процессора.

Владельцем процесса является пользователь, процесс которого его создал. Этот процесс обычно связан с управляющим терминалом, с которого зарегистрированный пользователь запустил программу на исполнение.

Процесс всегда создается другим процессом. Создающий процесс называется родителем, а созданный – потомком или дочерним процессом. Каждый потомок может иметь только одного родителя. Это создает иерархическую структуру, подобную структуре файловой системы.

Процесс идентифицируется уникальным идентификационным номером (PID), который используется системой.

Процесс является объектом управления операционной системы Unix. Он создается только по запросу другого процесса, процесса-родителя, и становится его потомком. Новый процесс выполняется определенное время и завершается, либо по команде пользователя, либо после завершения выполнения своей работы.

Используйте **команду whodo**, которая выводит список процессов выполняемых пользователями системы

```
whodo
```

Какие процессы порождены вашим процессом shell? Определите уникальные идентификаторы (PID) каждого процесса.

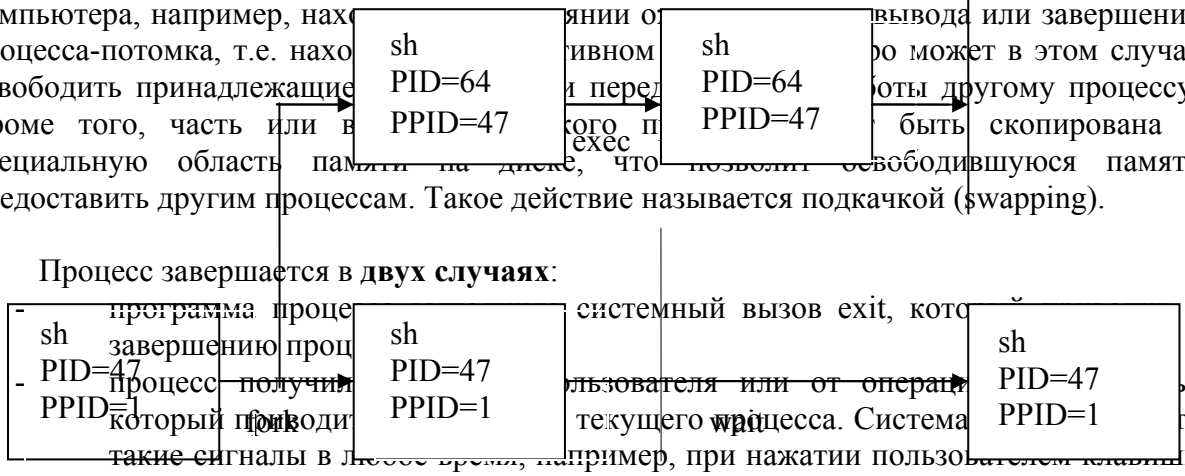
Для создания нового процесса имеется только один системный вызов функции ядра. Оболочка (shell) использует этот вызов, чтобы создать процесс, который выполнит введенную команду.

Схема создания и выполнения процесса выглядит следующим образом:

- родительский процесс выполняет системный вызов fork, который создает новый процесс, обычно идентичный родительскому процессу (другими совами, создается копия родительского процесса);
- потомку присваивается уникальный номер (PID);
- потомок замещает текущую программу новой программой (обращаясь к системному вызову exec). Новая программа выполняется с атрибутами процесса потомка (полученными по наследству от родительского процесса);

- потомок завершает свою работу с помощью системного вызова exit. По завершению процесса потомка все занимаемые ресурсы освобождаются и возвращаются системе;
- одновременно родительский процесс обычно выполняет системный вызов wait, который приостанавливает выполнение родительского процесса до тех пор, пока процесс-потомок не завершится. Следует обратить внимание на то, что терминал в этом случае является ресурсом потомка, поэтому до завершения работы потомка он не будет доступен родительскому процессу.

Процесс может существовать, но не выполняться центральным процессором компьютера, например, нахо[illegible]янии о[illegible]вывода или завершения процесса-потомка, т.е. нахо[illegible]тивном [illegible]ро может в этом случае освободить принадлежащие [illegible] и передать работы другому процессу. Кроме того, часть или в[illegible]кого п[illegible] быть скопирована в специальную область памяти на диске, что позволит освободившуюся память предоставить другим процессам. Такое действие называется подкачкой (swapping).



Процесс завершается в **двух случаях**:

- программа проце[illegible] системный вызов exit, кото[illegible]к завершению проц[illegible]
- процесс получи[illegible]ользователя или от операц[illegible]ы, который произво[illegible] текущего процесса. Система[illegible]ть такие сигналы в любое время, например, при нажатии пользователем клавиши <Del>, завершении работы пользователя или выключении питания компьютера.

