**Вопросы к экзамену**

*Возможны незначительные изменения в течение сентября 2023 года.*

1. Операционные системы: определение, назначение, состав, функции.

Операционная система (ОС) — комплекс системных и управляющих программ, предназначенных для наиболее эффективного использования всех ресурсов вычислительной системы (ВС) и удобства работы с ней.

Назначение:

ОС управляет всеми устройствами компьютерной системы и обеспечивает пользователя удобным интерфейсом для работы с аппаратурой. Две основные функции (назначение) ОС:

1) предоставлять пользователю некую расширенную виртуальную машину, с которой легче работать (легче программировать), чем непосредственно с аппаратурой реального компьютера или реальной сети;

2) управлять ресурсами вычислительной системы.

Cостав:

Функции ОС автономного компьютера обычно группируются в соответствии с типами локальных ресурсов, которыми управляет ОС. Такие группы называют подсистемами. Наиболее важные из них:

• подсистема управления процессами,

• подсистема управления памятью,

• подсистема управления файлами,

• подсистема управления внешними устройствами,

• подсистема пользовательского интерфейса,

• подсистема защиты данных и администрирования.

Функции ОС:

• прием от пользователя (оператора) заданий или команд, сформулированных на соответствующих языках, и их обработка;

• загрузка в ОП программ и их исполнение;

• инициация программы (передача ей управления);

• прием и исполнение программных запросов на запуск, приостановку, остановку других программ; организация взаимодействия между задачами;

• идентификация всех программ и данных;

• обеспечение работы системы управления файлами и/или систем управления БД; • обеспечение режима мультипрограммирования (многозадачности);

• планирование и диспетчеризация задач;

• обеспечение функций по организации и управлению операциями ввода/вывода;

• удовлетворение жестким ограничениям на время ответа в режиме реального времени (для соответствующих ОС);

• управление памятью, организация виртуальной памяти;

• организация механизмов обмена сообщениями и данными между выполняющимися программами;

• защита одной программы от влияния другой; обеспечение сохранности данных; • аутентификация, авторизация и другие средства обеспечения безопасности;

• предоставление услуг на случай частичного сбоя системы;

• обеспечение работы систем программирования;

• параллельное исполнение нескольких задач.

1. Операционные системы: классификация, основные этапы развития, особенности современного этапа развития

Существует несколько классификаций ОС.

В зависимости от способа организации вычислений:

• Системы пакетной обработки – основной задачей является организация наибольшего количества вычислительных процессов за единицу времени. Определенные процессы объединяются в пакет, который затем обрабатывает ОС. • Системы разделения времени – создание возможности единовременного взаимодействия с устройством сразу несколькими людьми. В порядке очереди каждый пользователь получает определенный промежуток процессорного времени.

• Системы реального времени – организация работы каждой задачи за определенный промежуток времени, присущий каждой конкретной задаче.

В зависимости от типа ядра:

• OС с монолитным ядром;

• OС с микроядром;

• OС с гибридным ядром.

В зависимости от количества единовременно решаемых задач:

• однозадачные;

• многозадачные;

В зависимости от количества пользователей:

• однопользовательские;

• многопользовательские.

В зависимости от количества поддерживаемых процессоров:

• однопроцессорные

• многопроцессорные

В зависимости от возможности работы в компьютерной сети:

• локальные – автономные ОС, которые не позволяют работать с компьютерными сетями;

• сетевые – ОС с поддержкой компьютерных сетей.

В зависимости от роли в сетевом взаимодействии:

• серверные – ОС, открывающие доступ к ресурсам сети и осуществляющие управление сетевой инфраструктурой;

• клиентские – ОС, которые имеют возможность получения доступа к ресурсам сети.

В зависимости от типа лицензии:

• открытые – ОС с открытым исходным кодом, который можно изучать и редактировать;

• проприетарные – ОС, связанные с определенным правообладателем и, как правило, имеющие закрытый исходный код.

В зависимости от сферы использования:

• ОС мэйнфреймов – больших компьютеров;

• ОС серверов;

• ОС персональных компьютеров;

• OC мобильных устройств;

• встроенные OC;

• OC маршрутизаторов.

Основные этапы развития:

* Первое поколение(преимущественно электронные лампы,программы писались в машинном коде и загружались в память компьютера вручную, вычислит. система выполняла только одну операцию одновременно)
* Второе поколение(использование транзисторов, запись заданий на магнитную ленту, появляются первые настоящие компиляторы, Большие компьютеры второго поколения использовались главным образом для научных и технических вычислений, таких как решение дифференциальных уравнений в частных производных)
* Третье поколение(использовали малые интегральные схемы, разбиение памяти на разделы для многозадачности, способность считывать задание с перфокарт на диск по мере того, как их приносили в машинный зал(подкачка данных или спуллинг), разработка режима разделения времени — вариант многозадачности, при котором у каждого пользователя есть свой диалоговый терминал)
* Четвертое поколение(Операционные системы для микрокомпьютеров(весь процессор помещался в одной микросхеме), Первоначально системы четвертого поколения работали в текстовом режиме, позже появился графический интерфейс)
* Пятое поколение(для использования многоядерного процесса нужна многоядерная операционная система, являющаяся расширением одноядерной операционной системы, виртуализации — представление вычислительных ресурсов, абстрагированное от аппаратной реализации.)

На современном этапе развития операционных систем на передний план вышли средства обеспечения безопасности. Многие операционные системы обладают сегодня развитыми средствами защиты информации, основанными на шифрации данных, аутентификации и авторизации. Современным операционным системам присуща многоплатформенность, т.е. способность работать на совершенно различных типах компьютеров. В последние годы получила дальнейшее развитие долговременная тенденция повышения удобства работы человека с компьютером.

Постоянно повышается удобство интерактивной работы с компьютером путем включения в операционную систему развитых графических интерфейсов, использующих наряду с графикой звук и видеоизображение.

1. Компоненты архитектуры вычислительных систем, их назначение и взаимодействие.

Классическая архитектура ЭВМ, построенная по принципу фон Неймана (фон-неймановская архитектура) и реализованная в вычислительных машинах двух (трех) поколений, содержит следующие основные блоки:

• арифметическо-логическое устройство (АЛУ), выполняющее арифметические и логические операции;

• управляющее устройство (УУ), организующее процесс выполнения программ;

• внешнее запоминающее устройство (ВЗУ), или память, для хранения программ и данных;

• оперативное запоминающее устройство (ОЗУ);

• устройства ввода и вывода информации (УВВ).

Для связи между отдельными функциональными узлами ЭВМ используется общая шина (или магистраль)

Контроллер можно рассматривать как специализированный процессор, управляющий работой "вверенного ему" внешнего устройства по специальным встроенным программам обмена. Такой процессор имеет собственную систему команд.

Любая шина состоит из трех частей:

• шина данных, по которой передается информация;

• шина адреса, определяющая, кому передаются данные;

• шина управления, регулирующая процесс обмена информацией.

Также может присутствовать шина питания.

Центральный процессор — это «мозг» компьютера. Он выбирает команды из памяти и выполняет их. Обычный цикл работы центрального процессора выглядит так: выборка из памяти первой команды, ее декодирование для определения ее типа и операндов, выполнение этой команды, а затем выборка, декодирование и выполнение последующих команд. Этот цикл повторяется до тех пор, пока не закончится программа. у всех центральных процессоров есть несколько собственных регистров для хранения основных переменных и промежуточных результатов.

Большинство центральных процессоров имеют два режима работы: режим ядра и пользовательский режим (режим пользователя). Обычно режимом управляет специальный бит в слове состояния программы — PSW. При работе в режиме ядра процессор может выполнять любые команды из своего набора и использовать любые возможности аппаратуры. Пользовательские программы всегда работают в режиме пользователя, который допускает выполнение только подмножества команд и дает доступ к определенному подмножеству возможностей аппаратуры.

Есть несколько архитектур процессоров с различными наборами команд. Аббревиатура CISC обозначает Complex Instruction Set Computer(использовать сложные инструкции процессора, которые делают много действий; после ее решили с помощью микрокода. Для каждой инструкции из набора создается подпрограмма, которая состоит из простых инструкций и хранится в специальной памяти внутри микропроцессора), а RISC — Reduced Instruction Set Computer (Идея RISC заключается в замене сложных инструкций на комбинацию простых. Так не придется заниматься сложной отладкой микрокода. Вместо этого разработчики компилятора будут решать возникающие проблемы. Еще одна основная идея RISC — это конвейеризация. Инструкции разделены на этапы, каждый из которых выполняется примерно одинаковое количество времени)

Система памяти создается в виде иерархии уровней (регистры, кэш, основная память, магнитный диск, магнитная лента). Верхние уровни обладают более высоким быстродействием, меньшим объемом и более высокой удельной стоимостью хранения одного бита информации, чем нижние уровни, иногда в миллиарды и более раз.

Верхний уровень состоит из внутренних регистров процессора. Они выполнены по той же технологии, что и сам процессор, и поэтому не уступают ему в быстродействии. Затем следует кэш-память, которая управляется главным образом аппаратурой. Оперативная память разделяется на кэш-строки, обычно по 64 байт, с адресами от 0 до 63 в кэш-строке 0, адресами от 64 до 127 в кэш-строке 1 и т. д. Наиболее интенсивно используемые кэш-строки оперативной памяти сохраняются в высокоскоростной кэш-памяти, находящейся внутри процессора или очень близко к нему.

Следующей в иерархии идет оперативная память. Это главная рабочая область системы памяти машины. Оперативную память часто называют оперативным запоминающим устройством (ОЗУ), или памятью с произвольным доступом (Random Access Memory (RAM)). Все запросы процессора, которые не могут быть удовлетворены кэш-памятью, направляются к оперативной памяти. Дополнительно к оперативной памяти многие компьютеры оснащены небольшой по объему неизменяемой памятью с произвольным доступом — постоянным запоминающим устройством (ПЗУ), оно же память, предназначенная только для чтения (Read Only Memory (ROM)). В отличие от ОЗУ она не утрачивает своего содержимого при отключении питания, то есть является энергонезависимой.

Флеш-память также обычно используется как носитель информации в портативных электронных устройствах. По быстродействию флеш-память занимает промежуточное положение между ОЗУ и диском. Также, в отличие от дисковой памяти, если флеш-память стирается или перезаписывается слишком часто, она приходит в негодность.

Еще одна разновидность памяти — CMOS-память, которая является энергозависимой. Во многих компьютерах CMOS-память используется для хранения текущих даты и времени.

Диск:

Магнитный жесткий диск состоит из одной или нескольких металлических пластин, вращающихся со скоростью 5400, 7200, 10 000 или 15 000 оборотов в минуту. Информация записывается на диск в виде последовательности концентрических окружностей. В каждой заданной позиции привода каждая из головок может считывать кольцеобразный участок, называемый дорожкой. Из совокупности всех дорожек в заданной позиции привода составляется цилиндр. Каждая дорожка поделена на определенное количество секторов, обычно по 512 байт на сектор.

С целью дальнейшей увеличения плотности записи дорожки стали частично накладывать друг на друга. Такие диски эффективны при последовательной записи данных и редком их чтении. При черепичной магнитной записи (SMR — Shingled Magnetic Recording) на жесткий диск дорожки размещаются друг над другом, подобно черепице на крыше. Это позволяет повысить плотность записи.

Для простоты изложения, к жестким дискам также отнесем твердотельные накопители (SSD — Solid State Disks), которые хранят данных во флэш-памяти и не имеют движущихся частей. У них то же назначение, что и у магнитных дисков, и с точки зрения операционной системы выглядят так же. SSD-диски обладают бóльшим быстродействием, чем магнитные, но и стоимость таких дисков выше. Гибридные диски используют обе технологии. SSD-диски могут в одной ячейке хранить несколько бит данных. Чем больше бит данных в одной ячейке, тем выше емкость, ниже цена и меньше количество циклов перезаписи.

Шины PCI и PCIe

При использовании PCI центральный процессор взаимодействует с контроллером памяти по выделенному высокоскоростному соединению. Таким образом, контроллер соединяется с памятью непосредственно, то есть передача данных между центральным процессором и памятью происходит не через шину PCI. Другие периферийные устройства подсоединяются прямо к шине PCI. Машина такого типа обычно содержит 2 или 3 пустых разъема PCI, чтобы покупатели имели возможность подключать карты PCI для новых периферийных устройств)

Собственно, PCIe это вообще не шина, а одноранговая сеть, использующая разрядно-последовательные линии и коммутацию пакетов.

в шине PCIe соединения между устройствами являются последовательными, то есть имеют разрядность в один бит вместо 8, 16, 32 или 64 бит. Это позволяет значительно увеличить пропускную способность шины

Устройство может иметь до 32 проводных пар, называемых трактами (lanes) или дорожками. На большинстве материнских плат имеется 16-трактовый разъем для графической карты, что для PCIe 3.0 обеспечивает пропускную способность в 16 Гбайт/с — примерно в 30 раз больше, чем у графических карт PCI. Все взаимодействия являются одноранговыми. Когда процессор хочет обратиться к устройству, он отправляет этому устройству пакет и обычно получает ответ.

Шина USB

Общая пропускная способность первой версии шины (USB 1.0) составляла 1,5 Мбит/с. Версия 1.1 работает на скорости 12 Мбит/с, что было вполне достаточно для принтеров, цифровых камер и многих других устройств. Версия 2.0 поддерживает устройства со скоростью до 480 Мбит/c, что позволило подключать внешние диски, веб-камеры высокого разрешения и сетевые интерфейсы. В версии USB 3.0 скорость возросла до 5 Гбит/с. USB 3.2 поддерживает скорость до 20 Гбит/с. Последняя версия стандарта, USB4, поддерживает скорость до 40 Гбит/с. Шина USB состоит из корневого хаба, который подключается к главной шине. Этот корневой хаб содержит разъемы для кабелей, которые могут подсоединяться к устройствам ввода-вывода или к дополнительным хабам, чтобы увеличить количество разъемов. Топология шины USB представляет собой дерево с корнем в корневом хабе, который находится внутри компьютера. Коннекторы кабеля со стороны устройства отличаются от коннекторов со стороны хаба, чтобы пользователь случайно не подсоединил кабель другой стороной.

Видеокарта

Видеокарта – устройство, преобразующее цифровую информацию в форму, пригодную для дальнейшего вывода на экран монитора. Обычно видеокарта выполнена в виде печатной платы (плата расширения) и вставляется в слот расширения материнской платы, универсальный либо специализированный (AGP, PCI Express), но может быть реализована и на системной плате. Видеокарты не ограничиваются простым выводом изображения, они имеют встроенный графический процессор, который может производить дополнительную обработку, снимая эту задачу с центрального процессора компьютера. Также имеет место тенденция использовать вычислительные возможности графического процессора для решения неграфических задач (например, добычи криптовалюты).

Разъём HDMI меньше по размеру, чем DVI, а также поддерживает передачу многоканальных цифровых аудиосигналов. При помощи пассивного переходника порт DVI совместим с HDMI, но только в пределах режимов, которым достаточно single-link DVI.

В компьютере видеокарта подключается с одной из шин компьютера. В современных компьютерах это обычно PCI Express, хотя в старых компьютерах использовались и другие шины.

1. Система прерываний..

Существует два источника поступления прерываний (interrupt) и три типа особых ситуаций (исключений, exception). Кроме того, различают внутренние (программные, software) и внешние (аппаратные, hardware) источники генерации прерываний и особых ситуаций. В зависимости от источника, прерывания делятся на: аппаратные - возникают как реакция микропроцессора на физический сигнал от некоторого устройства (клавиатура, системные часы, клавиатура, жесткий диск и т.д.), по времени возникновения эти прерывания асинхронны, т.е. происходят в случайные моменты времени; программные - вызываются искусственно с помощью соответствующей команды из программы (int), предназначены для выполнения некоторых действий операционной системы, являются синхронными; исключения - являются реакцией микропроцессора на нестандартную ситуацию, возникшую внутри микропроцессора во время выполнения некоторой команды программы (деление на ноль, прерывание по флагу TF (трассировка)).

Общая классификация прерываний: внешние - вызываются внешними по отношению к микропроцессору событиями (это группа аппаратных прерываний); внутренние - возникают внутри микропроцессора во время вычислительного процесса (по существу это исключительные ситуации и программные прерывания). Внешние прерывания возникают по сигналу какого-нибудь внешнего устройства. Внешние прерывания подразделяются на немаскируемые и маскируемые.

Обработка прерываний в процессорах x86

Обработка прерывания в реальном режиме производится в три этапа:

1. Прекращение выполнения текущей программы.

Для этого необходимо сохранить содержимое регистров, так как они являются ресурсами, разделяемыми между программами. Обязательными для сохранения являются регистры cs, ip, flags. Эти регистры сохраняются микропроцессором автоматически. Сохранение остальных регистров - должно обеспечиваться программистом. Наиболее удобным местом хранения регистров является стек. После сохранения регистров в стеке микропроцессор сбрасывает бит флага IF (т.е.=0). Этим предотвращается возможность возникновения вложенных внешних прерываний и порча регистров исходной программы вследствие неконтролируемых действий со стороны программы - обработчика вложенного прерывания. После того как необходимые действия по сохранению контекста завершены, обработчик аппаратного прерывания может разрешить вложенные прерывания командой sti.

2. Переход к выполнению и выполнение программы обработки прерывания.

Здесь определяется источник прерывания и вызывается соответствующий обработчик

прерывания. В реальном режиме микропроцессора допускается 256 источников - по

количеству элементов таблицы векторов прерываний.

3. Возврат управления прерванной программе

Необходимо привести стек в состояние, в котором он был сразу после передачи

управления данной процедуре. Для этого программист должен указать необходимые

действия по восстановлению регистров и очистке стека. Этот участок необходимо

защитить от возможного искажения содержимого регистров (в результате появления

аппаратного прерывания) с помощью команды cli.

Различия в работе системы прерываний в реальном и защищенном режимах процессора x86 связаны с уровнем привилегий и доступа к системным ресурсам.

В реальном режиме процессора x86 система прерываний работает простым и прямым способом. Прерывания могут быть запрошены внешними устройствами или программно через инструкции процессора. При возникновении прерывания процессор сохраняет текущее состояние выполнения программы, переключается на обработчик прерывания и выполняет соответствующий код. В реальном режиме нет механизма защиты памяти или ограничения доступа к системным ресурсам, поэтому обработчик прерывания может выполняться с полными привилегиями.

В защищенном режиме процессора x86 система прерываний работает более сложным образом и предоставляет механизмы для защиты системных ресурсов и обеспечения безопасности. Процессор работает в режиме сегментации и пейджинга памяти, что позволяет разделять память между различными задачами и контролировать доступ к системным ресурсам. В защищенном режиме прерывания могут быть классифицированы по уровням привилегий, называемым дескрипторами привилегий (Privilege Levels, PL). У каждого прерывания есть свой уровень привилегий, и процессор проверяет, может ли текущая задача обработать прерывание на этом уровне. Если прерывание имеет более высокий уровень привилегий, чем текущая задача, процессор переключается на соответствующий уровень привилегий и выполняет обработчик прерывания.

Векторы прерываний - это адреса обработчиков прерываний, которые используются процессором для перехода к соответствующему коду при возникновении прерывания. Каждое прерывание имеет свой уникальный вектор, по которому процессор определяет местоположение обработчика в памяти.

В процессорах семейства x86, включая x86-32 (IA-32) и x86-64 (AMD64), векторы прерываний используются для определения адресов обработчиков прерываний в таблице прерываний. Таблица прерываний называется IDT (Interrupt Descriptor Table) и содержит записи, каждая из которых соответствует определенному вектору прерывания.

IDT состоит из 256 (в x86-32) или 256 (в x86-64) записей. Каждая запись в IDT содержит сегментный селектор и смещение, которые образуют 32-битный или 64-битный адрес обработчика прерывания. Селектор указывает на сегмент кода, а смещение - на смещение внутри этого сегмента.

При возникновении прерывания процессор использует номер прерывания (вектор) для индексации в IDT и получения адреса обработчика прерывания. Затем процессор переходит по этому адресу и начинает выполнение кода обработчика прерывания.

Векторы прерываний в x86 являются стандартизированными и определены архитектурой процессора. Они включают прерывания от внешних устройств, таких как таймер, клавиатура, сетевые адаптеры, а также прерывания, связанные с программными исключениями, ошибками и другими событиями.

1. Структура ядра и его функции. Объекты ядра. Основные операции над объектами ядра.

Возможны следующие типы структуры (архитектуры) ядра операционной системы:

• Монолитная структура ядра. (Все элементы монолитного ядра работают в едином адресном формате. При такой организации операционной системы все составляющие части её ядра выступают как элементы основной программы, применяют одни и те же системы организации данных и работают друг с другом, используя непосредственный вызов процедуры.)

• Модульная структура ядра. (Она отличается от классического монолитного ядра тем, что не требует общей реструктуризации ядра при различных вариациях аппаратной оснастки компьютеров. У модульных ядер есть возможность подгружать различные модули ядра, которые поддерживают нужное аппаратное оборудования, причём подзагрузка модуля возможна как в динамическом режиме, то есть без перезагрузки операционной системы, так и в статике, когда выполняется переконфигурирование системы и её перезагрузка.)

• Микроструктура ядра (решает лишь самые простые задачи по управлению процессами и имеет небольшой комплект абстракций оборудования. Основная часть функций выполняется специальными процессами пользователя, которые называются сервисами. Главным признаком микроядра можно считать распределение практически всех драйверов и элементов в процессах сервиса.)