Например, выполняется команда date. Ваш процесс shell порождает потомка, который выполняет программу date (программа находится в файле /bin/date). После того, как потомок завершит запись текущей даты и времени в стандартный канал вывода, он обратится к системному вызову exit. Родитель (текущий процесс shell), дождется завершения процесса потомка, после чего выведет подсказку на ввод новой команды.

Используя **команду ps** ОС Unix, можно получить **информацию** о состоянии всех процессов в текущий момент времени.

Формат команды:

```
ps  [опции [аргументы] . . .]
```

В зависимости от используемых опций выводится полная или сокращенная информация о состоянии процессов. Если в команде не указываются опции и аргументы, выводится таблица процессов пользователя.

Таблицу процессов, содержащую более подробную информацию, можно увидеть с помощью ряда опций команды ps:

`–e` позволяет получить информацию обо всех процессах системы;

`–f` дает возможность получить подробную информацию о процессах пользователя и содержит, например, столбцы:

UID – идентификатор пользователя, который запустил процесс;

PPID – идентификатор процесса-родителя;

C – фактор оценки загрузки процессора процессом в предыдущий интервал времени;

STIME– время старта процесса;
CMD – имя процесса;

–l позволяет получить список всех атрибутов процесса;

–u *регистрационное_имя* позволяет получить информацию о процессах пользователя, имя которого указано в команде.

Процесс оболочки для каждого зарегистрированного пользователя является процессом-родителем всех процессов, которые выполняет пользователь.

Этот процесс обычно ожидает окончания всех порожденных процессов. Однако имеется возможность указать процессу sh, чтобы он не ожидая окончания процесса-потомка, разрешил выполнение следующей команды. Это называется запуском процесса в **фоновом режиме** (background).

Выполнение команды в фоновом режиме указывается **символом &**, который помещается в конце командной строки:

команда &

При этом на экран выводится порядковый номер фонового процесса (job) и его числовой идентификатор (job_ID).

```
$ find . –name .profile –print 2>/dev/null &
[ 1 ]     756
```

Обычно в фоновом режиме запускаются процессы, требующие большого времени выполнения. Такой режим **не допускается для интерактивных программ**, таких как vi.

Когда процесс выполняется в фоновом режиме, необходимо предусмотреть перенаправление его стандартных каналов вывода в файл; в противном случае он будет выводить на экран сообщения в непредсказуемое время, мешая работе других команд.

Оболочка sh информирует об окончании фонового процесса, выводя сообщение, содержащее имя команды и номер фонового процесса:

```
[ 1 ] + Done          find &
```

Это сообщение не будет смешиваться с сообщениями текущих команд, т.к. оно выводится только в тот момент, когда в командной строке появляется подсказка оболочки (по умолчанию символ $).

Если необходимо, чтобы процесс, который требует очень большого времени, выполнялся даже тогда, когда пользователю необходимо выйти из системы, его необходимо запустить с помощью **команды nohup** в фоновом режиме:

nohup *команда [аргументы]* &

Эта команда перенаправляет весь вывод в файл nohup.out, если перенаправления вывода не было сделано раньше.

Оболочка ksh имеет возможность управлять задачами (заданиями), переводя их из фонового режима (background) в режим переднего плана (foreground) и наоборот. Это называется управлением заданиями (job control). Для этого процесс переводится в *приостановленное* состояние (нажатием клавиш <Ctrl> z). Когда порожденный процесс приостанавливается, на экран выводится сообщение

```
[ 1 ] + Stopped  команда
```

Приостановленный процесс можно перевести в **фоновый режим командой bg** или **режим переднего плана командой fg**:

```
bg [ job_ID … ]

fg [ job_ID … ]
```

Командой jobs можно вывести состояние всех выполняемых процессов-потомков.

Используя **команду sleep**, запустите несколько процессов:

```
sleep 60&
sleep 100&
sleep 30
```

Теперь нажмите (Ctrl>z). Обратите внимание на номер остановленного процесса и появившуюся подсказку оболочки.

Просмотрите состояние порожденных процессов командой job.

Используйте команду bg для перевода процесса sleep 30 в фоновый режим.

Используйте команду fg для вывода процесса sleep 60 из фонового режима.

Что вы ожидаете увидеть командой job, когда завершится?

Сигнал является способом уведомления о некотором происшедшем событии между процессами или между ядром системы и процессами. Сигналы можно рассматривать, как простейшую форму взаимодействия между процессами, хотя они больше напоминают программные прерывания, при которых нарушается нормальное выполнение процесса. Современные версии ОС Unix поддерживают до 32 различных сигналов. К генерации сигналов могут привести различные ситуации:

Сигнал 1 (сигнал hangup). Ядро системы посылает такой сигнал дочерним процессам текущего процесса shell, когда вы выходите из этого процесса shell.