• Экзо структура ядра. (это ядро операционной системы, которое предоставляет только возможность взаимного обмена между процессами и надёжного распределения и высвобождения ресурсов.)

• Нано структура ядра. (имеет такую структуру, при которой очень простое ядро решает лишь проблему обработки аппаратного прерывания программы, которое генерируют различные блоки компьютера.)

• Гибридная структура ядра. (представляет собой модификацию микроядра, которая позволяет ускорить работу системы.)

• Комбинированная структура ядра (комбинирование вышеперечисленных вариантов построения структуры ядра операционной системы.)

Функции, входящие в состав ядра, можно разделить на два класса. 1 класс. Функции для решения внутрисистемных задач организации вычислительного процесса (переключение контекстов процессов, загрузка/выгрузка страниц, обработка прерываний). Эти функции недоступны для приложений. 2 класс. Функции для поддержки приложений (доступны приложениям). Эти функции создают для приложений так называемую прикладную программную среду и образуют интерфейс прикладного программирования - API. Приложения обращаются к ядру с запросами - системными вызовами. Функции API обслуживают системные вызовы - предоставляют доступ к ресурсам системы в удобной и компактной форме, без указания деталей их физического расположения.

Типичные объекты режима ядра включают следующие категории объектов:

• объекты устройств;

• объекты файлов;

• символические ссылки;

• разделы реестра;

• потоки и процессы;

• объекты диспетчера ядра, такие как объекты событий и объекты мьютексов;

• объекты обратного вызова;

• объекты section.

В качестве примера конкретных объектов можно привести: маркеры доступа, файлы, проекции файлов, порты завершения ввода вывода

Метод open вызывается всякий раз, когда создается или открывается объект и создается его новый описатель. Создание новых объектов, или открытие по имени уже существующих, приложение может осуществить при помощи CreateFile, CreateSemaphore, OpenSemaphore и т.д. Это библиотечные процедуры, за которыми стоят сервисы Windows и методы объектов. В случае успешного выполнения создается 64-битный описатель в таблице описателей процесса в памяти ядра. Рассмотрим операции с объектами ядра на примере работы с файлами. Прежде чем прочитать данные из файла, нужно получить описатель. Для этого используется «функция» CreateFile. Функция CreateFile, несмотря на первое впечатление от ее имени, используется и при работе с уже существующими файлами, так как слово File здесь обозначает не файл на диске, а объект ядра типа File. Полученный описатель затем нужно использовать как первый параметр в различных функциях, например, ReadFile и WriteFile, для работы с реальным файлом. Во избежание утечки памяти всегда рекомендуется закрывать объект, когда в нем отпала надобность. Впрочем, по окончании работы процесса система закрывает все его объекты. Одной из таких функций является функция CloseHandle. Некоторые дескрипторы требуют своей функции закрытия, например, функция FindClose должна применяться для закрытия дескриптора, полученного от функции FindFirstFile. Некоторые объекты ядра имеют разрешение Synchronize. Это означает, что их можно использовать как для синхронизации потоков.

1. Утилиты. Системные обрабатывающие программы. Библиотеки процедур. Программы дополнительных услуг.

Утилиты - вспомогательное программное обеспечение — тип системного программного обеспечения, используемого для управления, организации, оптимизации и улучшения функционирования компьютерной системы. Основные типы утилит: Антивирус, Системы управления файлами, Инструменты сжатия(служебные программы для сжатия больших файлов и уменьшения их размера), Инструменты управления диском, Инструменты очистки диска, Дефрагментация диска(Дефрагментация означает переупорядочение файлов и сохранение их в смежных местах памяти), Утилита резервного копирования.

Обрабатывающие системные программы отличаются от управляющих программ как по своим функциям, так и по способу их инициирования (запуска). Основные функции обрабатывающих программ: 1) перенос информации. Перенос может выполняться между различными устройствами или в пределах одного устройства. При этом под устройствами понимаются: ОП, устройства ВП, устройства ввода-вывода; 2) преобразование информации. То есть после считывания информации с устройства обрабатывающая программа преобразует эту информацию, а только затем записывает ее на это же или на другое устройство. В зависимости от того, какая из этих двух функций является основной, обрабатывающие системные программы делятся на утилиты и лингвистические процессоры. Основной функцией утилиты является перенос информации, а основная функция лингвистического процессора – перевод описания алгоритма с одного языка на другой. Сущность алгоритма при этом сохраняется, но форма его представления, ориентированная на программиста, преобразуется в форму, ориентированную на ЦП.

Библиотеки — это набор функций, которые могут использоваться в различных программах. Библиотеки могут быть статические (библиотека привязывается к определенной программе или софт содержит данную библиотеку в своем теле.) и динамическими (библиотеки грузятся в оперативную память и используются).

В категорию «Программы дополнительных услуг» входит большое количество разнообразных программ: специальный вариант пользовательского интерфейса, калькулятор, некоторые игры (какие, например, поставляются в составе ОС).

1. Понятие процесса. Системные и пользовательские процессы. Операции над процессами.

Процесс (process) - это некоторая часть (единица) работы, создаваемая операционной системой. Чтобы некоторую часть работы можно было назвать процессом, она должна иметь адресное пространство, назначаемое операционной системой, и идентификатор, или идентификационный номер (id процесса). Процесс должен обладать определенным статусом и иметь свой элемент в таблице процессов. В соответствии со стандартом POSIX он должен содержать один или несколько потоков управления, выполняющихся в рамках его Процесс состоит из множества выполняющихся инструкций, размещенных в адресном пространстве этого процесса. Адресное пространство процесса распределяется между инструкциями, данными, принадлежащими процессу, и стеками, обеспечивающими вызовы функций и хранение локальных переменных.

Процессы, которые выполняют системный код, называются системными и применяются к системе в целом. Они занимаются выполнением таких служебных задач, как распределение памяти, обмен страницами между внутренним и вспомогательным запоминающими устройствами, контроль устройств и т.п. Они также выполняют некоторые задачи «по поручению» пользовательских процессов, например, делают запросы на ввод-вывод данных, выделяют память и т.д.

Пользовательские процессы выполняют собственный код и иногда обращаются к системным функциям. Выполняя собственный код, пользовательский процесс пребывает в пользовательском режиме (user mode). В пользовательском режиме процесс не может выполнять определенные привилегированные машинные команды.

Основные операции:

Запуск процесса (из числа процессов, находящихся в состоянии готовность, операционная система выбирает один процесс для последующего исполнения). Приостановка процесса (работа процесса, находящегося в состоянии исполнение, приостанавливается в результате какого-либо прерывания). Блокирование процесса (процесс блокируется, когда он не может продолжать работу, не дождавшись возникновения какого-либо события в вычислительной системе). Разблокирование процесса (после возникновения в системе какого-либо события операционной системе нужно точно определить, какое именно событие произошло). Переключение контекста.

1. Организация межпроцессного взаимодействия в ОС. Сигналы. Каналы. Классические проблемы межпроцессного взаимодействия.

При выполнении параллельных процессов может возникать проблема, когда каждый процесс, обращающийся к разделяемым данным, исключает для всех других процессов возможность одновременного с ним обращения к этим данным - это называется взаимоисключением (mutual exclusion). Ресурс, который допускает обслуживание только одного пользователя за один раз, называется критическим ресурсом. Если несколько процессов хотят пользоваться критическим ресурсом в режиме разделения времени, им следует синхронизировать свои действия таким образом, чтобы этот ресурс всегда находился в распоряжении не более чем одного их них. Для организации коммуникации между одновременно работающими процессами применяются средства межпроцессного взаимодействия (Interprocess Communication - IPC). Выделяются три уровня средств IPC:

• локальный(привязаны к процессору и возможны только в пределах компьютера(каналы, разделяемая память и очереди сообщений));

• удаленный(обеспечивают взаимодействие как в пределах одного процессора, так и между программами на различных процессорах, соединенных через сеть(удаленные вызовы процедур, сокеты Unix, а также интерфейс транспортного уровня фирмы Sun));

• высокоуровневый (подразумеваются пакеты программного обеспечения, которые реализуют промежуточный слой между системной платформой и приложением. Эти пакеты предназначены для переноса уже испытанных протоколов коммуникации приложения на более новую архитектуру).

Сигнал в операционных системах семейства Unix — асинхронное уведомление процесса о каком-либо событии, один из основных способов взаимодействия между процессами. Когда сигнал послан процессу, операционная система прерывает выполнение процесса, при этом, если процесс установил собственный обработчик сигнала, операционная система запускает этот обработчик, передав ему информацию о сигнале, если процесс не установил обработчик, то выполняется обработчик по умолчанию.

Отдельные сигналы подразделяются на три класса: системные сигналы (ошибка аппаратуры, системная ошибка и т.д.); сигналы от устройств; сигналы, определенные пользователем

Канал (pipe) представляет собой средство связи стандартного вывода одного процесса со стандартным вводом другого. Каналы старейший из инструментов IPC, существующий приблизительно со времени появления самых ранних версий оперативной системы UNIX. Для реализации IPC возможно использование полудуплексных и/или именованных каналов (FIFO).

Классические проблемы межпроцессного взаимодействия

Синхронный доступ. Если все процессы считывают данные из файла, то в большинстве случае проблем не возникает. Однако, при попытке одним из процессов изменить этот файл, могут возникнуть так называемые конкурентные ситуации

Дисциплина доступа. Если нужно, чтобы как можно большее количество пользователей могли записывать данные, организуется так называемая очередь (по правилу «один пишет, несколько читают»). Практически организуется две очереди: одна — для чтения, другая — для записи. Такую дисциплину доступа можно организовать с помощью барьеров (блокировок). При этом создается общий барьер для считывателей, так как несколько процессов могут одновременно считывать данные, а также отдельный барьер для процесса-писателя. Такая организация предотвращает взаимные помехи в работе.

Голодание процессов. Организация дисциплины доступа может привести к ситуации, когда процесс будет длительно ждать своей очереди для записи данных. Поэтому иногда нужно организовывать очереди с приоритетами

Задача обедающих философов(тарелки со спагетти, которые можно есть только 2 вилками, решается с помощью семафора и массива состояний)

Задача читателей и писателей(бронирование авиабилетов)

1. Концепция потока. Параллельное исполнение потоков. Главный поток процесса.

Поток выполнения — наименьшая единица обработки, исполнение которой может быть назначено ядром операционной системы. Несколько потоков выполнения могут существовать в рамках одного и того же процесса и совместно использовать ресурсы, такие как память, тогда как процессы не разделяют этих ресурсов. В частности, потоки выполнения разделяют инструкции процесса и его контекст.

Основные характеристики концепции потоков:

Легковесность: Потоки являются легковесными, поскольку они используют общие ресурсы процесса, такие как память и файловые дескрипторы. Создание и переключение контекста между потоками требует меньше ресурсов, чем создание и переключение контекста между процессами.

Совместное использование ресурсов: Все потоки внутри одного процесса имеют общее адресное пространство и ресурсы процесса, такие как память, файлы и устройства ввода-вывода. Это позволяет им совместно использовать данные и взаимодействовать друг с другом, не требуя явного механизма межпроцессного взаимодействия.

Параллельное выполнение: Потоки могут выполняться параллельно, что позволяет использовать многопроцессорные системы более эффективно.

Структура и планирование: Потоки внутри процесса могут иметь свою собственную структуру и планирование выполнения. Также существуют различные алгоритмы планирования, которые определяют порядок выполнения потоков и распределение процессорного времени между ними.

Преимущества потоков.

• Потоки довольно просто обмениваются данными по сравнению с процессами.

• Создавать потоки для ОС проще и быстрее (иногда на порядок), чем создавать процессы. Недостатки потоков.

• При программировании приложения с множественными потоками необходимо постоянно думать о потокобезопасности (т. н. thread safety). Приложения, выполняющиеся через множество процессов, не имеют таких требований.

• Один неправильно работающий поток может повредить остальные, так как потоки делят общее адресное пространство. Процессы более изолированы друг от друга.

• Потоки конкурируют друг с другом в адресном пространстве. Стек и thread-local storage, занимая часть виртуального адресного пространства процесса, тем самым делают его недоступным для других потоков. Для встраиваемых (embedded) систем в условиях ограниченности ресурсов такое ограничение может иметь существенное значение.

Термин "параллельность" рассматривается в контексте вытесняющей многозадачности операционной системы - то есть, операционная система выделяет потоку некоторый квант времени, а затем переключается на другой поток. Параллельный поток может добровольно отдавать часть своего кванта времени, если он переходит в состоянии ожидания некоторого события. Выполнение потока может быть прервано более приоритетным потоком.

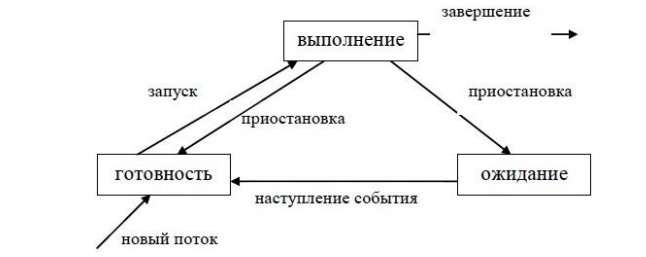
Многопоточность: процесс, порождённый в операционной системе, может состоять из нескольких потоков, выполняющихся «параллельно», то есть без предписанного порядка во времени.

Наиболее важные виды: совместное использование разделяемых данных, асинхронное взаимодействие, синхронное взаимодействие. В первом случае необходим взаимно-исключающий доступ к данным - недопустимо, чтобы один поток читал данные, в то время как другой поток их изменял. При асинхронном взаимодействии вводится посредник-буфер между параллельными потоками. Иногда этот способ называют взаимодействием с помощью обмена сообщениями. При синхронном взаимодействии оба взаимодействующих потока подходят к точке синхронизации, обмениваются данными и затем продолжают работать параллельно и независимо. Если один из потоков подошел к точке синхронизации раньше, то он дожидается партнера.

По умолчанию процесс создается с одним потоком, называемым главным или основным потоком. Потоки в Unix по существу являются дешевой копией процессов и по аналогии с процессами предоставляют механизм для одновременного выполнения нескольких параллельных задач в рамках одного приложения.

1. Диаграммы состояния потоков. Понятие контекста и переключения контекста.

Для каждого созданного потока в системе предусматриваются три возможных его состояния: состояние выполнения, когда код потока выполняется процессором; на однопроцессорных платформах в этом состоянии в каждый момент времени может находиться только один поток; состояние готовности к выполнению, когда поток готов продолжать свою работу и ждет освобождения ЦП; состояние ожидания наступления некоторого события; в этом случае поток не претендует на время ЦП, пока не наступит определенное событие (завершение операции ввода/вывода, освобождение необходимого потоку занятого ресурса, сигнала от другого потока), часто такие потоки называют блокированными.



«готовность» → «выполнение»: система в соответствии с алгоритмом планирования выбирает для выполнения текущий поток, предоставляя ему ЦП

• «выполнение» → «готовность»: поток готов продолжать свою работу, но система принимает решение прервать его выполнение; чаще всего это происходит по следующим двум причинам: o завершается выделенное потоку время владения процессором; o в числе готовых к выполнению появляется более приоритетный поток по сравнению с текущим;

• «выполнение» → «ожидание»: дальнейшее исполнение кода текущего активного потока невозможно без наступления некоторого события, и поэтому активный поток прерывает свое выполнение и переводится системой в состояние ожидания (блокируется);

• «ожидание» → «готовность»: в системе происходит некоторое событие, наступление которого ожидает один из блокированных потоков, и поэтому система переводит этот поток в состояние готовности (разблокирует), после чего он будет учитываться системой при планировании порядка предоставления ЦП; • наконец, поток может нормально или аварийно завершить свое выполнение, после чего система удаляет его дескриптор из своей внутренней структуры, и тем самым поток перестает существовать.

Переключение контекста (context switch) - в многозадачных ОС и средах - процесс прекращения выполнения процессором одной задачи (процесса, потока, нити) с сохранением всей необходимой информации и состояния, необходимых для последующего продолжения с прерванного места, и восстановления и загрузки состояния задачи, к выполнению которой переходит процессор.

В процедуру переключения контекста входит так называемое планирование задачи — процесс принятия решения, какой задаче передать управление.

При переключении контекста происходит сохранение и восстановление следующей информации:

• Регистровый контекст регистров общего назначения (в том числе флаговый регистр)

• Контекст состояния сопроцессора с плавающей точкой / регистров MMX (x86)

• Состояние регистров SSE, AVX (x86)

• Состояние сегментных регистров (x86)

• Состояние некоторых управляющих регистров (например, регистр CR3, отвечающий за страничное отображение памяти процесса) (

Кроме того, что очень важно, при переключении контекста происходят следующие аппаратные действия, влияющие на производительность:

• Происходит очистка конвейера команд и данных процессора

• Очищается TLB,

С точки зрения прикладного уровня переключение контекста можно разделить на добровольное (voluntary) и принудительное (non-voluntary): выполняющийся процесс/поток может сам передать управление другому потоку либо ядро может насильно отобрать у него управление.

1. Многозадачность в ОС. Типы многозадачности.

Многозадачность (multitasking) - свойство операционной системы или среды выполнения обеспечивать возможность параллельной (или псевдопараллельной) обработки нескольких задач. Истинная многозадачность операционной системы возможна только в распределённых вычислительных системах.

Более развитые многозадачные системы проводят распределение ресурсов динамически, когда задача стартует в памяти или покидает память в зависимости от её приоритета и от стратегии системы.

Основной трудностью реализации многозадачной среды является её надёжность, выраженная в защите памяти, обработке сбоев и прерываний, предохранении от зависаний и тупиковых ситуаций. Кроме надёжности, многозадачная среда должна быть эффективной. Затраты ресурсов на её поддержание не должны: мешать процессам проходить, замедлять их работу, резко ограничивать память.

Существует два типа многозадачности: процессная многозадачность (Здесь программа — наименьший элемент управляемого кода, которым может управлять планировщик операционной системы. Более известна большинству пользователей (работа в текстовом редакторе и прослушивание музыки)) и поточная многозадачность (Наименьший элемент управляемого кода — поток (одна программа может выполнять 2 и более задачи одновременно).).

Псевдопараллельная многозадачность

К псевдопараллельной многозадачности можно отвести следующие типы: простое переключение(Тип многозадачности, при котором операционная система одновременно загружает в память два или более приложений, но процессорное время предоставляется только основному приложению. Для выполнения фонового приложения оно должно быть активизировано.), кооперативная(Тип многозадачности, при котором следующая задача выполняется только после того, как текущая задача явно объявит себя готовой отдать процессорное время другим задачам.) многозадачность и вытесняющая (Вид многозадачности, в котором операционная система сама передает управление от одной выполняемой программы другой в случае завершения операций ввода-вывода, возникновения событий в аппаратуре компьютера, истечения таймеров и квантов времени, или же поступлений тех или иных сигналов от одной программы к другой.) многозадачность.

1. Иерархия, приоритеты и планирование потоков. Динамические уровни приоритетов.

Иерархия потоков представляет собой структуру, в которой потоки организованы по отношению родитель-потомок. В такой структуре каждый поток, кроме главного, может порождать другие потоки, которые называются дочерними потоками. Дочерние потоки наследуют некоторые свойства от родительского потока, такие как приоритет и ресурсы.

За время существования ОС было предложено и реализовано несколько принципов управления потоками. В настоящее время большинство универсальных ОС используют метод вытесняющей многозадачности (preemptive multitasking), который тоже имеет несколько разновидностей. В основе метода лежат два важнейших и достаточно понятных принципа: квантование времени ЦП и приоритеты потоков.

Квантование означает, что каждому потоку система выделяет определенный интервал времени (квант), в течение которого процессор потенциально может выполнять код этого потока. По завершении выделенного кванта планировщик принудительно переключает процессор на выполнение другого готового потока (если, конечно, такой есть), переводя старый активный поток в состояние готовности.  
Можно связать величину кванта с приоритетом потока. Приоритет определяет важность потока и влияет на частоту запуска потока и, возможно, на величину выделяемого кванта. Интуитивно понятно, что потоки могут иметь разную степень важности: системные – более высокую (иначе ОС не сможет решать свои задачи), прикладные – менее высокую.

Многие ОС позволяют группировать потоки по их важности, выделяя три группы, или класса:

• потоки реального времени с максимально высоким уровнем приоритета;

• системные потоки с меньшим уровнем приоритета;

• прикладные потоки с самым низким приоритетом.

Если приоритет потока может меняться системой, то такие приоритеты называют динамическими, иначе – фиксированными.

Потоки планируются на основе их приоритета планирования [4]. Каждому потоку назначается приоритет планирования. Уровни приоритета варьируются от нуля (самый низкий приоритет) до 31 (наивысший приоритет). Только поток обнуления страниц памяти может иметь нулевой приоритет.

Каждый процесс принадлежит к одному из следующих классов приоритетов:

• IDLE\_PRIORITY\_CLASS

• BELOW\_NORMAL\_PRIORITY\_CLASS

• NORMAL\_PRIORITY\_CLASS

• ABOVE\_NORMAL\_PRIORITY\_CLASS

• HIGH\_PRIORITY\_CLASS

• REALTIME\_PRIORITY\_CLASS

Ниже приведены уровни приоритета в каждом классе приоритета.

• THREAD\_PRIORITY\_IDLE

• THREAD\_PRIORITY\_LOWEST

• THREAD\_PRIORITY\_BELOW\_NORMAL

• THREAD\_PRIORITY\_NORMAL

• THREAD\_PRIORITY\_ABOVE\_NORMAL

• THREAD\_PRIORITY\_HIGHEST

• THREAD\_PRIORITY\_TIME\_CRITICAL

Приоритеты в Linux

Существует два вида приоритетов, связанных с каждым процессом - один - это значение "приятности", которое варьируется от -20 (самый высокий приоритет) до 19 (самый низкий приоритет), а другой - приоритет реального времени, варьирующийся от 1 до 99. Когда мы хотим установить приоритет процесса, мы изменяем значение "приятности" процесса. Причем, уменьшать приоритет можно с правами обычного пользователя, но, чтобы его увеличить, нужны права суперпользователя.

Приоритет nice и приоритет планировщика процессов ядра ОС — разные числа. Число nice — приоритет, который пользователь хотел бы назначить процессу. Приоритет планировщика — действительный приоритет, назначенный процессу планировщиком.

Планировщик процессов ядра ОС Linux поддерживает приоритеты от 0 (реальное время) до 139 включительно. Приоритеты -20…+19 утилиты или команды nice соответствуют приоритетам 100…139 планировщика процессов.

Алгоритмы планирования в пакетных системах

Первым пришел — первым обслужен. Сначала самое короткое задание. Приоритет

наименьшему времени выполнения

Алгоритмы планирования в интерактивных системах

Циклическое планирование.( В этом алгоритме каждый поток получает равное количество процессорного времени, после чего переключается на следующий поток. Если поток не завершил свою работу, он будет помещен в конец очереди.) Приоритетное планирование. Использование нескольких очередей.( Цель алгоритма — время от времени выделять процессам, ограниченным скоростью вычислений, более существенные кванты времени, вместо того чтобы слишком часто выделять им небольшие кванты времени) Выбор следующим самого короткого процесса. Гарантированное планирование.( Этот подход к планированию заключается в предоставлении пользователям реальных обещаний относительно производительности, а затем в выполнении этих обещаний. Одно из обещаний, которое можно дать и просто выполнить, заключается в следующем: если в процессе работы в системе зарегистрированы n пользователей, то вы получите 1/n от мощности центрального процессора.) Лотерейное планирование.( Основная идея состоит в раздаче процессам лотерейных билетов на доступ к различным системным ресурсам, в том числе к процессорному времени. Когда планировщику нужно принимать решение, в случайном порядке выбирается лотерейный билет, и ресурс отдается процессу, обладающему этим билетом. Применительно к планированию процессорного времени система может проводить лотерейный розыгрыш 50 раз в секунду, и каждый победитель будет получать в качестве приза 20 мс процессорного времени. Более важным процессам, чтобы повысить их шансы на выигрыш, могут выдаваться дополнительные билеты.) Справедливое планирование.( В этой модели каждому пользователю распределяется некоторая доля процессорного времени и планировщик выбирает процессы, соблюдая это распределение. Таким образом, если каждому из двух пользователей было обещано по 50 % процессорного времени, то они его получат, независимо от количества имеющихся у них процессов.)

Система не повышает приоритет потоков с базовым уровнем приоритета от 16 до 31. Динамические повышения приоритета получают только потоки с базовым приоритетом от 0 до 15.

Функцию повышения приоритета можно отключить, вызвав функцию SetProcessPriorityBoost или SetThreadPriorityBoost. Чтобы определить, отключена ли эта функция, вызовите функцию GetProcessPriorityBoost или GetThreadPriorityBoost. После повышения динамического приоритета потока планировщик уменьшает этот приоритет на один уровень каждый раз, когда поток завершает квант времени, пока поток не вернется к базовому приоритету. Динамический приоритет потока никогда не меньше базового приоритета.

Система повышает динамический приоритет потока для повышения его отклика следующим образом.

• При переносе процесса, использующего NORMAL\_PRIORITY\_CLASS, на передний план планировщик увеличивает класс приоритета процесса, связанного с окном переднего плана, чтобы он был больше или равен классу приоритета любых фоновых процессов. Класс приоритета возвращается к исходному параметру, когда процесс больше не находится на переднем плане.

• Когда окно получает входные данные, такие как сообщения таймера, сообщения мыши или ввод с клавиатуры, планировщик повышает приоритет потока, которому принадлежит окно.

• При выполнении условий ожидания для заблокированного потока планировщик повышает приоритет потока. Например, когда завершается операция ожидания, связанная с диском или вводом-выводом с клавиатуры, поток получает повышение приоритета.

1. Синхронизация и взаимоблокировка ресурсов. Механизмы синхронизации.

Ситуации, в которых два (и более) процесса считывают или записывают данные одновременно и конечный результат зависит от того, какой из них был первым, называются состояниями состязания.

Взаимоблокировка в группе процессов возникает в том случае, если каждый процесс из этой группы ожидает события, наступление которого зависит исключительно от другого процесса из этой же группы.

Критическая область — часть программы, в которой есть обращение к совместно используемым данным. Соответственно, критический ресурс — тот ресурс, к которому осуществляется одновременный доступ

Семафоры

Целочисленная переменная для подсчета количества активизаций, отложенных на будущее. Значение семафора может быть равно 0, что будет свидетельствовать об отсутствии сохраненных активизаций, или иметь какое-нибудь положительное значение, если ожидается не менее одной активизации.

Дейкстра предложил использовать две операции с семафорами, которые сейчас обычно называют down и up (обобщения sleep и wakeup соответственно). Операция down выясняет, отличается ли значение семафора от 0. Если отличается, она уменьшает это значение на 1 (то есть использует одну сохраненную активизацию) и продолжает свою работу. Если значение равно 0, процесс приостанавливается, не завершая в этот раз операцию down. И проверка значения, и его изменение, и, возможно, приостановка процесса осуществляются как единое и неделимое атомарное действие. Тем самым гарантируется, что с началом семафорной операции никакой другой процесс не может получить доступ к семафору до тех пор, пока операция не будет завершена или заблокирована.

Операция up увеличивает значение, адресуемое семафором, на 1. Если с этим семафором связаны один или более приостановленных процессов, способных завершить ранее начатые операции down, система выбирает один из них (к примеру, произвольным образом) и позволяет ему завершить его операцию down.

Мьютекс — это совместно используемая переменная, которая может находиться в одном из двух состояний: заблокированном или незаблокированном. Следовательно, для их представления нужен только один бит, но на практике зачастую используется целое число, при этом нуль означает незаблокированное, а все остальные значения — заблокированное состояние. Для работы с мьютексами используются две процедуры. Когда потоку (или процессу) необходим доступ к критической области, он вызывает процедуру mutex\_lock. Если мьютекс находится в незаблокированном состоянии (означающем доступность входа в критическую область), вызов проходит удачно и вызывающий поток может свободно войти в критическую область. В то же время, если мьютекс уже заблокирован, вызывающий поток блокируется до тех пор, пока поток, находящийся в критической области, не завершит свою работу и не вызовет процедуру mutex\_unlock. Если на мьютексе заблокировано несколько потоков, то произвольно выбирается один из них, которому разрешается воспользоваться заблокированностью других потоков.

Объект критического раздела обеспечивает синхронизацию, аналогичную той, которая предоставляется объектом мьютекса, за исключением того, что критический раздел может использоваться только потоками одного процесса. Объекты критических разделов нельзя совместно использовать в процессах.

Объект события - это объект синхронизации, состояние которого может быть явно задано с помощью функции SetEvent. Существуют два типа объекта события. Событие сброса вручную. Объект события, состояние которого остается сигнальным до тех пор, пока функция ResetEvent явно не будет сброшена на незначимую. Во время передачи сигнала можно освободить любое количество ожидающих потоков или потоков, которые впоследствии указывают один и тот же объект события в одной из функций ожидания. Событие автоматического сброса. Объект события, состояние которого остается сигнальным до тех пор, пока не будет освобожден один поток ожидания, в этот момент система автоматически устанавливает состояние без знака. Если ожидающих потоков нет, состояние объекта события остается сигнальным.

Барьер. Этот механизм синхронизации предназначен для групп процессов. Некоторые приложения разбиты на фазы и следуют правилу, согласно которому ни один из процессов не может перейти к следующей фазе, пока все процессы не будут готовы перейти к следующей фазе. Добиться выполнения этого правила можно с помощью барьеров, поставленных в конце каждой фазы. Когда процесс достигает барьера, он блокируется до тех пор, пока этого барьера не достигнут все остальные процессы. Это позволяет синхронизировать группы процессов.

Атомарные операции - это операции, которые гарантированно выполняются целикком и неделимы для других потоков. Это означает, что атомарные операции не могут быть прерваны или перехвачены другими потоками. Атомарные операции обычно используются для простых операций чтения, записи или модификации общих переменных без необходимости использования блокировок или других механизмов синхронизации.

1. Взаимоблокировка ресурсов в многозадачных системах. Решение задачи взаимоблокировки ресурсов.

Взаимоблокировка в группе процессов возникает в том случае, если каждый процесс из этой группы ожидает события, наступление которого зависит исключительно от другого процесса из этой же группы.

Коффман показал, что для возникновения ресурсных взаимоблокировок должны выполняться четыре условия:

1. Условие взаимного исключения. Каждый ресурс либо выделен в данный момент только одному процессу, либо доступен.

2. Условие удержания и ожидания. Процессы, удерживающие в данный момент ранее выделенные им ресурсы, могут запрашивать новые ресурсы.

3. Условие невыгружаемости. Ранее выделенные ресурсы не могут быть принудительно отобраны у процесса. Они должны быть явным образом высвобождены тем процессом, который их удерживает.

4. Условие циклического ожидания. Должна существовать кольцевая последовательность из двух и более процессов, каждый из которых ожидает высвобождения ресурса, удерживаемого следующим членом последовательности.

Для возникновения ресурсной взаимоблокировки должны соблюдаться все четыре условия. Если одно из них не соблюдается, ресурсная взаимоблокировка невозможна.

Решение задачи взаимоблокировки ресурсов

Чаще всего для борьбы с взаимными блокировками используются четыре стратегии:

1. Игнорирование проблемы. Может быть, если вы проигнорируете ее, она проигнорирует вас.

2. Обнаружение и восстановление. Дайте взаимоблокировкам проявить себя, обнаружьте их и выполните необходимые действия.

3. Динамическое уклонение от них за счет тщательного распределения ресурсов.

4. Предотвращение за счет структурного подавления одного из четырех условий, необходимых для их возникновения.

1. Компьютерное время. Ожидаемые таймеры.

Географический часовой пояс - условная полоса на земной поверхности шириной ровно 15° (±7,5° относительно среднего меридиана). Средним меридианом нулевого часового пояса считается гринвичский меридиан. Административный часовой пояс — участок земной поверхности, на котором в соответствии с некоторым законом установлено определённое официальное время.

UT1, или универсальное время — основная версия всемирного времени. Хотя концептуально это местное среднее солнечное время на долготе 0°, но измерения среднего Солнца трудноосуществимы, поэтому UT1 вычисляется пропорционально углу вращения Земли относительно квазаров, а точнее, относительно международной небесной системы координат (ICRS). UTC (всемирное координированное время) — атомная шкала времени, аппроксимирующая UT1. Это международный стандарт, на котором базируется время часовых поясов. В UTC в качестве единицы времени используется секунда СИ, поэтому UTC идёт синхронно с международным атомным временем (TAI). Обычно в сутках UTC 86 400 секунд СИ, но для поддержания расхождения UTC и UT1 не более чем 0,9 с, при необходимости, 30 июня или 31 декабря добавляется (или, теоретически, вычитается) секунда координации

UTC (Всемирное Координированное Время) - одно из общеизвестных названий для UTC+0 часового пояса, который на 0ч. впереди UTC (Всемирного координированного времени). Используется как стандартное время. Нет отклонения от Гринвича (UTC+00). Локальное (местное) время служит для того, чтобы солнечный день приходился приблизительно на один и тот же промежуток времени

Unix-время (также POSIX-время) — система описания моментов во времени, принятая в Unix и других POSIX-совместимых операционных системах [3]. Определяется как количество секунд, прошедших с полуночи (00:00:00 UTC) 1 января 1970 года; этот момент называют эпохой Unix.

Функции Posix:

Календарное время включает в себя год, месяц, день месяца и текущее время суток. Для хранения календарного времени используется тип time\_t. В POSIX-системах календарное время хранится как количество секунд, прошедшее от «начала эпохи», причём для простоты считается, что каждый 4-й год всегда високосный.

В настоящее время в POSIX-системах тип time\_t эквивалентен типу long. Диапазона значений типа long (32 бита) достаточно для представления дат, лежащих в отрезке длиной примерно 136 лет. Поскольку отрицательные числа используются для представления дат до 1970 года, переполнение текущего представления типа time\_t может произойти в 2038 году, если, конечно, до этого момента представление не будет изменено.

Функция time позволяет получить текущее системное календарное время(time\_t time(time\_t \*pval);).

Функции localtime(в определённом пользователем или системой часовом поясе), gmtime(в UTC) преобразовывают дату из секундного представления time\_t в структурное представление struct tm.

Функция mktime конвертирует календарное время местного часового пояса в развёрнутом структурном формате в секундное представление

Функции ctime и asctime конвертируют календарное время в символьную строку

Функция strftime позволяет получать строковое представление времени согласно заданной форматной строке.

Объект таймера ожидания - это объект синхронизации, состояние которого по достижении указанного срока устанавливается в значение Signaled. Существует два типа таймеров ожидания, которые можно создать: сброс вручную и синхронизация. Таймер любого типа также может быть периодическим. Таймер сброса вручную - таймер, состояние которого остается сигнальным до вызова SetWaitableTimer, чтобы установить новое время выполнения. Таймер синхронизации - таймер, состояние которого остается сигнальным до тех пор, пока поток не завершит операцию ожидания в объекте таймера. Периодический таймер - таймер, который повторно активируется каждый раз, когда истечет указанный период, пока таймер не будет сброшен или отменен. Периодический таймер - это либо периодический таймер сброса вручную, либо периодический таймер синхронизации.

Всеобщее распространение Интернета потребовало синхронизации работы различных процессов в серверах и программах клиента. Для этого используется сетевой протокол задания времени NTP. Он предусматривает возможности работы с иерархически распределенными первичными эталонами (такими как синхронизуемые радиочасы).

NTP в основном используется в следующих сценариях, когда часы всех сетевых устройств должны быть согласованы: • Управление сетью: Синхронизированное время используется в качестве эталона, когда система управления сетью (NMS) анализирует журналы и отладочную информацию, собранную с разных устройств. • Система начисления платы: для обеспечения точности и достоверности информации о начислении платы требуется синхронизированное время. • Несколько систем взаимодействуют по одному и тому же сложному событию: системы должны использовать одни и те же часы в качестве эталона, чтобы обеспечить правильную последовательность операций. • Инкрементное резервное копирование между сервером резервного копирования и клиентом: синхронизированное время обеспечивает целостность данных резервного копирования,

Принцип работы протокола NTP Клиент и сервер добавляют к сообщению метки времени при каждом отправлении и получении сообщения. В результате при получении ответа от сервера клиент имеет четыре метки времени: T1, T2, T3 и T4.

В примере предполагается, что односторонняя задержка канала равна Delay, а разница во времени между клиентом NTP и сервером NTP равна Offset (текущее время сервера NTP минус время клиента NTP). Формулы расчета следующие: T2 - T1 = Delay + Offset T4 - T3 = Delay – Offset. Следовательно, смещение рассчитывается следующим образом: Offset = [(T2 - T1) - (T4 - T3)]/2. Клиент NTP настраивает свои часы на основе значения смещения для достижения синхронизации с сервером NTP.

1. Управление памятью: адресное пространство процесса, организация памяти, основные механизмы управления памятью, концепция рабочего множества.

Адресное пространство - это набор адресов, который может быть использован процессом для обращения к памяти. У каждого процесса имеется собственное адресное пространство, независимое от того адресного пространства, которое принадлежит другим процессам (за исключением тех особых обстоятельств, при которых процессам требуется совместное использование их адресных пространств).

Операционная система может размещаться в нижней части адресов, в оперативном запоминающем устройстве (ОЗУ), или, по-другому, в памяти с произвольным доступом — RAM (Random Access Memory). Она может размещаться также в постоянном запоминающем устройстве (ПЗУ), или, иначе, в ROM (Read-Only Memory), в верхних адресах памяти. Или же драйверы устройств могут быть в верхних адресах памяти, в ПЗУ, а остальная часть системы — в ОЗУ, в самом низу. Первая модель прежде использовалась на универсальных машинах и мини-компьютерах, а на других машинах — довольно редко. Вторая модель использовалась на некоторых КПК и встроенных системах. Третья модель использовалась на ранних персональных компьютерах (например, на тех, которые работали под управлением MS-DOS), где часть системы, размещавшаяся в ПЗУ, называлась базовой системой ввода-вывода — BIOS (Basic Input Output System). Недостаток моделей(1 и 3) заключается в том, что ошибка в программе пользователя может затереть операционную систему, и, возможно, с весьма пагубными последствиями

Управление памятью является важной задачей для эффективного использования ресурсов компьютерной системы. Основные механизмы управления памятью включают следующие:

Разделение памяти: Операционная система разделяет физическую память между различными процессами или задачами, чтобы каждый из них имел свое собственное адресное пространство. Это обеспечивает изоляцию между процессами и защиту от несанкционированного доступа.

Виртуальная память: Операционная система может использовать механизм виртуальной памяти, который позволяет процессам использовать больше памяти, чем есть физическая память в системе. Виртуальная память использует комбинацию физической памяти и дискового пространства для хранения данных. Когда процесс обращается к виртуальной памяти, операционная система может перемещать данные между физической памятью и диском по мере необходимости.

Сегментация и пагинация: Для упрощения управления памятью операционные системы могут использовать различные методы адресации, такие как сегментация и пагинация. Сегментация разбивает адресное пространство процесса на логические блоки, а пагинация разбивает память на фиксированные блоки фиксированного размера. Это позволяет эффективно управлять памятью и обеспечивает защиту от несанкционированного доступа.

Кэширование: Для повышения производительности операционная система может использовать кэширование данных. Кэш — это небольшая, но очень быстрая память, которая хранит наиболее часто используемые данные. Когда процесс обращается к данным, операционная система сначала проверяет наличие данных в кэше, и если они там есть, то извлекает их из кэша, что гораздо быстрее, чем обращение к основной памяти.

Сборка мусора: В языках программирования со сборкой мусора операционная система автоматически освобождает память, которая больше не используется программой. Сборка мусора отслеживает объекты, на которые нет ссылок, и освобождает память, занимаемую этими объектами. Это позволяет программистам избежать ручного управления памятью и снижает вероятность утечек памяти.

Фрагментация памяти:

Фрагментация памяти возникает, когда свободное пространство в памяти разделено на несколько непрерывных фрагментов, что затрудняет выделение больших блоков памяти. Она может быть внешней (когда фрагментация происходит между выделенными блоками памяти) или внутренней (когда фрагментация происходит внутри выделенных блоков памяти). Для управления фрагментацией используются различные алгоритмы, такие как компактация и сборка мусора.

Подкачка (paging) и страничное управление:

Подкачка - это механизм, который позволяет операционной системе перемещать страницы памяти между физической памятью и хранилищем на диске (называемым файлом подкачки или своп-файлом). Это позволяет операционной системе эффективно использовать доступную физическую память при работе с большими объемами данных. Страничное управление разбивает память на фиксированные блоки (страницы) и управляет их перемещением между физической и виртуальной памятью.

Рабочий набор процесса — это набор страниц в виртуальном адресном пространстве процесса, которые в настоящее время находятся в физической памяти. Рабочий набор содержит только страничные выделения памяти. В рабочий набор не включаются ресурсы памяти, не допускающие подкачку, такие как расширения адресных окон (AWE) или большие страницы

Если процесс ссылается на страницу памяти, которая в настоящее время не входит в рабочий набор, происходит ошибка страницы. Обработчик ошибки системной страницы пытается устранить ошибку страницы, и в случае успеха страница добавляется в рабочий набор.

Ошибка жесткой страницы должна быть устранена путем чтения содержимого страницы из резервного хранилища страницы, которое является системным файлом подкачки или файлом, сопоставленным в памяти, созданным процессом. Ошибка мягкой страницы может быть устранена без доступа к резервному хранилищу.

Если несколько процессов совместно используют страницу, удаление страницы из рабочего набора одного процесса не влияет на другие процессы. После удаления страницы из рабочих наборов всех процессов, которые ее использовали, страница становится страницей перехода. Страницы перехода остаются кэшируемыми в ОЗУ, пока на страницу не будет повторно ссылаться какой-либо процесс или не будет перепрофилирована

1. Классификация запоминающих устройств. Иерархия памяти. Оперативные и постоянные запоминающие устройства.

Запоминающее устройство — носитель информации, предназначенный для записи и хранения данных. В основе работы запоминающего устройства может лежать любой физический эффект, обеспечивающий приведение системы к двум или более устойчивым состояниям.

Классификация запоминающих устройств по устойчивости записи и возможности перезаписи:

• Постоянные (ПЗУ), содержание которых не может быть изменено конечным пользователем (например, BIOS). ПЗУ в рабочем режиме допускает только считывание информации.

• Записываемые (ППЗУ), в которые конечный пользователь может записать информацию только один раз (например, CD-R).