Сигнал 2 (сигнал прерывания). Ядро отправляет процессу сигнал при нажатии пользователем определенных клавиш или комбинации клавиш. Например, нажатие клавиши <Del> (или <Ctrl> C) приводит к завершению текущего процесса.

Сигнал 3 (сигнал выхода). Нажатие комбинации клавиш <Ctrl>\ приводит к завершению текущего процесса и записи его образа на диск с целью дальнейшего анализа с помощью программы отладчика. Некоторые приложения могут игнорировать сигнал 2. В этом случае может помочь сигнал 3. Процессы, работающие в фоновом режиме, обычно не реагируют на сигналы 2 или 3. Для этих и других подобных процессов передача сигналов должна осуществляться с помощью команды kill.

Другие сигналы могут быть посланы процессам в результате различных обстоятельств. Например, **сигнал 15** приводит к завершению выполнения процесса, получившего сигнал

Иногда процесс продолжает существовать и после посылки ему сигнала 15. В этом случае применяется более жесткое средство – посылка процессу **сигнала 9**, поскольку этот сигнал нельзя ни перехватить, ни игнорировать.

Как уже отмечалось выше, некоторые сигналы генерируются в результате нажатия клавиш:

- нажатие клавиши <Del> приводит к генерации сигнала 2, который посылается текущему процессу;
- нажатие комбинации клавиш <Ctrl>\ приводит к генерации сигнала 3.

Другие сигналы, например, сигналы, посылаемые ядром системы в результате выполнения процессом недопустимой команды, обращения к несуществующему адресу или выполнения операции деления на нуль, не могут генерироваться с клавиатуры. Однако посылка любого сигнала процессу возможна с помощью **команды kill**:

kill [*–номер сигнала*] [pid …]

Команда kill может послать любой сигнал, номер которого указан в качестве опции команды. Если он не указан, по умолчанию посылается сигнал номер 15.

Сигнал посылается командой kill процессам, идентификаторы которых указаны в качестве аргументов. С помощью команды kill сигнал может быть послан **только** процессам, владельцем которых является пользователь, выполняющий команду kill.

Для удаление процесса, например, find выполните команду:

```
find / -name core –print /dev/null 2>&1 &
```

С помощью команды ps определите идентификатор процесса (PID) find, затем удалите его с помощью команды kill.

6.6.10. Интерпретатор командного языка

Интерпретатор shell - это программа, которая позволяет вам связываться с операционной системой. Shell считывает команды, которые вы вводите, и интерпретирует их как запросы на выполнение других программ, на доступ к файлу или обеспечение вывода. shell также является мощным языком программирования.

В этом разделе описаны команды и символы, имеющие специальное значение, которые позволяют:

- находить с помощью шаблона и манипулировать группами файлов;
- запускать команду в фоновом режиме или в определенное время;
- выполнять последовательно группу команд;
- перенаправлять стандартный ввод и вывод;
- завершать работающие программы.

Метасимволы, используемые оболочкой shell:

*** ? [ ]** эти метасимволы позволяют указывать сокращенные имена файлов при поиске по шаблону;

& означает, что команда будет выполняться в фоновом режиме;

; точка с запятой разделяет команды в командной строке;

\ отменяет специальное значение символов, таких как *****, **?**, **[**, **]**, **&**, **;**, <, >, |;

'...' одиночные кавычки отменяют значение пробела как разделителя и специальное значение всех символов;

"..." двойные кавычки отменяют значение пробела как разделителя и специальное значение всех символов, за исключением $ и \;

> перенаправляет вывод команды в файл;

<	перенаправляет ввод для команды из файла;
>>	перенаправляет вывод команды, который должен быть добавлен в конец существующего файла;
|	создает канал, направляющий вывод одной команды во ввод другой команды;
`...`	используется в паре; позволяет использовать вывод команды как аргументы в командной строке;
$	используется с позиционными параметрами и определенными пользователем переменными; также используется по умолчанию в качестве подсказки shell.

Метасимволы используются для поиска имен файлов, посредством их упрощается задача указания имен файлов или групп имен файлов как аргументов команды.

Метасимвол * осуществляет поиск **любой строки символов**, включая нулевую (пустую) строку. Вы можете использовать * для обозначения полного или частичного имени файла. Просто символ * ищет все имена файлов и каталогов в текущем каталоге, за исключением тех, которые начинаются с точки. Чтобы посмотреть метасимвол * в действии, введите его как аргумент в команде echo:

```
echo *
```

В ответ система выведет перечень всех имен файлов в вашем текущем каталоге.