• Многократно перезаписываемые (ПППЗУ) (например, CD-RW).

• Оперативные (ОЗУ) — обеспечивают режим записи, хранения и считывания информации в процессе её обработки. Быстрые, но дорогие ОЗУ (SRAM) строят на триггерах, более медленные, но более дешёвые разновидности ОЗУ — динамические ЗУ (DRAM) строят на элементах, состоящих из ёмкости (конденсатора) и полевого транзистора, используемого в качестве ключа разрешения записи-чтения. В обоих видах ЗУ информация исчезает после отключения от источника питания (например, тока).

По типу доступа ЗУ делятся на:

• устройства с последовательным доступом (например, магнитные ленты).

• устройства с произвольным доступом (RAM) (например, оперативная память).

• устройства с прямым доступом (например, жесткие магнитные диски).

• устройства с ассоциативным доступом (специальные устройства, для повышения производительности БД)

Классификация запоминающих устройств по геометрическому исполнению:

• дисковые (магнитные диски, оптические, магнитооптические);

• ленточные (магнитные ленты, перфоленты);

• барабанные (магнитные барабаны);

• карточные (магнитные карты, перфокарты, флэш-карты, и др.)

• печатные платы (карты DRAM).

Классификация запоминающих устройств по физическому принципу:

• перфорационные (перфокарта; перфолента);

• с магнитной записью (ферритовые сердечники, магнитные диски, магнитные ленты, магнитные карты);

• оптические (CD, DVD, HD-DVD, Blu-ray Disc);

• использующие эффекты в полупроводниках (флэш-память) и другие.

Современная система памяти образует иерархию от быстрых типов памяти маленького размера до медленных типов памяти большого размера. Мы говорим, что конкретный уровень иерархии кэширует или является кэшем для данных, расположенных на более низком уровне. Это значит, что он содержит копии данных с более низкого уровня. Когда процессор хочет получить какие-то данные, он их сперва ищет на самых быстрых высоких уровнях. И спускается на более низкие, если не может найти. На вершине иерархии находятся регистры процессора. Доступ к ним занимает 0 тактов, но их всего несколько штук. Далее идёт несколько килобайт кэш-памяти первого уровня, доступ к которой занимает примерно 4 такта. Потом идёт пара сотен килобайт более медленной кэш-памяти второго уровня. Потом несколько мегабайт кэш-памяти третьего уровня. Она гораздо медленней, но всё равно быстрее оперативной памяти. Далее расположена относительно медленная оперативная память. Оперативную память можно рассматривать как кэш для локального диска (локальные диски тоже можно разделить на отдельные категории). Локальный диск сам может рассматриваться как кэш для данных, расположенных на удалённых серверах. Резервные копии данных можно хранить на магнитных лентах

В ПЗУ находятся:

• программа управления работой процессора;

• программа запуска и останова компьютера;

• программы тестирования устройств, проверяющие при каждом включении компьютера правильность работы его блоков;

• программы управления дисплеем, клавиатурой, принтером, внешней памятью;

• информация о том, где на диске находится операционная система.

Оперативная память (также оперативное запоминающее устройство, ОЗУ) - предназначена для временного хранения данных и команд, необходимых процессору для выполнения им операций. Оперативная память передаёт процессору данные непосредственно, либо через кэш-память. Каждая ячейка оперативной памяти имеет свой индивидуальный адрес. Устанавливается в виде модулей объемом до 64 ГБ. Наиболее распространенный тип модулей — DIMM, тип памяти — DDR.

1. Виртуальная память. Структуризация адресного пространства виртуальной памяти. Задачи управления виртуальной памятью.

Однако в современной операционной системе программы используют абстракцию — виртуальное адресное пространство. Каждая программа пишется в такой модели, что она выполняется одна, всё пространство принадлежит ей, код использует адреса логической памяти, которые должны быть преобразованы в физические адреса до того, как будет выполнен доступ к памяти.

Структура адресного пространства процесса в Linux

У каждого процесса в системе Linux есть адресное пространство, состоящее из трех логических сегментов: текста, данных и стека. Текстовый сегмент (text segment) содержит машинные команды, образующие исполняемый код программы. Он создается компилятором и ассемблером при трансляции программы (написанной на языке высокого уровня, например, C или C++) в машинный код. Как правило, текстовый сегмент доступен только для чтения. Таким образом, не изменяются ни размеры, ни содержание текстового сегмента. Сегмент данных (data segment) содержит переменные, строки, массивы и другие данные программы. Он состоит из двух частей: инициализированных и неинициализированных данных. По историческим причинам вторая часть называется BSS (Block Started by Symbol). Инициализированная часть сегмента данных содержит переменные и константы компилятора, значения которых должны быть заданы при запуске программы. Все переменные в BSS должны быть инициализированы в нуль после загрузки. Операционная система Linux разрешает сегменту данных расти при выделении памяти и уменьшаться при освобождении памяти. Сегмент стека (stack segment) на большинстве компьютеров начинается около старших адресов виртуального адресного пространства и растет вниз к 0. Например, на 32-битной платформе х86 стек начинается с адреса 0xC0000000, который соответствует предельному виртуальному адресу, видимому процессам пользовательского режима.

Структура адресного пространства процесса в Windows

В Windows каждый пользовательский процесс имеет собственное виртуальное адресное пространство. Для компьютеров х86 в 32-разрядных операционных системах виртуальные адреса имеют длину 32 бита, так что каждый процесс имеет 4 Гбайт виртуального адресного пространства, по 2 Гбайта пользователю и ядру. Верхние 2 Гбайт содержат операционную систему (включая код, данные, а также резидентный и нерезидентный пулы). В 64-разрядных версиях Windows нижние 128ТБ адресного пространства выделяются для пользовательского режима, старшие 128ТБ - для режима ядра.

Задачи управления виртуальной памятью: задача размещения, задача перемещения, задача преобразования адресов, задача замещения

Функциями ОС по управлению памятью в мультипрограммной системе являются:

• отслеживание свободной и занятой памяти;

• выделение памяти процессам и освобождение памяти по завершении процессов;

• вытеснение кодов и данных процессов из оперативной памяти на диск (полное или частичное), когда размеры основной памяти не достаточны для размещения в ней всех процессов, и возвращение их в оперативную память, когда в ней освобождается место;

• настройка адресов программы на конкретную область физической памяти.

Виртуальная память решает следующие задачи:

• размещает данные в запоминающих устройствах разного типа, например, часть программы в оперативной памяти, а часть на диске;

• перемещает по мере необходимости данные между запоминающими устройствами разного типа, например, подгружает нужную часть программы с диска в оперативную память;

• преобразует виртуальные адреса в физические. Все эти действия выполняются автоматически, без участия программиста, то есть механизм виртуальной памяти является прозрачным по отношению к пользователю

1. Подкачка. Алгоритмы замещения страниц. Куча (heap). Стек.

Файл подкачки или виртуальная память — это способ системы виртуальной памяти увеличить оперативную память, когда ее не хватает для совершения операций. Система автоматически задействует файл подкачки, когда приложениям не хватит системной памяти ОЗУ. Хотя система сама регулирует объем файла подкачки иногда может понадобиться вручную увеличить виртуальную память.

Подкачка реализована частично ядром, а частично новым процессом, называемым демоном страниц (page daemon). Демон страниц — это процесс 2 (процесс 0 — это процесс idle, традиционно называемый своппером, а процесс 1 — это init). Как и все демоны, демон страниц работает периодически. После пробуждения он осматривается, есть ли для него работа. Если он видит, что количество страниц в списке свободных слишком мало, то он начинает освобождать страницы.

Операционная система Linux является системой с подкачкой страниц по требованию (без упреждающей подкачки) и без концепции рабочего набора (хотя в ней есть системный вызов для указания пользователем страницы, которая ему может скоро понадобиться). Текстовые сегменты и отображаемые на адресное пространство памяти файлы подгружаются из соответствующих им файлов на диске. Все остальное выгружается либо в раздел подкачки (если он присутствует), либо в один из файлов подкачки (фиксированной длины), которые называются областью подкачки (swap area). Файлы подкачки могут динамически добавляться и удаляться, и у каждого есть свой приоритет.

Windows использует синхронную (just-in-time) стратегию. Зафиксированным страницам (поддерживаемым файлом подкачки) не выделяется место в файле подкачки до того момента, когда их необходимо вытеснить в файл подкачки. Для тех страниц, которые никогда не вытесняются, дисковое пространство не выделяется. Если суммарная виртуальная память меньше, чем имеющаяся физическая память, то файл подкачки не нужен совсем. Когда хранящиеся в файле подкачки страницы считываются в память, они сохраняют свое место в файле подкачки (до первой модификации). Если страница не модифицируется, она попадает в специальный список свободных физических страниц, называемый списком резервирования (standby list), из которого она может быть взята для повторного использования (без необходимости записывать ее обратно на диск). Если же она модифицируется, то диспетчер памяти освобождает страницу в файле подкачки и теперь единственный экземпляр страницы находится в памяти. Диспетчер памяти делает это путем маркировки страницы «только для чтения» (после ее загрузки). В первый раз, когда поток пытается записать в страницу, диспетчер памяти обнаружит эту ситуацию и освободит страницу в файле подкачки, обеспечит доступ на запись к странице, а затем даст потоку возможность повторить попытку. Windows поддерживает до 16 файлов подкачки

При возникновении ошибки отсутствия страницы операционная система должна выбрать выселяемую (удаляемую из памяти) страницу, чтобы освободить место для загружаемой страницы. Если предназначенная для удаления страница за время своего нахождения в памяти претерпела изменения, она должна быть переписана на диске, чтобы привести дисковую копию в актуальное состояние. Но если страница не изменялась (например, она содержала текст программы), дисковая копия не утратила своей актуальности и перезапись не требуется. Тогда считываемая страница просто пишется поверх выселяемой

Оптимальный алгоритм замещения страниц

Наилучший алгоритм замещения страниц несложно описать, но совершенно невозможно реализовать. В нем все происходит следующим образом. На момент возникновения ошибки отсутствия страницы в памяти находится определенный набор страниц. К некоторым из этих страниц будет осуществляться обращение буквально из следующих команд (эти команды содержатся на странице). К другим страницам обращения может не быть и через 10, 100 или, возможно, даже 1000 команд. Каждая страница может быть помечена количеством команд, которые должны быть выполнены до первого обращения к странице. Оптимальный алгоритм замещения страниц гласит, что должна быть удалена страница, имеющая пометку с наибольшим значением. Если какая-то страница не будет использоваться на протяжении 8 млн команд, а другая какая-нибудь страница не будет использоваться на протяжении 6 млн команд, то удаление первой из них приведет к ошибке отсутствия страницы, в результате которой она будет снова выбрана с диска в самом отдаленном будущем.

Использование битов состояния в алгоритмах исключения страниц

Чтобы позволить операционной системе осуществить сбор полезной статистики востребованности страниц, большинство компьютеров, использующих виртуальную память, имеют два бита состояния, R и M, связанных с каждой страницей. Бит R устанавливается при каждом обращении к странице (при чтении или записи). Бит M устанавливается, когда в страницу ведется запись (то есть когда она модифицируется). Эти биты присутствуют в каждой записи таблицы страниц. Важно усвоить, что эти биты должны обновляться при каждом обращении к памяти, поэтому необходимо, чтобы их значения устанавливались аппаратной частью. После установки бита в 1 он сохраняет это значение до тех пор, пока не будет сброшен операционной системой. Биты R и M могут использоваться для создания следующего простого алгоритма замещения страниц. При запуске процесса оба страничных бита для всех его страниц устанавливаются операционной системой в 0. Время от времени (например, при каждом прерывании по таймеру) бит R сбрасывается, чтобы отличить те страницы, к которым в последнее время не было обращений, от тех, к которым такие обращения были. При возникновении ошибки отсутствия страницы операционная система просматривает все страницы и на основе текущих значений принадлежащих им битов R и M делит их на четыре категории

Алгоритм исключения давно использовавшейся страницы

Алгоритм исключения давно использовавшейся страницы (Not Recently Used (NRU)) удаляет произвольную страницу, относящуюся к самому низкому непустому классу. В этот алгоритм заложена идея, суть которой в том, что лучше удалить модифицированную страницу, к которой не было обращений по крайней мере за последний такт системных часов (обычно это время составляет около 20 мс), чем удалить интенсивно используемую страницу

Алгоритм «первой пришла, первой и ушла» Другим низкозатратным алгоритмом замещения страниц является алгоритм FIFO (First In, First Out — «первым пришел, первым ушел»). Операционная система ведет список всех страниц, находящихся на данный момент в памяти, причем совсем недавно поступившие находятся в хвосте, поступившие раньше всех — в голове списка. При возникновении ошибки отсутствия страницы удаляется страница, находящаяся в голове списка, а к его хвосту добавляется новая страница

Алгоритм «второй шанс» Простой модификацией алгоритма FIFO, исключающей проблему удаления часто запрашиваемой страницы, может стать проверка бита R самой старой страницы. Если его значение равно нулю, значит, страница не только старая, но и невостребованная, поэтому она тут же удаляется. Если бит R имеет значение 1, он сбрасывается, а страница помещается в конец списка страниц и время ее загрузки обновляется, как будто она только что поступила в память. Затем поиск продолжается.

Алгоритм «часы» При всей своей логичности алгоритм «второй шанс» слишком неэффективен, поскольку он постоянно перемещает страницы в своем списке. Лучше содержать все страничные блоки в циклическом списке в виде часов. Стрелка указывает на самую старую страницу. При возникновении ошибки отсутствия страницы проверяется та страница, на которую указывает стрелка. Если ее бит R имеет значение 0, страница выселяется, на ее место в «циферблате» вставляется новая страница и стрелка передвигается вперед на одну позицию. Если значение бита R равно 1, то он сбрасывается и стрелка перемещается на следующую страницу. Этот процесс повторяется до тех пор, пока не будет найдена страница с R = 0.

Алгоритм замещения наименее востребованной страницы В основе неплохого приближения к оптимальному алгоритму лежит наблюдение, что страницы, интенсивно используемые несколькими последними командами, будут, скорее всего, снова востребованы следующими несколькими командами. И наоборот, долгое время не востребованные страницы наверняка еще долго так и останутся невостребованными. Эта мысль наталкивает на вполне реализуемый алгоритм: при возникновении ошибки отсутствия страницы нужно избавиться от той страницы, которая длительное время не была востребована. Эта стратегия называется замещением наименее востребованной страницы

Алгоритм нечастого востребования Одно из возможных решений называется алгоритмом нечастого востребования (Not Frequently Used (NFU)). Для его реализации потребуется программный счетчик с начальным нулевым значением, связанный с каждой страницей. При каждом прерывании от таймера операционная система сканирует все находящиеся в памяти страницы. Для каждой страницы к счетчику добавляется значение бита R, равное 0 или 1. Счетчики позволяют приблизительно отследить частоту обращений к каждой странице. При возникновении ошибки отсутствия страницы для замещения выбирается та страница, чей счетчик имеет наименьшее значение.

Алгоритм старения К счастью, небольшая модификация алгоритма NFU позволяет довольно близко подойти к имитации алгоритма LRU. Модификация состоит из двух частей. Во-первых, перед добавлением к счетчикам значения бита R их значение сдвигается на один разряд вправо. Во-вторых, значение бита R добавляется к самому левому, а не к самому правому биту. При возникновении ошибки отсутствия страницы удаляется та страница, чей счетчик имеет наименьшее значение

Алгоритм «рабочий набор» При использовании замещения страниц в простейшей форме процессы начинают свою работу, не имея в памяти вообще никаких страниц. Как только центральный процессор попытается извлечь первую команду, он получает ошибку отсутствия страницы, заставляющую операционную систему ввести в память страницу, содержащую первую команду. Обычно вскоре за этим следуют ошибки отсутствия страниц с глобальными переменными и стеком. Через некоторое время процесс располагает большинством необходимых ему страниц и приступает к работе, сталкиваясь с ошибками отсутствия страниц относительно редко. Эта стратегия называется замещением страниц по требованию (demand paging), поскольку страницы загружаются только по мере надобности, а не заранее.

Алгоритм WSClock Базовый алгоритм рабочего набора слишком трудоемок, поскольку при возникновении ошибки отсутствия страницы для определения местонахождения подходящего кандидата на удаление необходимо просканировать всю таблицу страниц. Усовершенствованный алгоритм, основанный на алгоритме «часы», но также использующий информацию о рабочем наборе, называется WSClock. Благодаря простоте реализации и хорошей производительности он довольно широко используется на практике. Необходимая структура данных сводится к циклическому списку страничных блоков, как в алгоритме «часы» и как показано на рис. 3.2.21, а. Изначально этот список пуст. При загрузке первой страницы она добавляется к списку. По мере загрузки следующих страниц они попадают в список, формируя замкнутое кольцо. В каждой записи содержится поле времени последнего использования из базового алгоритма рабочего набора, а также бит R (показанный на рисунке) и бит M (не показанный на рисунке). Как и в алгоритме «часы», при каждой ошибке отсутствия страницы сначала проверяется страница, на которую указывает стрелка. Если бит R установлен в 1, значит, страница была использована в течение текущего такта, поэтому она не является идеальным кандидатом на удаление. Затем бит R устанавливается в 0, стрелка перемещается на следующую страницу, и алгоритм повторяется уже для нее.

Куча (heap) — это сегмент памяти, для которого не устанавливается постоянный размер перед компиляцией и который может динамически управляться программистом. То есть, это «свободный пул» памяти, который можно использовать при запуске приложения. Размер кучи приложения определяется физическими ограничениями оперативной памяти (оперативной памяти) и обычно намного больше размера стека. Память из кучи выделяется с помощью функции malloc(). malloc возвращает указатель void, который затем нужно привести к правильному типу. После использования блока памяти его нужно освободить помощью функции free().

Каждый процесс имеет кучу по умолчанию, предоставляемую системой. Приложения, которые часто выделяют ресурсы из кучи, могут повысить производительность с помощью частных куч. Частная куча — это блок одной или нескольких страниц в адресном пространстве вызывающего процесса. После создания частной кучи процесс использует такие функции, как HeapAlloc и HeapFree, для управления памятью в этой куче. Функции кучи также можно использовать для управления памятью в куче процесса по умолчанию с помощью дескриптора, возвращаемого функцией GetProcessHeap. Для этой цели новые приложения должны использовать функции кучи вместо глобальных и локальных функций. Нет различий между памятью, выделенной из частной кучи, и памятью, выделенной с помощью других функций выделения памяти. Функция HeapCreate создает частный объект кучи, из которого вызывающий процесс может выделять блоки памяти с помощью функции HeapAlloc. После фиксации страницы не удаляются до завершения процесса или до тех пор, пока куча не будет уничтожена путем вызова функции HeapDestroy. Функция HeapAlloc выделяет указанное количество байтов из частной кучи и возвращает указатель на выделенный блок. Этот указатель можно использовать в функциях HeapFree, HeapReAlloc, HeapSize и HeapValidate.

Стек — это сегмент памяти, в котором данные, такие как локальные переменные и вызовы функций, добавляются и/или удаляются по принципу «последним пришел — первым вышел» (LIFO). Вообще говоря, стек — это структура данных, которая хранит значения данных в памяти последовательно. Однако, в отличие от массива, вы получаете доступ (чтение или запись) к данным только на «верхней части» стека. Чтение из стека называется «извлечение» (pop), а запись в стек — «вталкивание» (push).

1. Типы устройств ввода/вывода. Обработка внешних прерываний. Синхронный и асинхронный ввод/вывод.

Устройства ввода-вывода можно условно разделить на две категории: блочные устройства и символьные устройства. К блочным относятся такие устройства, которые хранят информацию в блоках фиксированной длины, у каждого из которых есть собственный адрес. Обычно размеры блоков варьируются от 512 до 65 536 байт. Вся передача данных ведется пакетами из одного или нескольких целых (последовательных) блоков. Важным свойством блочного устройства является то, что оно способно читать или записывать каждый блок независимо от всех других блоков. Среди наиболее распространенных блочных устройств жесткие диски, приводы оптических дисков и флеш-накопители USB. Другой тип устройств ввода-вывода — символьные устройства. Они выдают или воспринимают поток символов, не относящийся ни к какой блочной структуре. Они не являются адресуемыми и не имеют никакой операции позиционирования. В качестве символьных устройств могут рассматриваться терминалы, принтеры, сетевые интерфейсы, мыши (в качестве устройства-указателя) и множество других устройств, не похожих на дисковые устройства. Некоторые устройства не подпадают под эту классификацию, например, часы и сенсорные экраны. Тем не менее модель блочных и символьных устройств является достаточно общей для того, чтобы использовать ее в качестве основы для придания части программного обеспечения операционной системы независимости от устройства ввода-вывода.