Символ * может представлять символы в любой части имени файла. Например, если вы знаете, что несколько файлов имеют одинаковые первые и последние буквы, то вы можете выдать запрос, основываясь на этом факте. Если в вашем каталоге находятся файлы FATE, FE, FADED_LINE, FIG3.4E, FINE_LINE, FAST_LINE, то для отображения всех этих файлов на экране введите команду:

```
$ ls F*E
```

Вы можете, например, с помощью метасимвола * найти все файлы с именами memo в системном каталоге: ls */memo

Метасимвол ? заменяет любой **один символ** в имени файла за исключением лидирующей точки. Предположим, вы имеете книгу, в которой 12 глав и хотите получить список глав до 9-ой главы. Если ваш каталог содержит следующие файлы:

```
Chapter1
Chapter2
Chapter5
Chapter9
Chapter11
```

то для получения всех глав, которые начинаются со строки "Chapter" и заканчиваются одним символом введите команду ls с метасимволом ?:

```
$ ls Chapter?
Chapter1  Chapter2 Chapter5 Chapter9
$
```

Хотя метасимвол осуществляет поиск одного символа, вы можете использовать его для поиска более одного символа в имени файла. Например, вы получите перечень всех остальных глав в текущем каталоге, если введете следующую команду:

```
$ ls Chapter??
```

И, конечно, чтобы получить список всех глав в текущем каталоге, используйте метасимвол *:

```
$ ls Chapter*
```

Если вы хотите, чтобы shell нашел любой символ из перечисленных вами символов, то заключите эти символы в квадратные скобки. Предположим, ваш каталог содержит следующие файлы: cat, fat, mat, rat. Если вы воспользуетесь в качестве части имени файла шаблоном [crf], то shell будет искать имена файлов, в которые входят либо буква "c", либо буква "r", либо буква "f" в указанной позиции, например:

```
$ ls [crf]at
cat  fat rat
$
```

Символы, которые могут быть сгруппированы в скобки, называются классом символов.

Скобки могут также использоваться для обозначения диапазона символов, цифр или букв. Предположим, что в вашем каталоге содержатся следующие файлы: chapter1, chapter2, chapter3, chapter4, chapter5, chapter6. Если вы укажете:

```
chapter[1-5]
```

то shell найдет файлы с chapter1 по chapter5.

Класс символов можно также указать с помощью диапазона букв. Если вы укажете [A-Z], то shell будет искать только большие буквы, если [a-z] - то малые буквы.

shell обрабатывает также и другие символы, которые позволяют вызывать другие полезные функции.

Некоторые команды shell занимают много времени при выполнении. Эти команды можно запустить в фоновом режиме с использованием &, освобождая тем самым терминал для других задач. Общий формат для запуска команд в фоновом режиме следующий:

```
команда &
```

Примечание. Интерактивные команды shell (например, read, sh, vi) нельзя запускать в фоновом режиме.

Когда вы запускаете команду в фоновом режиме, то система UNIX выводит номер процесса. Вы можете использовать этот номер для завершения выполняющейся в фоновом режиме команды. Появившаяся подсказка означает, что терминал свободен и ожидает вашу команду.

Запустить команду в фоновом режиме вы можете только в том случае, если ваш терминал предоставляет вам такую возможность.

В одной командной строке вы можете указать несколько команд. Эти команды должны быть разделены символом ; (точка с запятой) или символом &:

команда1; команда2; команда3 …

Система UNIX выполняет команды в том порядке, в котором они стоят в командной строке, и выводит вывод этих команд в том же порядке. Этот процесс называется последовательным выполнением.

Например, введите:

```
$ cd; pwd; ls
```

shell выполнит эти команды последовательно:

cd изменит ваше местоположение, переместив вас в регистрационный каталог;
pwd выведет полное имя пути вашего текущего каталога;
ls перечислит файлы в вашем текущем каталоге.

Символ \ позволяет вам **отменить специальное значение** следующего за ним символа. Например, у вас есть файл trail, который содержит следующий текст:

```
The all * game
was held in Summit.
```

Чтобы найти символ звездочка (*) в файле, воспользуйтесь командой grep:

```
$ grep \* trail
The all * game
$
```

Команда grep найдет символ * в тексте и отобразит строку, в которой она появилась. Без символа \, символ звездочка будет интерпретироваться shell как метасимвол.