На аппаратном уровне прерывания работают следующим образом [2]. Когда устройство ввода-вывода заканчивает свою работу, оно инициирует прерывание (при условии, что прерывания разрешены операционной системой). Для этого устройство выставляет сигнал на выделенную устройству специальную линию шины. Этот сигнал распознается микросхемой контроллера прерываний, расположенной на материнской плате. Контроллер прерываний принимает решение о дальнейших действиях

При отсутствии других необработанных запросов прерывания контроллер прерываний обрабатывает прерывание немедленно. Если прерывание уже обрабатывается, и в это время приходит запрос от другого устройства по линии с более низким приоритетом, то новый запрос просто игнорируется. В этом случае устройство продолжает удерживать сигнал прерывания на шине до тех пор, пока оно не будет обслужено центральным процессором. Для обработки прерывания контроллер выставляет на адресную шину номер устройства, требующего к себе внимания, и устанавливает сигнал прерывания на соответствующий контакт процессора. Этот сигнал заставляет процессор приостановить текущую работу и начать выполнять обработку прерывания. Номер, выставленный на адресную шину, используется в качестве индекса в таблице, называемой вектором прерываний, из которой извлекается новое значение счетчика команд. Новый счетчик команд указывает на начало соответствующей процедуры обработки прерывания. Обыч­но с этого места аппаратные и эмулированные прерывания используют один и тот же механизм и часто пользуются одним и тем же вектором. Расположение вектора может быть либо жестко прошито на аппаратном уровне, либо, наоборот, располагаться в произвольном месте памяти, на которое указывает специальный регистр процессора, загружаемый операционной системой. Вскоре после начала своей работы процедура обработки прерываний подтверждает получение прерывания, записывая определенное значение в порт контроллера прерываний. Это подтверждение разрешает контроллеру издавать новые прерывания.

Аппаратура всегда, прежде чем начать процедуру обработки прерывания, со­храняет определенную информацию. Сохраняемая информация и место ее хра­нения широко варьируются в зависимости от центрального процессора. Как минимум сохраняется счетчик команд, что позволяет продолжить выполнение прерванного процесса. Другая крайность представляет собой сохранение всех программно доступных регистров и большого количества внутренних регистров центрального процессора.

Асинхронный ввод-вывод используется там, где можно оптимизировать производительность приложения. При асинхронном вводе-выводе приложение инициирует операцию ввода-вывода, а затем может продолжить свою работу (во время выполнения этого запроса). При синхронном вводе-выводе приложение блокируется до завершения выполнения операции ввода-вывода. С точки зрения вызывающего потока асинхронный ввод-вывод более эффективен, поскольку позволяет продолжать выполнение, в то время как операция ввода-вывода ставится диспетчером ввода-вывода в очередь и впоследствии выполняется. Однако приложение, использующее асинхронный ввод-вывод, требует механизма определения завершенности этой операции.

1. Файловые системы. Файлы и директории. Управление внешней памятью.

Файловая система — порядок, определяющий способ организации, хранения и именования данных на носителях информации в компьютерах, а также в другом электронном оборудовании: цифровых фотоаппаратах, мобильных телефонах и т. п. Файловая система определяет формат содержимого и способ физического хранения информации, которую принято группировать в виде файлов. Конкретная файловая система определяет размер имен файлов (и каталогов), максимальный возможный размер файла и раздела, набор атрибутов файла. Некоторые файловые системы предоставляют сервисные возможности, например, разграничение доступа или шифрование файлов.

Список файловых систем, которые поддерживаются ядром, находится в файле /proc/filesystems. Виды файловых систем, предлагаемых при установке ОС на базе Linux: Ext, Ext2(Не поддерживает ведение журнала Подходит для SD-карт и USB-накопителей, поскольку имеет высокую производительность и меньшее количество операций записи), Ext3(Поддерживает ведение журнала Ведение журнала отслеживает изменения файлов, что помогает быстрому восстановлению и снижает вероятность потери данных в случае сбоя системы.), Ext4(Поддерживает ведение журнала Введено много новых функций. Экстенты, обратная совместимость, постоянное предварительное выделение, отложенное выделение, неограниченное количество подкаталогов, контрольная сумма журнала, более быстрая проверка FS, прозрачное шифрование), JFS, XFS(высоко производительная 64- битная, журналируемая файловая система.), Btrfs(Btrfs - это новая файловая система, разработанная с нуля. Расшифровывается как B-Tree Filesystem. Btrfs поддерживает управление несколькими томами на одном разделе, контрольные суммы для блоков, асинхронную репликацию, прозрачное сжатие, а также многие другие возможности современных файловых систем), Swap.

Типы файловых систем Windows включают: таблица размещения файлов (FAT), FAT32 и расширенная таблица размещения файлов (exFAT); файловая система NT (NTFS); устойчивая файловая система (ReFS).

Файл - поименованная совокупность данных, обычно размещаемая на внешних запоминающих устройствах.

Файл в системе Linux — это последовательность байтов произвольной длины (от 0 до некоторого максимума), содержащая произвольную информацию. Не делается различия между текстовыми (ASCII) файлами, двоичными файлами и любыми другими типами файлов. Значение битов в файле целиком определяется владельцем файла. Системе это безразлично. Имена файлов ограничены 255 символами. В именах файлов разрешается использовать все ASCII-символы, кроме символа NUL.

Для удобства файлы могут группироваться в каталоги. Каталоги хранятся на диске в виде файлов, и с ними можно работать практически так же, как с файлами. Каталоги могут содержать подкаталоги, что приводит к иерархической файловой системе. Корневой каталог называется / и всегда содержит несколько подкаталогов. Символ / используется также для разделения имен каталогов, поэтому имя /usr/ast/x обозначает файл x,

При создании каталога в нем автоматически создаются две записи, «.» и «..». Первая запись обозначает сам каталог. Вторая является ссылкой на родительский каталог, то есть каталог, в котором данный каталог числится как запис

Домашний каталог (домашняя папка, домашняя директория) – предназначен для хранения собственных данных пользователя Linux и личных настроек для программ. Как правило, становится текущим непосредственно после регистрации пользователя в системе.

Каждый сервер NFS экспортирует один или несколько своих каталогов, предоставляя доступ к ним удаленным клиентам. Как правило, доступ к каталогу предоставляется вместе со всеми его подкаталогами, так что, фактически все дерево каталогов экспортируется как единое целое. Список экспортируемых сервером каталогов хранится в файле (обычно это файл /etc/exports), чтобы эти каталоги экспортировались автоматически при загрузке сервера. Клиенты получают доступ к экспортируемым каталогам, монтируя эти каталоги. Если клиент монтирует (удаленный) каталог, то этот каталог становится частью иерархии каталогов клиента

Файловой системой NFS поддерживается большинство системных вызовов операционной системы Linux, за исключением (как ни странно) системных вызовов open и close. Исключение системных вызовов open и close не случайно. Это сделано преднамеренно. Нет необходимости открывать файл, прежде чем прочитать его. Также не нужно закрывать файл после того, как данные из него прочитаны. Вместо этого, чтобы прочитать файл, клиент посылает на сервер сообщение lookup (содержащее имя файла) с запросом найти этот файл и вернуть описатель файла, представляющий собой структуру, идентифицирующую файл (то есть содержащую идентификатор файловой системы и номер i-узла вместе с прочей информацией). В отличие от системного вызова open, операция lookup не копирует никакой информации во внутренние системные таблицы. Системный вызов read содержит описатель файла (который предстоит прочитать), смещение в файле (с которого надо начинать чтение), а также требуемое количество байтов. Таким образом, каждое сообщение является самодостаточным.

1. Принципы организации и структура ОС Windows. Обзор версий Windows. Методы инсталляции ОС Windows.

Архитектура операционной системы Windows включает ядро операционной системы, системные службы и приложения

Ядро операционной системы

На самом низком уровне операционной системы ядро операционной системы состоит из самого ядра Windows и драйверов устройств низкого уровня. Ядро отвечает за прием запросов операционной системы от системных служб. Затем ядро преобразует эти запросы в инструкции для аппаратного обеспечения компьютера, включая центральный процессор (ЦП), память и аппаратные устройства. При запуске операционной системы сначала инициализируются ядро и связанные с ним низкоуровневые драйверы устройств, а затем службы операционной системы.

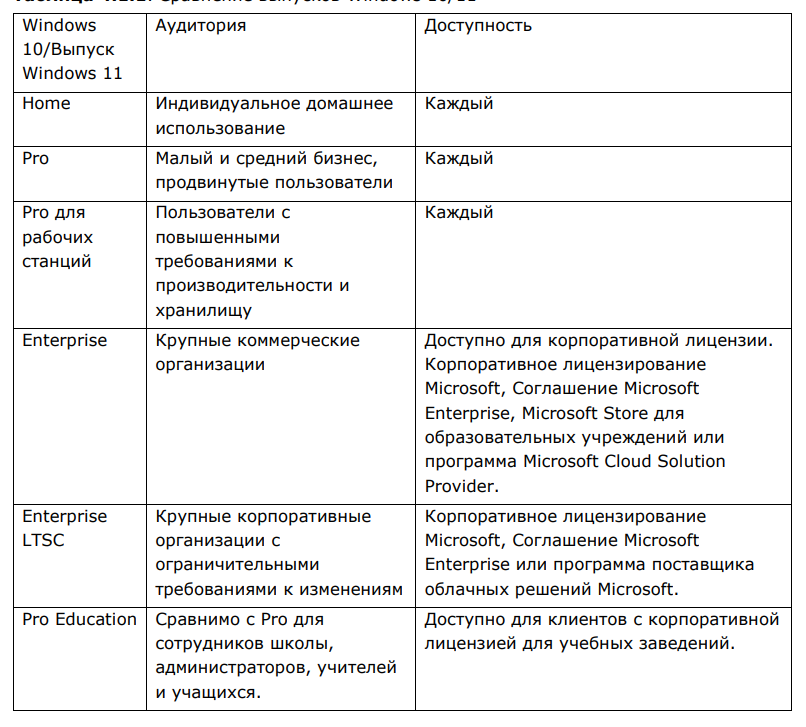
Системные службы

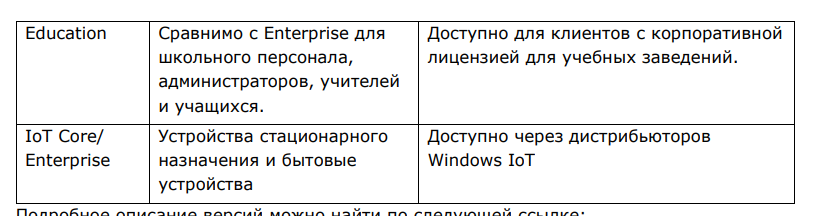
Службы операционной системы являются частью операционной системы, а не компонентами, которые вы устанавливаете после развертывания операционной системы. Кроме того, службы операционной системы функционируют без каких-либо действий со стороны пользователя. Фактически они начинаются до того, как пользователь войдет в систему. И службы операционной системы, и драйверы устройств являются программным обеспечением. Однако разница между ними заключается в том, что драйверы устройств напрямую взаимодействуют с аппаратными устройствами или компонентами.

Понимание приложений

На верхнем уровне операционной системы приложения работают путем взаимодействия с пользователем компьютера, а на нижнем уровне — путем интеграции со службами операционной системы. Вы устанавливаете приложения после установки операционной системы, и для их использования вам необходимо запускать приложения вручную.

Клиентские версии Windows:





Актуальные серверные версии:

Windows server 2019

Windows server 2022 – редакции Standard, Datacenter, AzureEdition

Azure Stack HCI

Установка:

Ручная установка с использованием установочного носителя

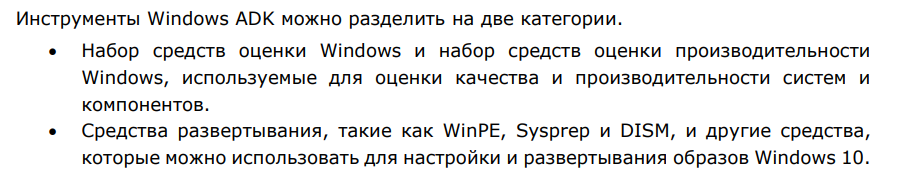
Автономная (необслуживаемая) установка (конфигурационые файлы)

Cопровождаемая производителе

Создание копий или образов систем (WIM)

Использование файлов ответов

Набор инструментальных средств Windows Assessment and Deployment Kit



Служба Windows Deployment Services

Загрузочные образы

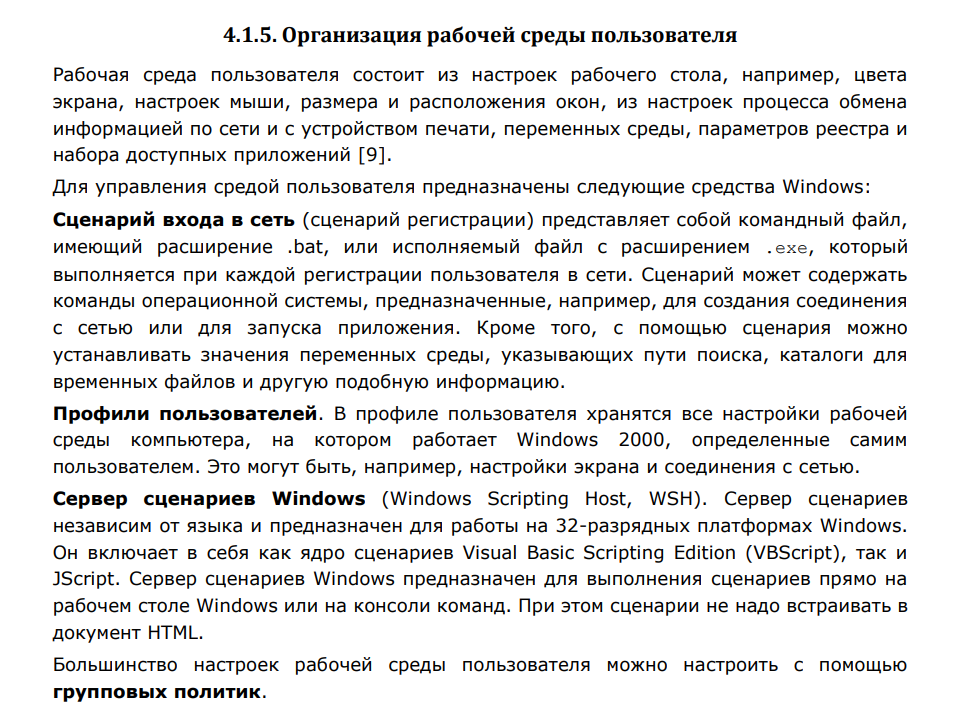
Установочные образы

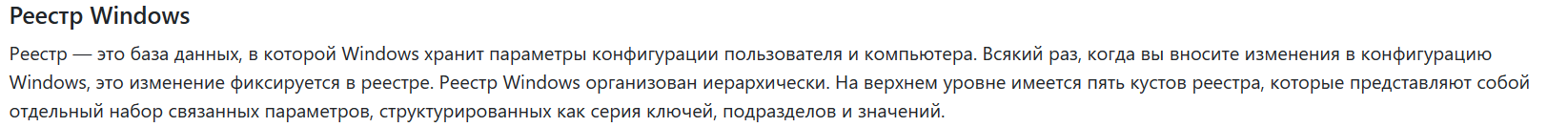
Образы обнаружения

Образы захвата

Набор инструментальных средств Microsoft Deployment Toolkit

1. ОС Windows: организация рабочей среды пользователя, работа с учетными записями пользователей и групп, работа с профилями пользователей.

****



Локальные учетные записи пользователей — это субъекты безопасности, которые используются для защиты и управления доступом к ресурсам на устройстве для служб или пользователей.

Учетные записи локальных пользователей по умолчанию — это встроенные учетные записи, создаются автоматически при установке ОС. не могут быть удалены и не предоставляют доступ к сетевым ресурсам

Учётные записи:

**Админинстратор** – создаётся первой, имеет полный контроль над файлами, каталогами, службами и другими ресурсами на локальном устройстве. может создавать других локальных пользователей, назначать права пользователей и назначать разрешения. может в любое время управлять

локальными ресурсами, изменив права и разрешения пользователя. Нельзя удалить, но можно переименовать или отключить.

**Гость** – временный/разовый вход без учётной записи. По дефолту отключена.

Управление учётными записями – папка «Пользователи», NET.EXE LOCALGROUP, NET.EXE USER.

Контроль учетных записей (UAC) — это функция безопасности, которая информирует вас о том, что программа вносит изменения, требующие административных разрешений. UAC работает путем настройки уровня разрешений учетной записи пользователя. По умолчанию контроль учетных записей настраивается для уведомления при попытке приложений внести изменения в компьютер, но вы можете изменить это, когда управление учетными записями уведомляет вас

Учетные записи для служб

**SYSTEM –** используется ОС и службами Windows + По умолчанию учетной записи SYSTEM предоставляются разрешения на полный доступ ко всем файлам в томе NTFS.

**NETWORK SERVICE** - предопределенная локальная учетная запись, используемая диспетчером управления службами (SCM). Служба, работающая в контексте учетной записи NETWORK SERVICE, предоставляет учетные данные компьютера удаленным серверам.

**LOCAL SERVICE** — это предопределенная локальная учетная запись, используемая диспетчером управления службами. Она имеет минимальные привилегии на локальном компьютере и предоставляет анонимные учетные данные в сети.

Работа с профилями пользователей - Система создает профиль пользователя при первом входе пользователя на компьютер. При последующих входах система загружает профиль пользователя, а затем другие системные компоненты настраивают среду пользователя в соответствии с информацией в профиле

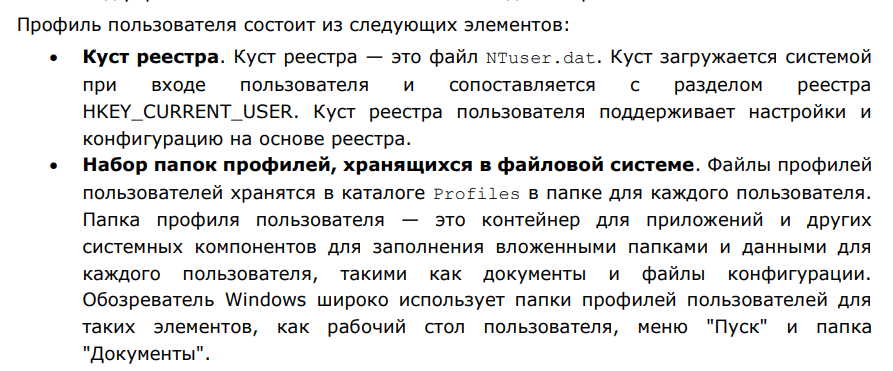
Типы:

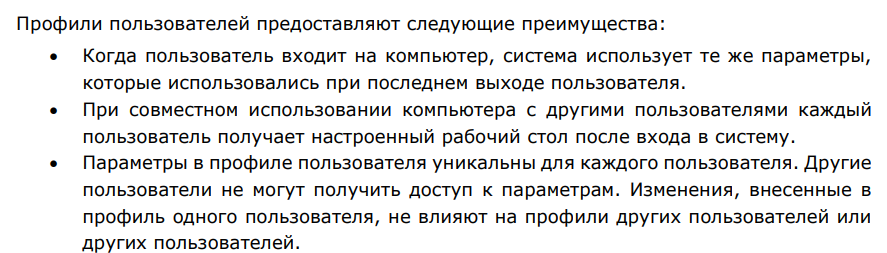
**Локальные профили пользователей** - создается при первом входе пользователя на компьютер. хранится на локальном жестком диске. Изменения, внесенные в профиль относятся к пользователю и компьютеру, на котором внесены изменения.

**Перемещаемые профили пользователей** - копия локального профиля, которая копируется в общую папку сервера и хранится в ней. Этот профиль загружается на любой компьютер, на который пользователь входит в сеть. Изменения, внесенные в перемещаемый профиль пользователя, синхронизируются с серверной копией профиля при выходе пользователя из системы. Преимущество перемещаемых профилей пользователей заключается в том, что пользователям не нужно создавать профиль на каждом компьютере, который они используют в сети

**Обязательные профили пользователей** - тип профиля, который администраторы могут использовать для указания параметров для пользователей. Только системные администраторы могут вносить изменения в обязательные профили пользователей. Изменения, внесенные пользователями в параметры рабочего стола, теряются при выходе пользователя из системы

**Временные профили пользователей** - Временный профиль выдается каждый раз, когда ошибка препятствует загрузке профиля пользователя. Временные профили удаляются в конце каждого сеанса, а изменения, внесенные пользователем в параметры рабочего стола и файлы, теряются при выходе пользователя из системы. Временные профили доступны только на компьютерах под управлением Windows 2000 и более поздних версий.





1. Системный реестр ОС Windows, его назначение и использование.

Реестр — это база данных, в которой Windows хранит параметры конфигурации пользователя и компьютера. Всякий раз, когда вы вносите изменения в конфигурацию Windows, это изменение фиксируется в реестре. Реестр Windows организован иерархически. На верхнем уровне имеется пять кустов реестра, которые представляют собой отдельный набор связанных параметров, структурированных как серия ключей, подразделов и значений: HKEY\_CLASSES\_ROOT, HKEY\_CURRENT\_USER, HKEY\_LOCAL\_MACHINE, HKEY\_USERS и HKEY\_CURRENT\_CONFIG.

HKEY\_CURRENT\_USER

Этот куст содержит информацию о конфигурации для текущего пользователя, вошедшего в систему. Такие элементы, как цветовая схема Windows пользователя и настройки шрифта, хранятся в соответствующих значениях под этим кустом. При ссылке на этот куст при редактировании реестра этот куст иногда называют HKCU. Этот куст представляет собой ярлык ключа, хранящегося в HKEY\_USERS. HKEY\_LOCAL\_MACHINE

Вероятно, это самый важный куст, в который вы, вероятно, будете вносить больше всего изменений. Этот куст, иногда называемый сокращенно HKLM, хранит все параметры конфигурации, связанные с компьютером.