Отменить специальное значение символа вы также можете с помощью метасимвола кавычки:

- одиночные кавычки ('...') отменяют специальное значение всех символов за исключением самих одиночных кавычек;
- двойные кавычки ("...") отменяют специальное значение всех символов, за исключением символов двойные кавычки, $ и `(слабое ударение).

Использование кавычек удобно для цифровых специальных символов.

Например, ваш файл trail содержит строку:

```
He really wondered why? Why???
```

Чтобы найти строку, содержащую три вопросительных знака, воспользуйтесь командой grep:

```
$ grep '???' trail
He really wondered why? Why???
$
```

Кавычки аналогично обратной косой черте часто используются для отмены специального значения пробела. shell интерпретирует пробел в командной строке как разделитель между аргументами команды. Одиночные и двойные кавычки и обратная косая черта позволяют отменить это значение.

Например, чтобы в тексте разместить два или более слова, сделайте эти два слова одним аргументом, заключив их в кавычки. Чтобы найти два слова "The all" в файле trail, введите следующую команду:

```
$ grep 'The all' trail
The all * game
$
```

Особенно полезно применение отмены специального значения пробела для функции banner, которая использует пробел как разделитель аргументов и выводит аргументы на отдельных строках.

Чтобы напечатать более одного аргумента на одной строке, заключите слова в двойные кавычки. Например, если вы введете:

```
$ banner happy birthday to you
```

то команда banner выведет ваше сообщение на 4-х строках. Если вы введете:

```
$ banner happy birthday "to you"
```

то команда banner выведет ваше сообщение на 3-х строках, причем слова "to" и "you" будут на одной строке.

Примечание. Команда banner выводит сообщения на экране терминала большими плакатного размера буквами.

В системе UNIX некоторые команды ожидают ввод только с клавиатуры (стандартный ввод) и большинство команд отображают свой вывод на экране терминала (стандартный вывод). Однако система UNIX позволяет вам перенаправлять ввод и вывод в файлы и программы, т.е. вы можете сказать shell:

- взять ввод из файла, а не с клавиатуры;
- послать вывод в файл, а не на терминал;
- использовать программу как исходные данные для другой программы.

Чтобы перенаправить ввод, укажите в командной строке после знака "меньше " (<) имя файла:

команда < имя_файла

Чтобы перенаправить вывод, укажите в командной строке после знака "больше " (>) имя файла:

команда > *имя_файла*

Примечание. Если вы перенаправите вывод в уже существующий файл, то вывод вашей команды заменит содержимое существующего файла.

Перед тем, как перенаправить вывод команды в конкретный файл убедитесь, что этот файл не существует. shell не предупреждает, что выполняет перезапись существующего файла.

Чтобы убедиться, что файл с запланированным именем не существует, воспользуйтесь командой ls с аргументом "имя_файла". Если этот файл не существует, то ls выдаст сообщение, что файл не найден в текущем каталоге. Например, проверка существования файлов temp и junk даст следующий результат:

```
$ ls temp
temp
$ ls junk
junk: no such file or directiry
$
```

Это означает, что вы можете назвать свой файл junk, но не можете использовать в качестве имени temp, если не хотите потерять содержимое существующего файла.

Чтобы добавить вывод в существующий файл и не разрушить его, вы можете воспользоваться символом >>:

```
команда >> имя_файла
```

В результате выполнения команды вывод будет добавлен в конец существующего файла. Если файл не существует, то он будет создан. Рассмотрим, например, как добавить вывод с помощью команды cat. Команда cat выводит содержимое файлов, имена которых являются ее аргументами, в стандартный вывод. Если нет аргументов, то она выводит стандартный ввод в стандартный вывод. Сначала выполните команду cat без перенаправления вывода. Затем содержимое файла trial2 добавляем после последней строки в файл trial1 при выполнении команды cat над файлом rtial2, перенаправив вывод в файл trial1:

```
$ cat trial1
This is the first line of trial1.
Hello.
This is the last line of trial1.
$
$ cat trial2
This is the beginning of trial2.
Hello.
This is the end of trial2.
$ cat trial2 >> trial1
$ cat trial1
This is the first line of trial1.
Hello.
This is the last line of trial1.
This is the beginning of trial2.
Hello.
This is the end of trial2.
$
```

Перенаправление вывода очень удобно в том случае, если вы не хотите, чтобы вывод появлялся на экране немедленно, или хотите сохранить его.

Команда sort размещает строки указанного файла в алфавитном или цифровом порядке. Прежде чем перенаправить вывод команды в файл убедитесь, что имя этого файла не существует. Команда sort сначала очищает файл, который будет содержать вывод, затем выполняет сортировку и помещает вывод в пустой файл.