HKEY\_USERS

Этот куст содержит коллекцию всей информации о конфигурации для всех пользователей, выполнивших вход локально на компьютер, включая текущего пользователя, вошедшего в систему. Фактически, один из ключей под этим кустом — это ключ текущего пользователя, вошедшего в систему, который отображается как куст

HKEY\_CURRENT\_USER.

Важно знать, что вы, скорее всего, будете вносить прямые изменения в настройки пользователя только для текущего пользователя, вошедшего в систему. HKEY\_CURRENT\_CONFIG

Этот куст содержит информацию о текущем профиле оборудования, который локальный компьютер использовал во время запуска системы. Обычно вы не вносите изменения в этот куст.

Запуск реестра – regedit. Также параметры конфигурации можно менять с помощью панели управления

Для поддержания структуры базы данных аналогичные настройки хранятся в папках и подпапках, известных как ключи и подразделы. Это упрощает ссылку на определенное значение реестра. Вы можете указать путь, объявив соответствующий куст, ключ, подразделы и значение

Значения определяют поведение операционной системы и хранятся в разделах и подразделах. Существует много типов значений, в зависимости от типа данных, которые каждое из них хранит. Например, вы можете захотеть хранить текстовые значения, числовые данные, переменные и подобные данные. Ниже перечислены наиболее распространенные типы значений реестра.

REG\_BINARY. Необработанные двоичные данные. Эти значения обычно отображаются в шестнадцатеричном формате. Информация об оборудовании часто хранится в значениях REG\_BINARY.

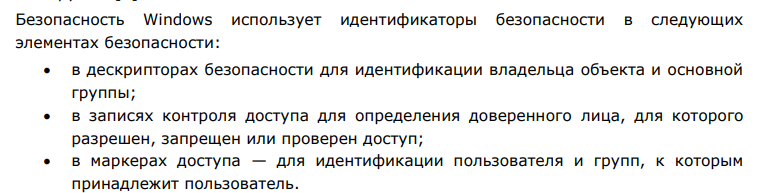
REG\_DWORD. 4-байтовые числа (32-битное целое число). Многие значения, связанные с драйверами устройств и службами, хранятся в значениях REG\_DWORD. Например, значения START и TYPE для драйверов устройств всегда определяются в значениях типа REG\_DWORD.

REG\_SZ. Текстовая строка фиксированной длины.

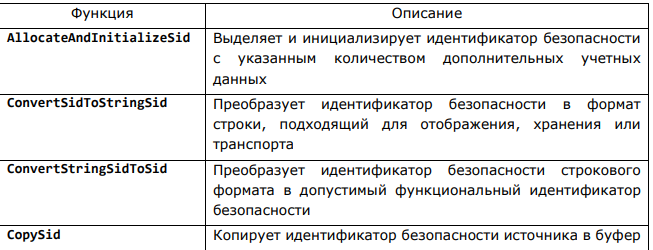
REG\_EXPAND\_SZ. Текстовая строка переменной длины. REG\_MULTI\_SZ. Несколько строковых значений.

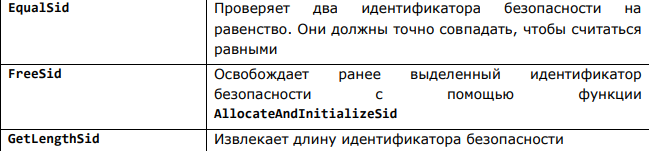
1. ОС Windows: планирование и назначение разрешений NTFS, списки управления доступом.

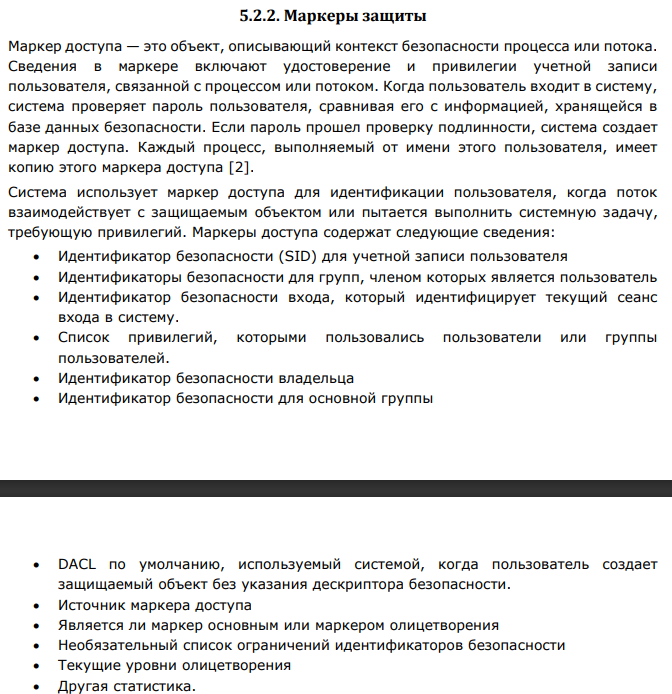
Идентификатор безопасности (SID) — это уникальное значение переменной длины, используемое для идентификации доверенного лица. Каждая учетная запись имеет уникальный идентификатор безопасности, выданный центром сертификации, таким как контроллер домена Windows, и хранящийся в базе данных безопасности. Каждый раз, когда пользователь входит в систему, система получает идентификатор безопасности для этого пользователя из базы данных и помещает его в маркер доступа для этого пользователя. Система использует SID в маркере доступа для идентификации пользователя во всех последующих взаимодействиях с безопасностью Windows

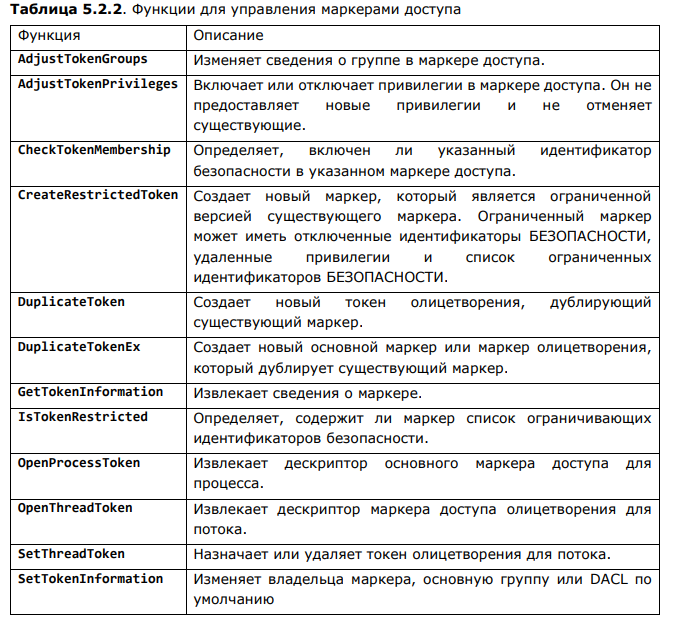


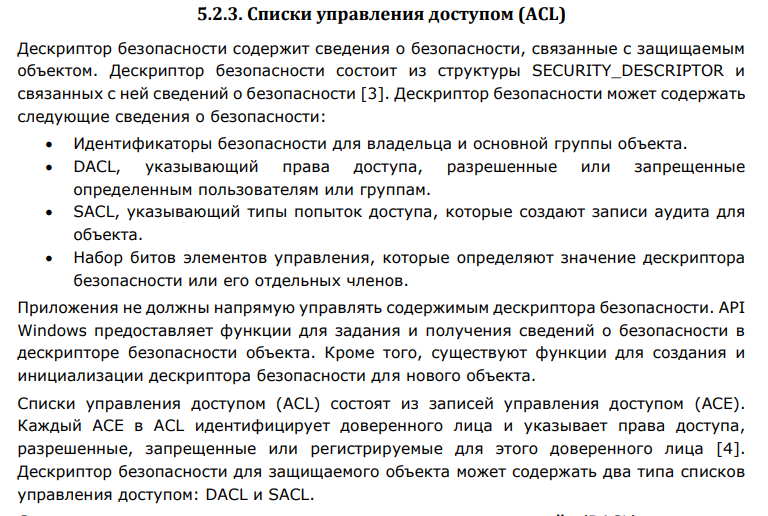
Функции Windows API для работы с идентификаторами безопасности





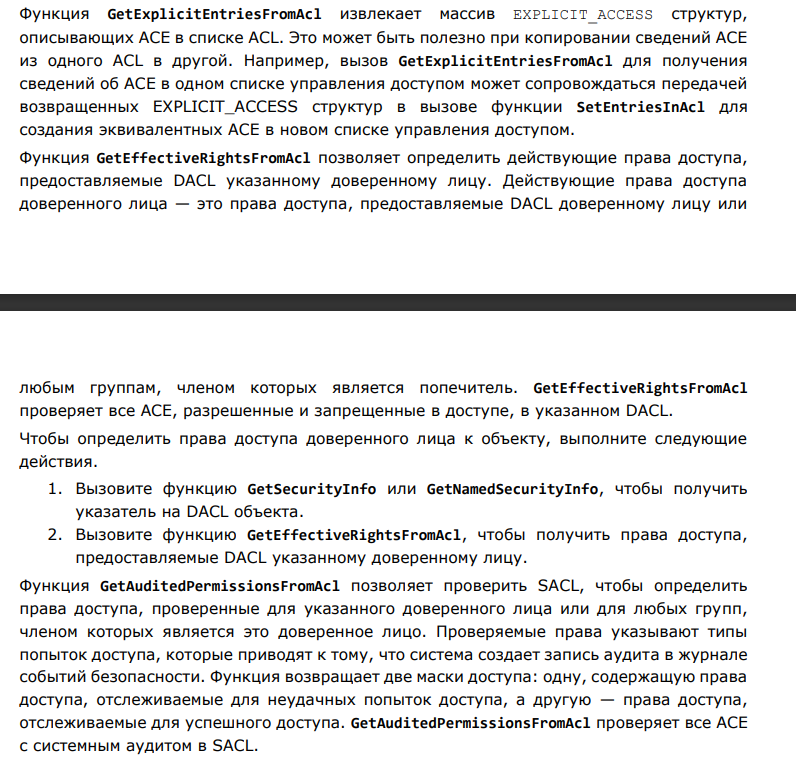




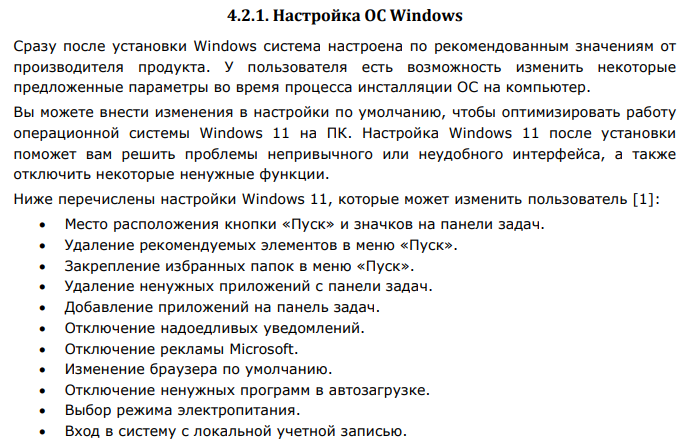


Список управления доступом на уровне пользователей (DACL) определяет доверенных лиц, которым разрешен или запрещен доступ к защищаемому объекту

Системный список управления доступом (SACL) позволяет администраторам регистрировать попытки доступа к защищенному объекту

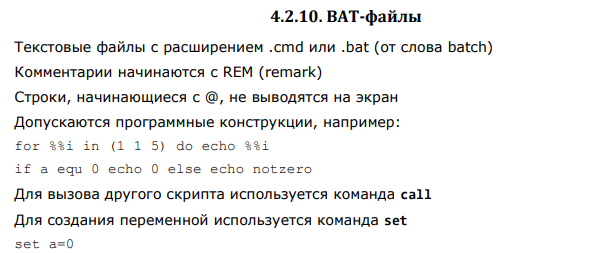


1. Средства автоматической настройки в ОС Windows. BAT-файлы. Powershell. WMI. ETW. Групповые политики. Административные шаблоны.

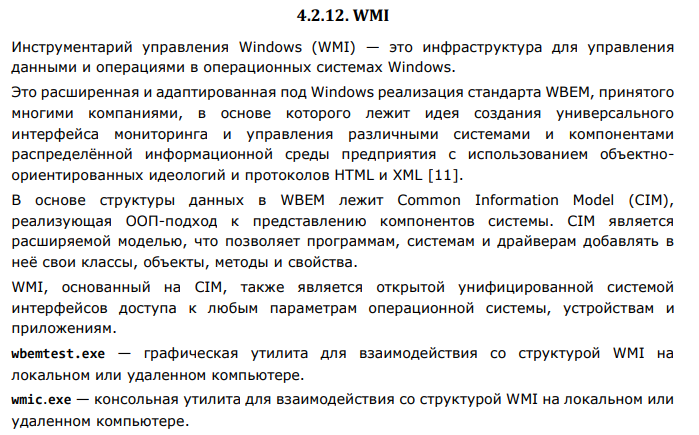


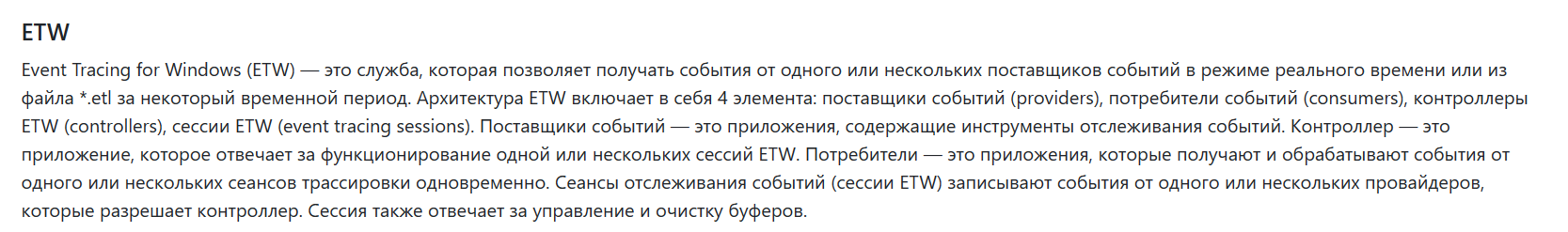
В Windows есть две оболочки командной строки: командная оболочка cmd и PowerShell. Каждая оболочка — это программная программа, которая обеспечивает прямую связь между оператором и операционной системой или приложением, предоставляя среду для автоматизации ИТ-операций.

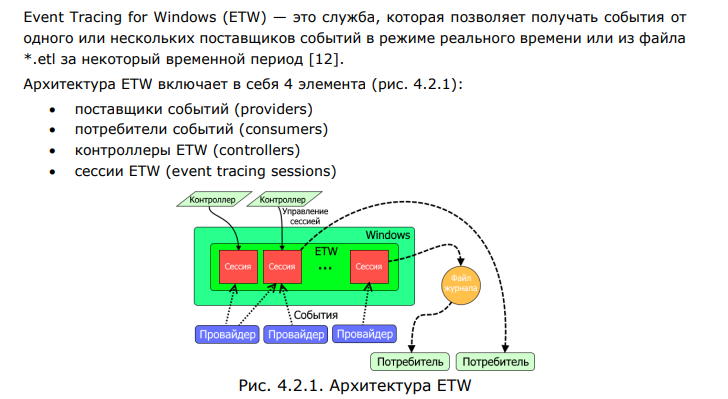
PowerShell был разработан для расширения возможностей командной оболочки для выполнения команд PowerShell, называемых командлетами. Командлеты похожи на команды Windows, но предоставляют более расширяемый язык сценариев. Команды Windows и командлеты PowerShell можно запускать в PowerShell, но командная оболочка может выполнять только команды Windows, а не командлеты PowerShell.

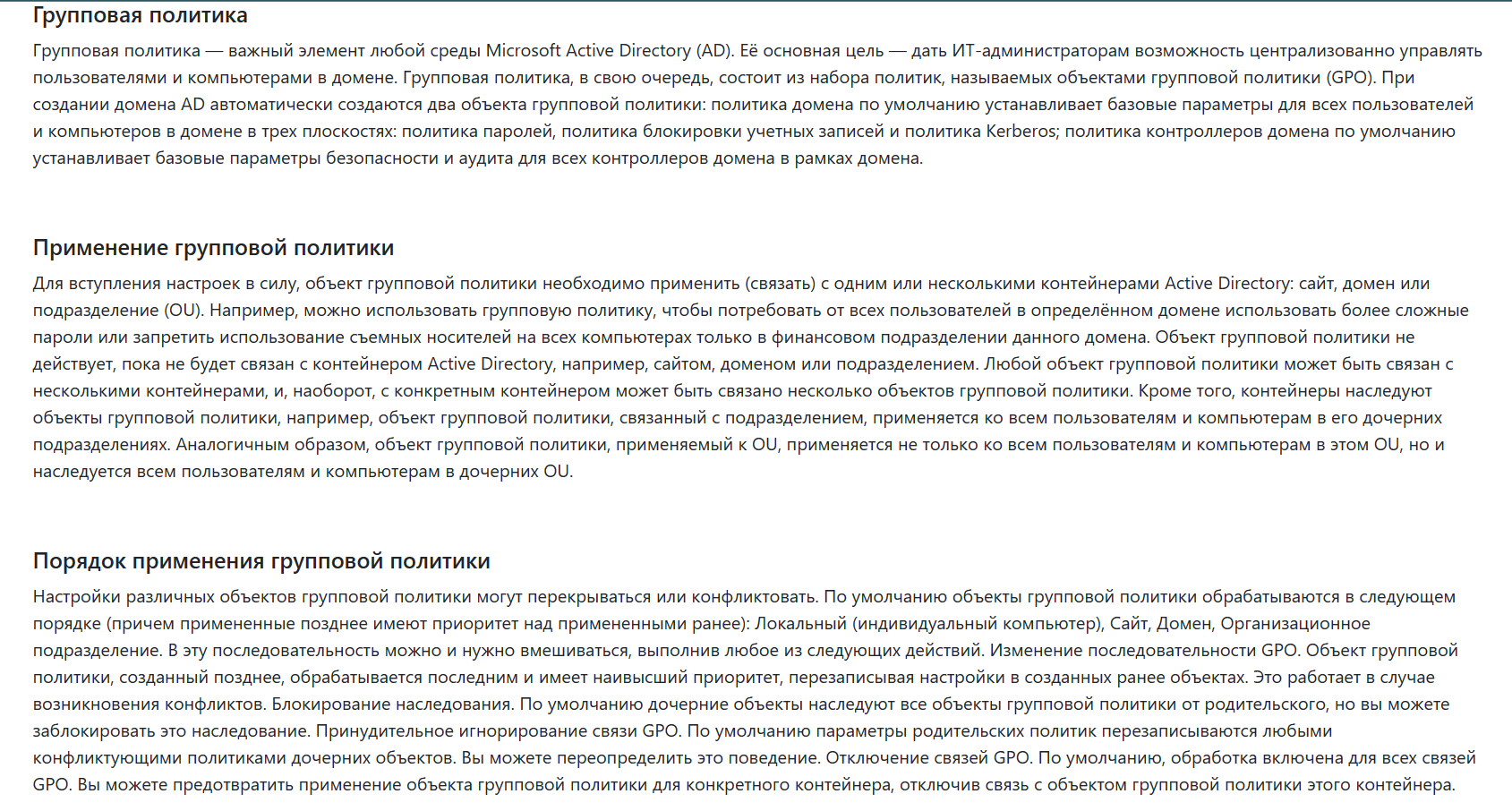


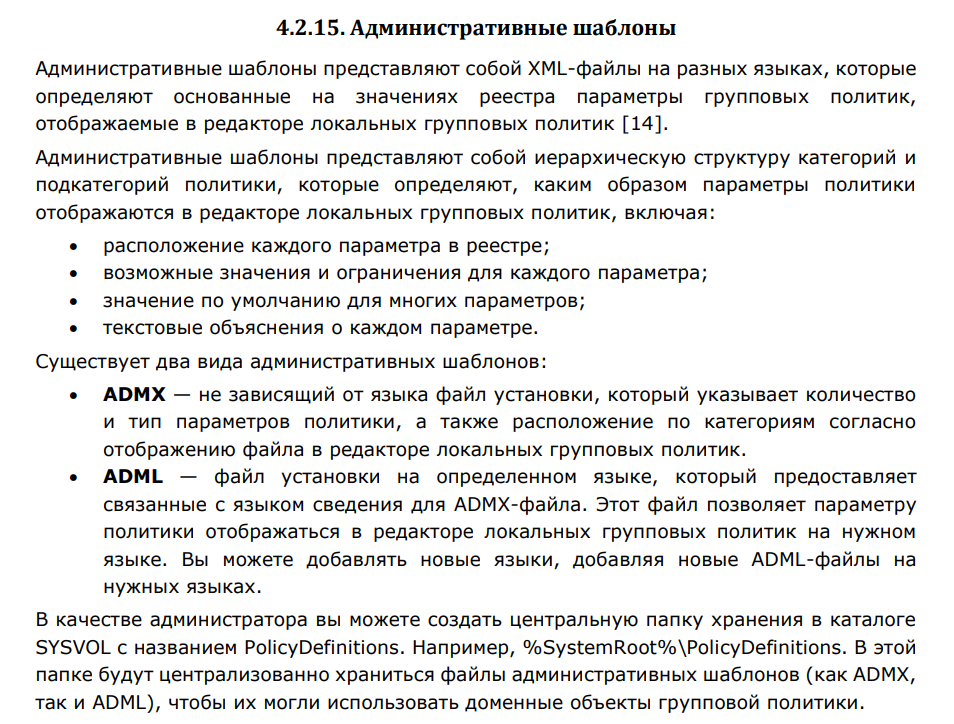
PowerShell — расширяемое средство автоматизации от Microsoft с открытым исходным кодом [10], состоящее из оболочки с интерфейсом командной строки и сопутствующего языка сценариев











1. Основные понятия системы UNIX. Пользователи системы, атрибуты пользователя. Файловая структура ОС.