Когда команда запущена в фоновом режиме, то вывод ее осуществляется на экране терминала. И если вы используете терминал в то же время для выполнения других задач, то вывод фоновой задачи будет прерывать вашу работу. Однако если перенаправить вывод в файл, то вы сможете спокойно работать.

Предположим, что вы хотите найти все появления слова "test" в файле schedule. Запустите команду grep в фоновом режиме и перенаправьте вывод в файл testfile:

```
$ grep test schedule > testfile &
```

Теперь вы можете использовать терминал для других работ и просмотреть файл testfile позднее.

Символ | называется каналом. Канал является мощным средством, которое позволяет вам брать вывод одной команды и использовать его в качестве ввода для другой команды без создания временных файлов. Таким образом, построенная последовательность команд называется конвейером. Общий формат конвейера:

```
команда1 | команда2 | команда3 ...
```

Вывод команды1 используется как ввод для команды2. Вывод команды2 используется как ввод для команды3.

Чтобы понять, насколько эффективен конвейер, рассмотрим 2 способа, которые дают одинаковый результат:

Использование метода перенаправления ввода/вывода. Запустим одну команду и перенаправим ее вывод во временный файл. Затем запустим вторую команду, которая берет содержимое временного файла как ввод. И в конце удалим временный файл.

Использование метода конвейера. Например, предположим, что вы хотите послать сообщение happy birthday с помощью команды banner владельцу user2.

Выполним сначала по первому методу:

1) введите команду banner и перенаправьте ее вывод во временный файл:

```
$ banner happy birhday > message.tmp
```

2) введите команду mail и в качестве ввода воспользуйтесь файлом message.tmp:

```
$ mail user2 < message.tmp
```

3) удалите временный файл:

```
$ rm message.tmp
```

Вторым методом это можно сделать быстрее:

```
$ banner happy birthday | mail user2
```

Вывод большинства команд может использоваться как **аргумент в командной строке**. Для этого команду заключите между знаками "слабое ударение" (`...`) и поместите ее в командной строке в том месте, где вывод будет трактоваться как аргумент.
Например, вы можете подставить вывод конвейера команд date и cut в качестве аргумента в команде banner:

```
$ banner `date | cut -c12-19`
```

Обратите внимание на результат: система выводит banner с текущим временем.

6.9.11. Выполнение, остановка и повторный запуск процессов

В этом подразделе описывается:

- как запустить команду в определенное время с помощью команд batch и at;
- как получить информацию о состоянии процесса;
- как завершить активный процесс;
- как вновь запустить остановленный процесс;
- как превести процесс из оперативного режима в фоновый режим и наоборот.

Команды batch и at позволяют вам определять время запуска команды или последовательности команд. При помощи команды batch система определяет время запуска команды, вы это можете определить с помощью команды at. Обе команды ожидают ввод со стандартного ввода (терминала); список команд, вводимых с терминала, должен завершаться нажатием клавиши ^d (одновременное нажатие клавиши Ctrl и клавиши d).

Команда batch очень полезна, если вы запускаете процесс или программу, которые занимают много системного времени. Команда batch представляет системе задание (содержащее последовательность команд для выполнения). Задание ставится в очередь и запускается, как только у системы появляется возможность. Это позволяет системе быстро отвечать на запросы других пользователей. Общий формат команды batch:

```
batch
первая_команда
.
.
.
последняя_команда
<^d>
```

Если запускается только одна команда, то ее можно ввести в одной командной строке:

batch *команда*

В следующем примере batch используется для выполнения команды grep в согласованное время. Команда grep осуществляет поиск всех файлов в текущем каталоге и перенаправляет вывод в файл dol.file.

```
$ batch
grep dollar * > dol.file
<^d>
job 155223141.b at Sun Dec 11:14:54 2000
$
```

После того как вы дадите задание batch, система выдаст ответ, в котором даны номер задания, дата и время. Номер задания не то же самое, что номер процесса, который система генерирует, когда вы запускаете команду в фоновом режиме.

Команда at позволяет вам указывать точное время выполнения команд. Общий фомат команды at следующий:

```
   at time
        первая_команда
. . .
        последняя_команда
        <^d>
```

Аргумент time состоит из времени дня и даты, если дата не сегодняшняя.

В следующем примере показано, как использовать команду at для посылки сообщения happy birthday пользователю с регистрационным именем chief:

```
$ at 8:15am Feb 27
banner happy birthday | mail chief
<^d>
$
```

Обратите внимание, что команда at подобно команде batch выдает ответ с номером задания, датой и временем.

Если вы не хотите, чтобы команды, находящиеся в данный момент в очереди заданий batch или at были выполнены, то можете удалить их из очереди. Для этого воспользуйтесь опцией -r в команде at, указав ее с номером задания. Общий формат такой команды:

```
at -r номер_задания
```

Например, чтобы удалить предыдущее задание at, введите:

```
$ at -r 453400603.a
```

Если вы забыли номер задания, то команда:

```
$ at -l
```

выведет список текущих заданий в очереди batch или at, как показано на следующем экране:

```
$ at -l
user mylogin 168302040.a at Sat Nov 25 13:00:00 2000
user mylogin 453400603.a at Fri Feb 24 08:15:00 2000
$
```

Таким образом, команда at выполняет команды в указанное время. Вы можете использовать от одной до 4-х цифр и буквосочетания "am" и "pm", чтобы указать время. Чтобы указать дату, задайте имя месяца и вслед за ним число. Если задание должно быть выполнено сегодня, то дату вводить не надо.

Пример

```
at 08:15am Feb 27
at 5:14pm Sept 24
```

Команда ps дает вам состояние всех процессов, запущенных на данный момент. Например, вы можете использовать команду ps, чтобы просмотреть состояние всех процессов, которые запущены в фоновом режиме, применив символ &.

В следующем подпункте обсуждается вопрос, как применить номер PID (идентификатор процесса), чтобы остановить выполнение команды. PID является уникальным номером, который система UNIX назначает каждому активному процессу.

В следующем примере команда grep запускается в фоновом режиме и затем выдается команда ps. Система сообщает в ответ номер идентификации процесса (PID) и номер терминала (TTY). Она также выдает время выполнения каждого процесса (TIME) и имя команды, которая выполняется (COMMAND):

```
$ grep word * > temp &
28223
$
$ ps
PID         TTY       TIME       COMD
28124       tty10     0:00       sh
28223       tty10     0:04       grep
28224       tty10     0:00       ps
$
```

Обратите внимание, на экране отображен номер PID для команды grep так же, как и для всех других запущенных процессов: для самой команды ps и команды sh, которая была запущена во время вашей регистрации.

Вы можете приостановить и вновь запустить программу, если в вашей системе предусмотрена функция управления заданиями. Команда jobs выдает список текущих фоновых процессов, запущенных или приостановленных. Команда jobs дополнительно к PID распечатывает идентификатор задания (JID) и имя задания. Чтобы вновь запустить приостановленное задание, либо возобновить фоновый процесс в оперативном режиме, вам необходимо знать JID. JID распечатывается на экране каждый раз, когда вы вводите команду запуска или останова процесса. Если вы введете:

```
jobs
```

то на экране появится следующая информация:

```
[JID] - Stopped (signal) <имя задания>
```

или

```
[JID] + Running      <имя задания>
```

Команда kill завершает активные процессы в фоновом режиме и команда stop приостанавливает временно процессы. Общий формат этих команд:

```
kill [номер_сигнала]PID
```

или

```
stop JID
```

Обратите внимание, что вы не можете завершать фоновые задания нажатием клавиш BREAK или DEL. Следующий пример показывает, как вы можете завершить команду grep, которая выполняется в фоновом режиме.

```
$ kill -9 28223
28223 Terminated
$
```

После того как система выдаст ответ на запрос, на экране появится подсказка $, означающая, что процесс завершен. Если система не найдет указанного PID, то появится сообщение об ошибке:

```
kill 28223:No such process
```

Чтобы приостановить оперативный процесс (если активна функция управления заданиями), введите:

```
<ctrl>Z
```

На экране появится следующее сообщение:

```
<JID > Stopped(user)  <имя_задания>
```

Если функция управления заданиями активна, то вы можете **вновь запустить приостановленный процесс**. Чтобы вновь запустить процесс, остановленный командой stop, вы сначала должны определить JID с помощью команды jobs. Затем вы можете использовать JID со следующими командами:

```
fg <JID>
```

– возобновляет приостановленное задание или переводит задание из фонового режима в оперативный;

```
bg <JID>
```

– вновь запускает приостановленное задание в фоновом режиме.

Все процессы, за исключением at и batch, завершаются, когда вы выходите из системы. Если вы хотите, чтобы после вашего выхода из системы **процесс в фоновом режиме продолжал выполняться**, то вам необходимо использовать **команду nohup**. Команда nohup имеет следующий формат:

```
nohup команда &
```

Предположим, вы хотите, чтобы команда grep осуществила поиск во всех файлах в вашем текущем каталоге строки "word" и перенаправила вывод в файл word.list, и затем, не ожидая завершения, вы хотите выйти из системы, то введите следующую строку:

```
$ nohup grep word * > word.list &
```

Вы можете завершить команду nohup с помощью команды kill.