Существует два основных объекта операционной системы UNIX, с которыми приходится работать пользователю – файлы и процессы.

Если рассматривать более подробно, операционную систему UNIX можно рассматривать как пирамиду (рис. 4.3.1). У основания пирамиды располагается аппаратное обеспечение, состоящее из центрального процессора, памяти, дисков, монитора и клавиатуры, а также других устройств. Операционная система работает на «голом железе». Ее функция заключается в управлении аппаратным обеспечением и предоставлении всем программам интерфейса системных вызовов. Эти системные вызовы позволяют программам пользователя создавать процессы, файлы и прочие ресурсы, а также управлять ими. Программы делают системные вызовы, помещая аргументы в регистры (или иногда в стек) и выполняя команду эмулированного прерывания для переключения из пользовательского режима в режим ядра. Поскольку на языке C невозможно написать команду эмулированного прерывания, то этим занимается библиотека, в которой есть по одной процедуре на системный вызов. Эти процедуры написаны на ассемблере, но они могут вызываться из языка C. Каждая такая процедура сначала помещает аргументы в нужное место, а затем выполняет команду эмулированного прерывания. Таким образом, чтобы обратиться к системному вызову read, программа на языке C должна вызвать библиотечную процедуру read. Кстати, в стандарте POSIX определен именно интерфейс библиотечных функций, а не интерфейс системных вызовов. Иначе говоря, стандарт POSIX определяет, какие библиотечные процедуры должна предоставлять соответствующая его требованиям система, каковы их параметры, что они должны делать и какие результаты возвращать. В стандарте даже не упоминаются реальные системные вызовы.

Прежде чем клиент сможет начать работу с ОС UNIX, он должен стать пользователем системы, т.е. получить имя, пароль и ряд других атрибутов. С точки зрения системы пользователь не обязательно человек. Пользователем является объект, который обладает определенными правами и может запускать на выполнение программы и владеть файлами. Пользователями могут быть отдельные клиенты, удаленные компьютеры или группы пользователей с одинаковыми правами и функциями. В системе существует один пользователь, обладающий неограниченными правами это суперпользователь или администратор системы (обычно с именем root)

Информация о пользователях обычно хранится в специальном файле: /etc/passwd, о группах — /etc/group. Этот файл доступен только для чтения. Писать в него может только администратор.

Как правило, все атрибуты пользователя хранятся в файле /etc/passwd. В конечном итоге, добавление пользователя в систему сводится к внесению в файл /etc/passwd соответствующей записи. Однако во многих системах информация о пользователе хранится и в других местах (например, в специальных базах данных), поэтому создание пользователя простым редактированием файла /etc/passwd может привести к неправильной регистрации пользователя, а иногда и к нарушениям работы системы. Вместо этого при возможности следует пользоваться специальными утилитами, поставляемыми с системой

— всего семь полей (атрибутов), разделенных двоеточиями. Рассмотрим подробнее каждый из атрибутов. name Регистрационное имя пользователя. Это имя пользователь вводит в ответ на приглашение системы login. passwd-encod Пароль пользователя в закодированном виде. Алгоритмы кодирования известны, но они не позволяют декодировать пароль. При входе в систему пароль, который вы набираете, кодируется, и результат сравнивается с полем passwd-encod. Поэтому часто пароль хранят в отдельном файле, а в поле passwd-encod ставится символ 'х' (в некоторых системах '!'). Пользователь, в данном поле которого стоит символ '\*', никогда не сможет попасть в систему. Дело в том, что алгоритм кодирования не позволяет символу '\*' появиться в закодированной строке. Таким образом, совпадение введенного и затем закодированного пароля и '\*' невозможно UID Идентификатор пользователя является внутренним представлением пользователя в системе. Этот идентификатор наследуется задачами, которые запускает пользователь, и файлами, которые он создает. По этому идентификатору система проверяет пользовательские права GID Определяет идентификатор первичной группы пользователя. Этот идентификатор соответствует идентификатору в файле /etc/group, который содержит имя группы и полный список пользователей, являющихся ее членами. Принадлежность пользователя к группе определяет дополнительные права в системе. comments Обычно, это полное "реальное" имя пользователя. Это поле может содержать дополнительную информацию, например, телефон или адрес электронной почты. home-dir Домашний каталог пользователя. shell Имя программы, которую UNIX использует в качестве командного интерпретатора.

Первоначально файловой системой в Linux была файловая система MINIX 1. Однако изза того обстоятельства, что имена файлов были ограничены в ней 14 символами (чтобы поддерживать совместимость с UNIX Version 7), а максимальный размер файла составлял 64 Мбайт (что было даже слишком много для жестких дисков того времени, размер которых составлял 10 Мбайт), интерес к более совершенным файловым системам появился сразу же после начала разработки системы Linux (которая началась примерно через 5 лет после выпуска MINIX 1). Первым улучшением стала файловая система ext, которая позволяла использовать имена файлов длиной 255 символов и размер файлов 2 Гбайт (однако она была медленнее, чем файловая система MINIX 1, так что поиски продолжались еще некоторое время). В итоге была изобретена файловая система ext2 (с длинными именами файлов, большими файлами и более высокой производительностью), которая и стала основной файловой системой. Однако Linux поддерживает несколько десятков файловых систем при помощи уровня виртуальной файловой системы Virtual File System (VFS), описанного в следующем разделе. Когда система Linux собирается, вам предлагается сделать выбор тех файловых систем, которые будут встроены в ядро. Другие можно загружать динамически (как модули) во время выполнения (если будет такая необходимость). Файл в системе Linux — это последовательность байтов произвольной длины (от 0 до некоторого максимума), содержащая произвольную информацию. Не делается различия между текстовыми (ASCII) файлами, двоичными файлами и любыми другими типами файлов. Значение битов в файле целиком определяется владельцем файла. Системе это безразлично. Имена файлов ограничены 255 символами. В именах файлов разрешается использовать все ASCII-символы, кроме символа NUL, поэтому допустимо даже состоящее из трех символов возврата каретки имя файла (хотя такое имя и не слишком удобно в использовании).

Для удобства файлы могут группироваться в каталоги. Каталоги хранятся на диске в виде файлов, и с ними можно работать практически так же, как с файлами. Каталоги могут содержать подкаталоги, что приводит к иерархической файловой системе. Корневой каталог называется / и всегда содержит несколько подкаталогов. Символ / используется также для разделения имен каталогов

Существует два способа задания имени файла в системе Linux (как в оболочке, так и при открытии файла из программы). Первый способ заключается в использовании абсолютного пути (absolute path), указывающего, как найти файл от корневого каталога. Также имена путей могут указываться относительно рабочего каталога. Путь, заданный относительно рабочего каталога, называется относительным путем (relative path).

Файлы ядра можно обнаружить в каталоге /boot. В заархивированном виде ядро системы Линукс располагается в vmlinuz, где z - символ, прямо указывающий на то, ядро сжато до минимального размера с целью экономии свободного пространства Файл initrd.img – это первичная файловая система, монтирующая в первую очередь, прежде чем будет осуществлено подключение реальных физических накопители к виртуальной файловой системе VFS

1. ОС UNIX: особенности процессов, сигналы, обработка сигналов.

Основными активными сущностями в системе Linux являются процессы. Каждый процесс выполняет одну программу и изначально получает один поток управления. Иначе говоря, у процесса есть один счетчик команд, который отслеживает следующую исполняемую команду. Linux позволяет процессу создавать дополнительные потоки (после того, как он начинает выполнение). Несколько независимых процессов могут работать одновременно. Более того, у каждого пользователя может быть одновременно несколько активных процессов, так что в большой системе могут одновременно работать сотни и даже тысячи процессов. Фактически на большинстве однопользовательских рабочих станций работают десятки фоновых процессов, называемых демонами (daemons). Они запускаются при загрузке системы из сценария оболочки. Процессы создаются в операционной системе Linux с помощью системного вызова fork, который создает точную копию исходного процесса, называемого родительским процессом (parent process). Новый процесс называется дочерним процессом (child process). У родительского и у дочернего процессов есть собственные (приватные) образы памяти. Открытые файлы используются родительским и дочерним процессами совместно. Процессы именуются своими PID-идентификаторами. При создании процесса его PID выдается родителю нового процесса. Если дочерний процесс желает узнать свой PID, то он может воспользоваться системным вызовом getpid.

Системные процессы являются частью ядра и всегда расположены в оперативной памяти. Демоны — это неинтерактивные процессы, которые запускаются обычным образом — путем загрузки в память соответствующих им программ (исполняемых файлов), и выполняются в фоновом режиме. К прикладным процессам относятся все остальные процессы, выполняющиеся в системе. Как правило, это процессы, порожденные в рамках пользовательского сеанса работы.

Сигналы — это способ информирования процесса со стороны ядра о происшествии некоторого события. По сути, сигналы являются программными прерываниями. Смысл термина «сигнал» состоит в том, что сколько бы однотипных событий в системе не произошло, по поводу каждой такой группы событий процессу будет подан ровно один сигнал. То есть, сигнал означает, что определяемое им событие произошло, но не несет информации о том, сколько именно произошло однотипных событий. Сигналы могут инициироваться одними процессами по отношению к другим процессам с помощью специального системного вызова kill. Процессы могут сообщить системе, какие действия следует предпринимать, когда придет входящий сигнал. Варианты такие: проигнорировать сигнал, перехватить его, позволить сигналу убить процесс (действие по умолчанию для большинства сигналов). Если процесс выбрал перехват посылаемых ему сигналов, он должен указать процедуру обработки сигналов. Когда сигнал прибывает, управление сразу же передается обработчику. Когда процедура обработки сигнала завершает свою работу, управление снова передается в то место, в котором оно находилось, когда пришел сигнал (это аналогично обработке аппаратных прерываний ввода-вывода).

1. Основные принципы функционирования Linux. Основные компоненты Linux. Дистрибутивы Linux. Файловая система Linux.

Организация программно-аппаратных средств во всех UNIX-совместимых системах организована по принципу клиент-сервер. С точки зрения распределения функций, возложенных на систему, все компьютеры в сети работают как один большой компьютер, который может быть легко дополнен аппаратными ресурсами, когда к сети подключается новый компьютер. Каждый пользователь работает с системой через виртуальный терминал, которых может быть до 12-ти в зависимости от версии Linux (на экране обозначается как tty1..tty12). В ОС Linux все файлы организованы в непрерывный поток байтов. Данные, вводимые с клавиатуры, представляют собой входной поток данных, а отображаемые данные – выходной поток. Поскольку процедуры ввода и вывода организованы также, как и файлы, то они могут свободно взаимодействовать с файлами. Чтобы передать данные из одной команды в другую, можно использовать конвейеры. Файл может быть каталогом или исполняемой программой (командой). Команда file помогает определить, для чего используется данный файл (например, текстовый файл или каталог). Файловая система в ОС Linux как и в большинстве других систем имеет иерархическую (древовидную) структуру. Вверху дерева всегда находится корневой каталог ROOT. В этой операционной системе также справедливо понятие текущего каталога. Каждый файл имеет относительное имя пути, которое определяет его принадлежность к какому-либо каталогу, и абсолютное имя пути, которое показывает весь путь файла, начиная от корневого каталога.

Название Linux в первую очередь относится к ядру операционной системы. Вообще говоря, в операционных системах этого семейства можно выделить следующие компоненты: ядро, модули, планировщик, файл подкачки, файловые системы, механизмы защиты, инструменты администрирования, серверные возможности, графический интерфейс пользователя, система управления пакетами и т. д.

Дистрибутивы состоят из отдельных пакетов, каждый из которых содержит какое-то приложение, утилиту или сервис. Отдельный пакет может содержать, например, веб-браузер, библиотеку для работы с графическими файлами в формате PNG, набор шрифтов и так далее. Программное обеспечение, содержащееся в пакете, поставляется в одном из двух основных видов: в виде бинарных файлов, которые предназначены для непосредственной установки в вашу систему, без какой-либо дополнительной обработки (например, компиляции); в виде исходных текстов, которые обычно содержат текст на каком-то языке программирования, заархивированный в формате tar и упакованный программой gzip, а также вспомогательные файлы, необходимые для компиляции приложения из файлов пакета.

Мы можем распределить дистрибутивы Linux на три группы: Enterprise Grade Linux (Red Hat Enterprise Linux, CentOS, SUSE Linux Enterprise Server, Debian GNU/Linux, Ubuntu LTS); Consumer Grade Linux (Fedora, Ubuntu non-LTS, openSUSE), Experimental and Hacker Linux (Arch, Gentoo). Дистрибутивы Enterprise Grade Linux предназначены для развертывания в крупных организациях с использованием оборудования предприятия. Они обычно включают более старые выпуски ядра и другое программное обеспечение, которое, как известно, работает надежно. Часто дистрибутивы портируют важные обновления, такие как исправления безопасности, на эти стабильные версии. Дистрибутивы Consumer Grade Linux больше ориентированы на малый бизнес или домашних пользователей и любителей. Они подготовлены для использования новейшего оборудования, установленного в системах потребительского уровня. Этим системам потребуются новейшие драйверы, чтобы максимально использовать новое оборудование. Дистрибутивы Experimental and Hacker Linux используют самые современные технологии. Они содержат самые последние версии программного обеспечения, даже если эти версии все еще содержат ошибки и непроверенные функции.

В операционной системе GNU/Linux все объекты являются файлами, в том числе и директории, которые служат для организации доступа к файлам. Существуют следующие типы файлов: обычные файлы, каталоги, символьные ссылки, блочные устройства, символьные устройства, сокеты, каналы. В любой файловой системе Linux всегда есть только один корневой каталог, который называется /. Пользователь Linux всегда работает с единым деревом каталогов, даже если данные расположены на разных носителях: жёстких или съёмных дисках, CD-ROM, сетевых дисках и т. д. Положение любого каталога в дереве каталогов точно и однозначно описывается при помощи полного пути. Полный путь всегда начинается от корневого каталога и состоит из перечисления всех вершин, встретившихся при движении по рёбрам дерева до искомого каталога включительно. Названия соседних вершин разделяются символом наклонной черты / (слэш). Для каждого процесса Linux определена текущая директория, с которого система начинает относительный путь при выполнении файловых операций. Между полным путём и относительным есть только одно существенное различие: относительный путь начинается от текущей директории, в то время как полный путь всегда начинается от корневой директории. То есть, первый символ полного пути всегда будет /. Точка . в начале имени файла делает его скрытым, то есть, он не показывается в выводе команды ls.

Домашний каталог (домашняя папка, домашняя директория) – предназначен для хранения собственных данных пользователя Linux и личных настроек для программ. Как правило, становится текущим непосредственно после регистрации пользователя в системе. Полный путь к домашнему каталогу хранится в переменной окружения HOME, в полном пути к файлу можно заменять на знак тильды ~. Для обычных пользователей домашний каталог находится в директории /home.

FHS (Filesystem Hierarchy Standard) – стандарт файловой системы, специфичной для GNU/Linux, создан в 1994-1996 гг. Текущая версия стандарта FHS 3.0 выпущена 03.06.2015 г. Некоторые Linux-системы отвергают FHS и следуют своему собственному стандарту.В FHS все файлы и каталоги находятся внутри корневого каталога, даже если они расположены на различных физических носителях. В корневой директории должны быть следующие директории: bin, boot, dev, etc, lib, media, mnt, opt, run, sbin, srv, tmp, usr, var. При наличии соответствующих подсистем добавляются папки home, root и lib64.

1. ОС Linux: управление процессами, выполнение задач в фоновом режиме, изменение приоритетов выполняющихся программ.

Процессы существуют в иерархии: после загрузки ядра в память запускается первый процесс (init или systemd), который, в свою очередь, запускает другие процессы, которые, опять же, могут запускать другие процессы. Каждый раз, когда пользователь вводит команду, запускается программа и генерируется один или несколько процессов. Каждый процесс имеет уникальный идентификатор (PID) и идентификатор родительского процесса (PPID). Это положительные целые числа, которые назначаются в последовательном порядке

Команда top динамически отображает все запущенные процессы:

Команда ps выводит статическую информацию о процессах. Без опций выводит только процессы, относящиеся к текущей оболочке

Иерархию процессов можно увидеть с помощью команды pstree. Завершить процесс можно с помощью команды kill.

Для выполнения команды в фоновом режиме достаточно добавить в конце символ амперсанда &. В выводе терминала будут отображены порядковый номер задачи (в квадратных скобках) и идентификатор процесса. Работая в фоновом режиме, команда все равно продолжает выводить сообщения в терминал, из которого была запущена. Узнать состояние всех остановленных и выполняемых в фоновом режиме задач в рамках текущей сессии терминала можно при помощи утилиты jobs c использованием опции -l. Вывод содержит порядковый номер задачи, идентификатор фонового процесса, состояние задачи и название команды, которая запустила задание. В любое время можно вернуть процесс из фонового режима на передний план. Для этого служит команда fg. Если в фоновом режиме выполняется несколько программ, следует также указывать номер. Для завершения фонового процесса применяют команду kill с номером программы. Если изначально процесс был запущен обычным способом, его можно перевести в фоновый режим, выполнив следующие действия: 1) остановить выполнение команды, нажав комбинацию клавиш Ctrl+Z; 2) перевести процесс в фоновый режим при помощи команды bg.

Утилита nice — программа, предназначенная для запуска процессов с изменённым приоритетом nice. Приоритет nice (целое число) процесса используется планировщиком процессов ядра ОС при распределении процессорного времени между процессами. Приоритет nice — число, указывающее планировщику процессов ядра ОС приоритет, который пользователь хотел бы назначить процессу. Утилита nice, запущенная без аргументов, выводит приоритет nice, унаследованный от родительского процесса. nice принимает аргумент «смещение» в диапазоне от -20 (наивысший приоритет) до +19 (низший приоритет). Если указать смещение и путь к исполняемому файлу, утилита nice получит приоритет своего процесса, изменит его на указанное смещение и использует системный вызов семейства exec() для замещения кода своего процесса кодом из указанного исполняемого файла. Команда nice сделает то же, но сначала выполнит системный вызов семейства fork() для запуска дочернего процесса (sub-shell). Если смещение не указано, будет использовано смещение +10. Привилегированный пользователь (root) может указать отрицательное смещение.

Для того, чтобы изменить приоритет у существующего процесса (т.е. такого процесса, который ранее был уже запущен), необходимо воспользоваться командой renice [значение приоритета] -p [id процесса]. У запущенной программы с помощью команды renice можно изменить назначенный приоритет. Согласно правилам, обычный пользователь может только увеличивать значение nice (уменьшать приоритет) любого процесса.

1. Понятие безопасности ОС. Основные угрозы безопасности ОС. Методы и защитные механизмы операционных систем.

Безопасность операционной системы (ОС) представляет собой комплекс мер и механизмов, направленных на обеспечение защиты компьютерной системы от различных угроз

Во многих работах, посвященных безопасности, безопасность информационных систем разбита на три компонента: конфиденциальность, целостность и доступность. Вместе все три компонента часто называют CIA (Confidentiality, Integrity, Availability). Конфиденциальность направлена на сохранение секретности данных. Целостность означает, что пользователи, не обладающие необходимыми правами, не должны иметь возможности изменять какие-либо данные без разрешения их владельцев (в этом контексте изменение данных включает в себя не только внесение в них изменений, но и их удаление или добавление в них ложных данных). Доступность означает, что никто не может нарушить работу системы и вывести ее из строя. Позже были добавлены дополнительные свойства, такие как аутентичность (authenticity), идентифицируемость (accountability), неотвергаемость (nonrepudiability), закрытость (privacy) и др.

Классификация угроз по цели атаки:

• несанкционированное чтение информации;

• несанкционированное изменение информации;

• несанкционированное уничтожение информации;

• полное или частичное разрушение операционной системы.

Классификация угроз по принципу воздействия на операционную систему:

• использование известных (легальных) каналов получения информации, например, угроза несанкционированного чтения файла, доступ пользователей к которому определен некорректно – разрешен доступ пользователю, которому, согласно политике безопасности, доступ должен быть запрещен;

• использование скрытых каналов получения информации, например, угроза использования злоумышленником недокументированных возможностей операционной системы;

• создание новых каналов получения информации с помощью программных закладок. Классификация угроз по типу используемой злоумышленником уязвимости защиты:

• неадекватная политика безопасности, в том числе и ошибки администратора системы;

• ошибки и недокументированные возможности программного обеспечения операционной системы, в том числе и так называемые люки – случайно или преднамеренно встроенные в систему «служебные входы», позволяющие обходить систему защиты;

• ранее внедренная программная закладка.

Классификация угроз по характеру воздействия на ОС:

• активное воздействие – несанкционированные действия злоумышленника в системе;

• пассивное воздействие – несанкционированное наблюдение злоумышленника за процессами, происходящими в системе.

Угрозы безопасности ОС можно также классифицировать по таким признакам, как способ действий злоумышленника, используемые средства атаки, объект атаки, способ воздействия на объект атаки, состояние атакуемого объекта ОС на момент атаки. Операционная система может подвергнуться следующим типичным атакам: • сканирование файловой системы. Злоумышленник просматривает файловую систему компьютера и пытается прочесть (или скопировать) все файлы подряд. Рано или поздно обнаруживается хотя бы одна ошибка администратора. В результате злоумышленник получает доступ к информации, который должен быть ему запрещен;

• подбор пароля. Существует несколько методов подбора паролей пользователей: o тотальный перебор; o тотальный перебор, оптимизированный по статистике встречаемости символов или с помощью словарей; o подбор пароля с использованием знаний о пользователе (его имени, фамилии, даты рождения, номера телефона и т. д.);

• кража ключевой информации. Злоумышленник может подсмотреть пароль, набираемый пользователем, или восстановить набираемый пользователем пароль по движениям его рук на клавиатуре. Носитель с ключевой информацией (смарткарта, Touch Memory и т. д.) может быть просто украден;

• сборка мусора. Во многих операционных системах информация, уничтоженная пользователем, не уничтожается физически, а помечается как уничтоженная (так называемый мусор). Злоумышленник восстанавливает эту информацию, просматривает ее и копирует интересующие его фрагменты;

• превышение полномочий. Злоумышленник, используя ошибки в программном обеспечении ОС или политике безопасности, получает полномочия, превышающие те, которые ему предоставлены в соответствии с политикой безопасности. Обычно это достигается путем запуска программы от имени другого пользователя;

• программные закладки. Программные закладки, внедряемые в операционные системы, не имеют существенных отличий от других классов программных закладок;

• жадные программы – это программы, преднамеренно захватывающие значительную часть ресурсов компьютера, в результате чего другие программы не могут выполняться или выполняются крайне медленно. Запуск жадной программы может привести к краху операционной системы.

Перечислим основные административные меры защиты [2]. • Постоянный контроль корректности функционирования операционной системы, особенно ее подсистемы защиты. Такой контроль удобно организовать, если операционная система поддерживает автоматическую регистрацию наиболее важных событий (event logging) в специальном журнале. • Организация и поддержание адекватной политики безопасности. Политика безопасности ОС должна постоянно корректироваться, оперативно реагируя на попытки злоумышленников преодолеть защиту операционной системы, а также на изменения в конфигурации операционной системы, установку и удаление прикладных программ. • Осведомление пользователей операционной системы о необходимости соблюдения мер безопасности при работе с ОС и контроль за соблюдением этих мер. • Регулярное создание и обновление резервных копий программ и данных ОС. • Постоянный контроль изменений в конфигурационных данных и политике безопасности ОС.

Аутентификация субъекта доступа заключается в том, что субъект предоставляет операционной системе, помимо идентифицирующей информации, еще и аутентифицирующую информацию, подтверждающую, что он действительно является тем субъектом доступа, к которому относится идентифицирующая информация. Авторизация субъекта доступа происходит после успешной идентификации и аутентификации. При авторизации субъекта ОС выполняет действия, необходимые для того, чтобы субъект мог начать работу в системе.

Криптография – это метод защиты информации путем использования закодированных алгоритмов, хэшей и подписей. Информация может находиться на этапе хранения (например, файл на жестком диске), передачи (например, электронная связь между двумя или несколькими сторонами) или использования (при применении для вычислений) [3]. Замысел криптографии заключается в том, чтобы закодировать открытый текст (plaintext) — сообщение или файл, превратив его в зашифрованный текст (ciphertext), чтобы о том, как его снова превратить в открытый текст, знали только те, кто имеет на это право. Для всех остальных зашифрованный текст будет лишь непонятным набором битов. Как бы странно это ни прозвучало для новичков, но алгоритмы (функции), используемые для шифрования и дешифрования, всегда должны быть открытыми. Попытка хранить их в секрете практически никогда не срабатывает и создает у людей, пытающихся сохранить секреты, ложное чувство безопасности. В коммерции такая тактика называется безопасностью за счет неизвестности (security by obscurity) и используется только дилетантами

Криптография преследует четыре основных цели [3]. • Конфиденциальность – предоставляет доступ к информации только авторизованным пользователям. • Целостность – гарантирует, что над информацией не производились манипуляции. • Подлинность – подтверждает подлинность информации или личность пользователя. • Обеспечение невозможности отказа – лишает возможности отрицать прежние обязательства или действия.

Домен (domain) представляет собой множество пар (объект, права доступа). Каждая пара определяет объект и некоторое подмножество операций, которые могут быть выполнены в отношении этого объекта. Права доступа (rights) означают в данном контексте разрешение на выполнение той или иной операции. Зачастую домен соотносится с отдельным пользователем, сообщая о том, что может, а что не может сделать этот пользователь, но он может также иметь и более общий характер, распространяясь не только на отдельного пользователя. К примеру, сотрудники одной группы программистов, работающие над одним и тем же проектом, могут целиком принадлежать к одному и тому же домену и иметь доступ к файлам проекта.

1. Механизмы безопасности в операционных системах семейства Windows.

Идентификатор безопасности (SID) — это уникальное значение переменной длины, используемое для идентификации доверенного лица. Каждая учетная запись имеет уникальный идентификатор безопасности, выданный центром сертификации, таким как контроллер домена Windows, и хранящийся в базе данных безопасности. Каждый раз, когда пользователь входит в систему, система получает идентификатор безопасности для этого пользователя из базы данных и помещает его в маркер доступа для этого пользователя. Система использует SID в маркере доступа для идентификации пользователя во всех последующих взаимодействиях с безопасностью Windows.

Маркер доступа — это объект, описывающий контекст безопасности процесса или потока. Сведения в маркере включают удостоверение и привилегии учетной записи пользователя, связанной с процессом или потоком. Когда пользователь входит в систему, система проверяет пароль пользователя, сравнивая его с информацией, хранящейся в базе данных безопасности. Если пароль прошел проверку подлинности, система создает маркер доступа. Каждый процесс, выполняемый от имени этого пользователя, имеет копию этого маркера доступа

Маркеры доступа содержат следующие сведения: • Идентификатор безопасности (SID) для учетной записи пользователя • Идентификаторы безопасности для групп, членом которых является пользователь • Идентификатор безопасности входа, который идентифицирует текущий сеанс входа в систему. • Список привилегий, которыми пользовались пользователи или группы пользователей. • Идентификатор безопасности владельца • Идентификатор безопасности для основной группы • DACL по умолчанию, используемый системой, когда пользователь создает защищаемый объект без указания дескриптора безопасности. • Источник маркера доступа • Является ли маркер основным или маркером олицетворения • Необязательный список ограничений идентификаторов безопасности • Текущие уровни олицетворения • Другая статистика

Список управления доступом (ACL) — это список записей управления доступом (ACE). Каждая запись ACE в ACL идентифицирует доверенного лица и указывает разрешенные, запрещенные или проверяемые права доступа для этого доверенного лица. Дескриптор безопасности защищаемого объекта может содержать два типа списков ACL: DACL и SACL.

Список управления дискреционным доступом (DACL)

Список управления дискреционным доступом (DACL) определяет доверенных лиц, которым разрешен или запрещен доступ к защищаемому объекту. Когда процесс пытается получить доступ к защищаемому объекту, система проверяет записи ACE в списке DACL объекта, чтобы определить, предоставлять ли к нему доступ. Если у объекта нет DACL, система предоставляет полный доступ всем. Если в DACL объекта нет записей ACE, система отклоняет все попытки доступа к объекту, поскольку DACL не разрешает никаких прав доступа. Система последовательно проверяет ACE, пока не найдет один или несколько ACE, которые разрешают все запрошенные права доступа, или пока какое-либо из запрошенных прав доступа не будет отклонено.

Список управления доступом к системе (SACL)

Список управления доступом к системе (SACL) позволяет администраторам регистрировать попытки доступа к защищенному объекту. Каждый ACE определяет типы попыток доступа со стороны указанного доверенного лица, которые заставляют систему создавать запись в журнале событий безопасности. ACE в SACL может генерировать записи аудита, когда попытка доступа не удалась, когда она увенчалась успехом, или в обоих случаях.

Логическая структура Active Directory представляет собой иерархическую структуру доменов Active Directory и подразделений в лесу. Взаимосвязь компонентов логической структуры контролирует доступ к хранимым данным, а также то, как информация реплицируется между различными контроллерами домена в лесу.

Лес — это высший уровень иерархии логических структур. Лес Active Directory представляет собой отдельный автономный каталог. Лес — это граница безопасности. Это означает, что администраторы леса имеют полный контроль над всем доступом к информации, хранящейся внутри леса, и к контроллерам домена, которые используются для реализации леса.

Домены разделяют информацию, хранящуюся внутри каталога, на более мелкие части, чтобы информацию было легче хранить на различных контроллерах домена, а администраторы имели большую степень контроля над репликацией. Данные, хранящиеся в каталоге, реплицируются по всему лесу от одного контроллера домена к другому. Некоторые данные, относящиеся ко всему лесу, реплицируются на все контроллеры домена. Другие данные, относящиеся только к определенному домену, реплицируются только на контроллеры домена в этом конкретном домене. Хороший дизайн домена позволяет реализовать эффективную топологию репликации. Это важно, поскольку позволяет администраторам управлять потоком данных по сети, то есть контролировать, сколько данных реплицируется и где происходит этот трафик репликации.

Группы безопасности — это способ объединения учетных записей пользователей, учетных записей компьютеров и других групп в управляемые единицы. Работа с группами вместо отдельных пользователей помогает упростить обслуживание сети и администрирование. Active Directory имеет два типа групп: • Группы безопасности. Используйте для назначения разрешений общим ресурсам. • Группы рассылки: создание списков рассылки электронной почты.

Группы безопасности могут обеспечить эффективный способ назначения доступа к ресурсам в сети. С помощью групп безопасности можно выполнять следующие действия. • Назначение прав пользователей группам безопасности в Active Directory. Например, включив пользователя в группу Администраторы схемы, можно предоставить ему разрешение изменять схему Active Directory. • Назначение разрешения группам безопасности для ресурсов. Группы безопасности могут быть использованы в списках управления доступом (DACLs), определяющих разрешения на ресурсы и объекты. Когда администраторы назначают разрешения для ресурсов, таких как общие папки или принтеры, они должны назначать эти разрешения группе безопасности вместо отдельных пользователей. Разрешения назначаются группе один раз вместо нескольких раз каждому отдельному пользователю. Каждая учетная запись, добавленная в группу, получает права, назначенные этой группе в Active Directory. Пользователь получает разрешения, определенные для этой группы.

Группы рассылки Группы рассылки можно использовать только для отправки электронной почты в коллекции пользователей с помощью почтового приложения, например, Exchange Server. Группы рассылки не включены в систему безопасности, поэтому их нельзя включить в списки управления доступом.

Каждая группа имеет область, определяющий степень применения группы в дереве домена или лесу. Область группы определяет, где можно предоставить разрешения сети для группы. Active Directory определяет следующие три области для групп: • Универсальная • Глобальная • Локальная в домене

33. Механизмы безопасности в операционных системах семейства Linux.

В операционной системе GNU/Linux существуют следующие типы файлов:

• обычные файлы

• каталог

• символьные ссылки

• блочные устройства

• символьные устройства

• сокеты

• каналы

Каждый тип имеет собственное обозначение одним символом.

Обычные файлы (-) Сюда относятся все файлы с данными, играющими роль ценной информации сами по себе. Linux не различает текстовые файлы, исполняемые или картинки. В любом случае это будет обычный (regular) файл. Все они обозначаются знаком "-". Остальные типы файлов считаются специальными (special).

Каталоги (d) В Linux каталог представляет собой такой тип файла, данными которого является список имен других файлов и каталогов, вложенных в данный каталог. Напрямую, то есть через какой-либо редактор, пользователь не может редактировать данные файла-каталога. Редактированием занимается ядро операционной системы, получая, в том числе от пользователя, команды создания файла, удаления и др. В файле каталога осуществляется связь между именами файлов (словесного обозначения для людей) и их индексными дескрипторами (истинным именем-числом, которым оперирует ОС).

Символьные ссылки (l) Символьная ссылка – это файл, в данных которого содержится адрес другого файла по его имени (а не индексному дескриптору). Выполнение символьной ссылки приводит к открытию файла, на который она указывает. Это аналог ярлыков в операционной системе Windows. Если удалить исходный файл, то символьная ссылка продолжит существовать. Она попрежнему будет указывать на файл, которого уже нет.

Ссылки на файлы и директории В системе Linux есть два типа ссылок: Символьные ссылки (symbolic links, или soft links) указывают путь к другому файлу. Если удалить файл, на который указывает ссылка (называемая target), ссылка все равно будет существовать, но перестанет «работать», поскольку теперь она указывает на несуществующий файл. Может указывать на файл в другой файловой системе. Жесткие ссылки (hard links) указываюn на то же место (inode) на диске, где находится исходный файл. После создания жесткой ссылки она становится неотличимой от оригинального имени файла, то есть как бы становится дополнительным именем файла. После удаления исходного файла жесткая ссылка продолжает существовать. Файл удаляется, когда количество жестких ссылок на него становится равным нулю. Должны указывать на файл в этой же файловой системе.

Символьные (c) и блочные устройства (b) Файлы устройств предназначены для обращения к аппаратному обеспечению компьютера (дискам, принтерам, терминалам и др.). Когда происходит обращение к файлу устройства, то ядро операционной системы передает запрос драйверу этого устройства. К символьным устройствам обращение происходит последовательно (символ за символом). Примером символьного устройства может служить терминал.

Сокеты (s) и каналы (p) Чтобы понять, что такое канал и сокет, необходимо понимание процессов в операционной системе. И каналы, и сокеты организуют их взаимодействие. Пользователь с данными типами файлов почти не сталкивается. Ключевым отличием канала от сокета является то, что канал однонаправлен. Через канал один процесс всегда передает данные второму, но не наоборот. Сокеты позволяют передавать данные в разных направлениях, т. е. осуществляют связь.

Временные файлы — это файлы, используемые программами для хранения данных, которые нужны только в течение короткого времени. Это могут быть данные о запущенных процессах, журналы ошибок, промежуточные файлы, используемые при преобразовании данных, файлы кэша и т.д

В Linux у каждого файла и каждого каталога есть два владельца: пользователь и группа. Эти владельцы устанавливаются при создании файла или каталога. Пользователь, который создаёт файл становится владельцем этого файла, а первичная группа, в которую входит этот же пользователь, так же становится владельцем этого файла. Чтобы определить, есть ли у вас как у пользователя права доступа к файлу или каталогу, оболочка проверяет владение ими

Команда chown позволяет сменить владельца

Когда пользователь создает файл, применяется владение по умолчанию. Пользователь, который создает файл, автоматически становится владельцем этого файла, а основная группа этого пользователя автоматически становится владельцем этого файла. Обычно это группа, которая указана в файле /etc/passwd в качестве основной группы пользователя. Однако если пользователь является членом нескольких групп, он может изменить эффективную основную группу.

Управление:

Опция –l команды ls выводит информацию о файлах и директориях в длинном формате, включая разрешения и владельцев.

Первая группа разрешений относится к пользователю-владельцу, вторая – к группе-владельцу, третья – ко всем остальным пользователям. Применяется наиболее точное разрешение. Значение разрешений для файлов:

♣ r – содержимое файла может быть прочитано

♣ w – содержимое файла может быть изменено

♣ x – файл может быть выполнен как команда Значение разрешений для каталогов:

♣ r – содержимое каталога (имена файлов) может быть перечислено

♣ w – любой файл в каталоге может быть создан или удален

♣ x – к содержимому каталога возможен доступ (в зависимости от разрешений на файлы) Числовое обозначение разрешений: rw-rw-r-- = 664

--- 0

r-- 4

--x 1

r-x 5

-w- 2

rw- 6

-wx 3

rwx 7

Команда chmod позволяет изменять разрешения двумя способами: символьным и числовым

При создании файла или директории, среда операционной системы присваивает им определенные права доступа по умолчанию. umask - пользовательская маска (user mask), которая используется для определения конечных прав доступа.

В операционной системе Linux базовые права для директории равны 0777 (rwxrwxrwx), а для файла 0666 (rw-rw-rw)

Атрибуты

В Linux атрибуты файла — это свойства метаданных, которые описывают поведение файла. Например, атрибут может указывать, сжат ли файл, или указывать, можно ли удалить файл [3]. Некоторые атрибуты, такие как неизменяемость, могут быть установлены или очищены, в то время как другие, такие как шифрование, доступны только для чтения и могут быть только просмотрены. Поддержка определенных атрибутов зависит от используемой файловой системы

Чтобы получить полный список всех атрибутов файла и флагов, введите man chattr в свой терминал. По умолчанию атрибуты файла не сохраняются при копировании файла с помощью таких команд, как cp или rsync.

Примеры атрибутов:

A - (no atime updates) не изменять время последнего обращения, что может благоприятно повлиять на производительность файловой системы, если обращение происходит очень часто.

a - (append only) в файл можно только дописывать, но нельзя удалять/переименовывать (удобно для логов). Если установлено на каталог, то находящиеся там файлы удалять нельзя, но можно создавать новые и модифицировать существующие.

c - (compressed) производится прозрачное сжатие на диске информации файла ядром, а при доступе возвращаются несжатые данные.

D - (synchronous directory updates) при модификации директории изменения синхронно записываются на диск.

d - (no dump) игнорировать при создании резервной копии программой dump.

i - (immutable) пожалуй самый используемый и полезный бит, который запрещает любые изменения файла (нельзя удалять, переименовывать и модифицировать файл). Для директории данный флаг позволяет модифицировать в ней файлы, но нельзя удалять или создавать новые.

j - (data journalling) если файловая система смонтирована с параметрами «data=ordered» или «data=writeback», данные файла с этим атрибутом сохраняются сначала в журнал файловой системы, и только потом в файл. При монтировании с параметром «data=journal» данные и так сохраняются сначала в журнал, поэтому атрибут не действует. Этот атрибут может снять и установить только root.

s - (secure deletion) полное удаление файла (место на диске, где он находился, после заполняется нулями) .

S - (synchronous updates) прямая запись на диск, без кэширования (обновления в файле происходит на диске синхронно с приложением, изменяющим данный файл) .

u - (undeletable) при удалении файла с таким атрибутом, его содержимое сохраняется, что позволяет успешно использовать инструменты для восстановления удаленных файлов.

T - каталог с таким атрибутом считается расположенным на вершине иерархии директорий с целью использования метода распределения блоков по Orlov.

t - к файлу с таким атрибутом нельзя присоединить в конец другой файл (tail-merging). На момент написания ext2 и ext3 не поддерживали (не считая очень экспериментальных патчей) tail-merging.

E - показывает, что при сжатии файла были ошибки. Нельзя установить/снять с помощью chattr, можно лишь посмотреть командой lsattr. e - показывает, что файл использует дополнения для размещения блоков на диске. Нельзя установить/снять с помощью chattr, можно лишь посмотреть командой lsattr.

I - показывает, что каталог был проиндексирован при использовании htree. Нельзя установить/снять с помощью chattr, можно лишь посмотреть командой lsattr.

H - показывает, что файл хранит свои блоки в единицах файловой системы, а не в единицах секторов, это означает, что файл имеет размер более 2TB (или когда-то занимал). Нельзя установить/снять с помощью chattr, можно лишь посмотреть командой lsattr.

X - показывает, что к сжатому файлу можно получить прямой непосредственный доступ. Нельзя установить/снять с помощью chattr, можно лишь посмотреть командой lsattr

Z - показывает, что сжатый файл is dirty (?). Нельзя установить/снять с помощью chattr, можно лишь посмотреть командой lsattr.

У каждого файла имеется определённый набор свойств в файловой системе. Например, это права доступа, владелец, имя, метки времени. В Linux каждый файл имеет довольно много свойств, например, права доступа устанавливаются трижды (для владельца, группы и всех прочих), метки времени также бывают трёх разных видов (время создание, доступа и изменения). Большинство этих свойств была рассмотрена в предыдущих материалах (права доступа и владелец в этой теме, имя файла – в теме 4.3 (подраздел 4.3.6). Осталось рассмотреть метки времени. У каждого файла доступны следующие метки времени [4]:

• Доступ

• Модифицирован

• Изменён

• Создан

Метки времени (timestamps) Доступ (Access - last access) — время, когда файл был прочитан последний раз. Это время меняется при доступе таких системных вызовов как mknod(2), utimes(2) и read(2). Если это текстовый файл, то дата последнего доступа обновляется при каждом его открытии. Если это исполнимый файл, то дата доступа обновится при его запуске. При некоторых опциях монтирования диска (noatime или relatime) это значение может быть неточным Модифицирован (Modify - last modified) — время последнего изменения содержимого файла. То есть если это текстовый файл, то время модификации поменяется когда вы его откроете и удалите какое-то слово или что-то допишите. Меняется системными вызовами mknod(2), utimes(2) и write(2). Изменён (Change - last changed) — Время последнего изменения метаданных файлов в файловой системе. То есть если в файле изображения вы измените EXIF метаданные — это будет модификация (поскольку по сути поменяется содержимое файла). Примером Изменения файла является смена разрешений доступа к нему (чтение, запись, выполнение), смена владельца, группы и т. д. Меняется с chmod(2), chown(2), link(2), mknod