6.7 Операционная система Linux

Linux - это UNIX-подобная операционная система для персональных компьютеров и рабочих станций, соответствующая стандарту POSIX.

Изначально ОС Linux создавалась как UNIX-подобная система для ПК типа IBM PC с процессором i80386. в настоящее время имеется реализация этой ОС практически для всех видов процессоров и компьютеров на их основе. На базе ОС Linux создаются и встроенные системы и суперкомпьютеры. Система поддерживает кластеризацию и большинство современных интерфейсов и технологий.

Linux поддерживает большинство свойств, присущих другим реализациям UNIX, плюс некоторые другие.

Linux – это полноценная многозадачная многопользовательская операционная система. Linux хорошо совместим с рядом стандартов для UNIX на уровне исходных текстов.

Все исходные тексты для Linux, включая ядро, драйверы устройств, библиотеки, пользовательские программы и инструментальные средства, распространяются свободно.

Linux поддерживает различные типы файловых систем для хранения данных.

Linux, как и все ОС UNIX, поддерживает полный набор протоколов TCP/IP для сетевой работы.

Ядро ОС Linux сразу было создано с учетом возможностей защищенного режима процессоров i80386 и i80486. Ядро Linux поддерживает загрузку только нужных страниц памяти, т.е. с диска в память загружаются те сегменты программы, которые действительно используются.

Ядро ОС Linux поддерживает универсальный пул памяти для пользовательских программ и дискового кэш. При этом для кэширования может использоваться вся свободная память, и наоборот, кэш уменьшается при работе больших программ. Этот механизм позволяет увеличить производительность системы.

Выполняемые программы используют динамически связываемые библиотеки, т.е. выполняемые программы могут совместно использовать библиотечную программу, представленную одним физическим файлом на диске. Это позволяет выполняемым программам занимать меньше места на диске, особенно при многократном использовании библиотечных функций.

В ОС Linux разделяемые библиотеки динамически связываются во время выполнения, позволяя программисту заменять библиотечные функции своими собственными.

6.8. Вопросы к главе 6

1) Изложите основные архитектурные особенности ОС UNIX.
2) Перечислите и поясните основные понятия системы UNIX.
3) Изложите основные моменты, связанные с защитой файлов в ОС UNIX.
4) Что представляет собой вызов удаленной процедуры?
5) Какая информация необходима пользователю для регистрации в системе?
6) Где хранится учетная информация о пользователях системы? Что такое UID и GID?
7) Опишите четыре различных типов файлов в ОС UNIX.
8) Опишите команды, реализующие доступ к содержимому каталогов, текстовых файлов, файлов данных.
9) Права доступа к файлам и каталогам и их изменение.
10) Права доступа по умолчанию.
11) Опишите режимы работы и основные команды редактора vi.
12) Как выполняемая команда использует дескрипторы stdin, stdout и stderr?
13) Как осуществляется перенаправление ввода, вывода и ошибок?
14) Приведите примеры конвейера, фильтра и тройника.
15) В чем различие между программой и процессом?
16) Как перевести выполняющийся процесс в фоновый режим?
17) Как пользователь может завершить выполнение стартового процесса sh?
18) Что такое метасимволы? Приведите примеры.
19) Опишите назначение и приведите примеры использования символов (&, $, ;).
20) Опишите назначение и приведите примеры использования символов (“…”,’…’,`…`).
21) Опишите назначение и приведите примеры использования символов (>, <, |, >>).
22) Приведите пример подстановки результата выполнения команды в качестве аргумента выполняемой команды.

7. Рекомендуемая литература

Основная:

1. А.В. Гордеев, А.Ю. Молчанов, Системное программное обеспечение, учебник, Питер, 2001.
2. Архитектура Windows для разработчиков. Учебный курс, «Русская редакция», 1998
3. Коцюбинский А.О., Грошев С.В., Хрестоматия работы на компьютере, «Триумф», 2001
4. Стен Келли-Бутл, Введение в UNIX, «Лори», 1995
5. Unix. Настольный справочник. П.Дайсон. Изд. "Лори", 1997

Дополнительная:

1. Операционная система UNIX. А.Робачевский. Изд. "BHV-СПб", 1997
2. Любой справочник по операционной системе Unix.
3. Гукин, Ден, Гукин, Сандра Хардин, Освой самостоятельно Windows 2000 Professional за 24 часа, «Вильямс», 2000
4. Перри, Грег. Освой самостоятельно Microsoft Office 2000 за 24 часа, «Вильямс», 2000
5. А.Кинг, WINDOWS 95 изнутри, Microsoft Corporation, «Петер», 1995
6. Любая литература по операционным системам Windows