# Тема 1.1. Введение в операционные системы

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

Определение

Операционная система - это комплекс программ, которые служат интерфейсом между модулями вычислительных систем и прикладными программными приложениями, а также управляют компьютерным оборудованием и процессами вычислений, эффективным распределением вычислительных мощностей среди процессов вычислений.

Назначение операционных систем

ОС управляет всеми устройствами компьютерной системы (процессорами, оперативной памятью, дисками, клавиатурой, монитором, принтерами, сетевыми устройствами и др.) и обеспечивает пользователя удобным интерфейсом для работы с аппаратурой. Две основные функции 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. Поэтому в специальной литературе ОС представляется всегда двояко: как расширенная виртуальная машина и как система управления ресурсами.

Состав операционных систем

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 операционных систем

Некоторые функции ОС: 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; обеспечение работы системы управления файлами и/или систем управления БД; планирование и диспетчеризация задач; управление памятью, организация виртуальной памяти; защита одной программы от влияния другой; обеспечение сохранности данных; аутентификация, авторизация и другие средства обеспечения безопасности.

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

Классификация ОС по способу 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.

 Классификация ОС количеству единовременно решаемых задач

Однозадачные; многозадачные.

 Классификация ОС возможности работы в компьютерной сети

Локальные - автономные ОС, которые не позволяют работать с компьютерными сетями; сетевые - ОС с поддержкой компьютерных сетей.

 Классификация ОС 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.

 Классификация ОС по типу 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.

Основные этапы развития ОС, поколения ОС

Так как возможности операционных систем зависят от возможностей компьютеров, для которых они разрабатываются, то и этапы их развития сходны. Первый настоящий цифровой компьютер был изобретен английским математиком Чарльзом Бэббиджем, но он был механическим и так и не заработал должным образом. Появление электронных приборов сделало возможным появление электронных вычислительных машин, и долгие годы в русскоязычной среде широко использовалось сокращение ЭВМ. В англоязычной среде использовалось слово компьютер (вычислитель). По мере прогресса в разработке электронных компонентов росли возможности компьютеров и уменьшался их размер.

Первое поколение (1945-1955 гг.)

Первое поколение компьютеров строилось преимущественно на **электронных лампах.** Они были достаточно примитивны. В начале программы писались непосредственно в машинных кодах и загружались в память компьютера с панели переключателей вручную. Позже появилась возможность загрузки с перфокарт. Вычислительная система выполняла одновременно только одну операцию. На этом этапе не было ни языков программирования, ни операционных систем как таковых. Так как электронные лампы часто выходили из строя, не было необходимости в постоянно работающей части программного обеспечения. В самом конце этого периода появляется первое системное программное обеспечение, например, компилятор Fortran и ассемблер для IBM-701. В целом, первый период характеризуется крайне высокой стоимостью вычислительных систем, их малым количеством и низкой эффективностью использования. Существенная часть времени уходила на подготовку запуска программы, а сами программы выполнялись строго последовательно (такой режим работы называется последовательной обработкой данных).

Второе поколение (1955-1965 гг.)

Второе поколение компьютеров характеризуется использованием **транзисторов**, что повысило их надёжность и продлило время непрерывной работы. Подготовка компьютера к выполнению очередного задания приводило к непроизводительному расходованию дорогостоящего машинного времени. Было принято решение использовать небольшие недорогие компьютеры для записи заданий на магнитную ленту и затем для настоящих вычислений использовали мощные дорогостоящие машины. Программа, которая считывала задания с ленты и запускала их на выполнение, является прообразом современной операционной системы.

Третье поколение (1965-1980 гг.)

Компьютеры третьего поколения использовали **малые интегральные схемы**, что дало им преимущество в цене и качестве по сравнению с машинами второго поколения. Вычислительная техника становится более надёжной и дешёвой. Растет сложность и количество задач, решаемых компьютерами. Повышается производительность процессоров. Самым важным достижением явилась **многозадачность**. В компьютерах предыдущего поколения вычисления останавливались на время ввода-вывода. Проблема была решена разбиением памяти на несколько частей, называемых разделами, в каждом из которых выполнялось отдельное задание. Пока одно задание ожидало завершения работы устройства ввода-вывода, другое могло использовать центральный процессор. Если в оперативной памяти содержалось достаточное количество заданий, центральный процессор мог быть загружен почти на все 100 % времени.

Другим важным плюсом операционных систем третьего поколения стала способность считывать задание с перфокарт на диск по мере того, как их приносили в машинный зал. При окончании выполнения каждого текущего задания операционная система могла загружать новое задание с диска в освободившийся раздел памяти и запускать это задание. Этот технический прием называется **подкачкой данных,** или **спулингом** (spooling — английское слово, которое произошло от Simultaneous Peripheral Operation On Line, то есть совместная периферийная операция в интерактивном режиме), и его также используют для выдачи полученных данных. С появлением подкачки отпала надобность как в 1401-х машинах, так и в многочисленных переносах магнитных лент. Желание сократить время ожидания ответа привело к разработке **режима разделения времени** — варианту многозадачности, при котором у каждого пользователя есть свой диалоговый терминал.

Примеры операционных систем третьего поколения — OS/360, CTSS, MULTICS. Упрощенная однопользовательская версия системы MULTICS позже переросла в операционную систему UNIX®, ставшую популярной в академических кругах, правительственных учреждениях и во многих компаниях.

Четвертое поколение (1980-2005 гг.)

Разработка **БИС (большие интегральные схемы, LSI, Large Scale Integration** — кремниевые микросхемы, содержащие тысячи транзисторов на одном квадратном сантиметре) привела к появлению **микрокомпьютеров**. Главная особенность таких компьютеров — весь процессор помещался в одной микросхеме. Размеры компьютера и его стоимость резко уменьшились, что предоставило возможность купить персональный компьютер каждому человеку. Поэтому со временем их и стали так называть — персональные компьютеры. Операционные системы для микрокомпьютеров принято относить к четвертому поколению. Первой такой операционной системой для микропроцессора Intel 8080 стала CP/M. Для процессора Intel 8086 была разработана операционная система MS-DOS. Первоначально персональные компьютеры предназначались для использования одним пользователем в однопрограммном режиме, что повлекло за собой деградацию архитектуры этих ЭВМ и их операционных систем (в частности, пропала необходимость защиты файлов и памяти, планирования заданий и т. п.). Компьютеры стали использоваться не только специалистами, что потребовало разработки «дружественного» программного обеспечения. Однако рост сложности и разнообразия задач, решаемых на персональных компьютерах, необходимость повышения их надежности их работы привели к возрождению практически всех черт, характерных для архитектуры больших вычислительных систем. Первоначально системы четвертого поколения работали в текстовом режиме, позже появился графический интерфейс. И с этого времени появляется множество операционных систем, имеющих удобный графический интерфейс. Это Mac OS X от Apple, семейство Windows от Microsoft, а также большое количество UNIX-подобных систем. В середине 1980-х годов стали бурно развиваться сети компьютеров, в том числе персональных, работающих под управлением сетевых или распределенных операционных систем. Распределённые системы могут объединять большое число компьютеров, поэтому наиболее важный аспект их дизайна — *масштабируемость* (способность увеличивать производительность пропорционально добавленным ресурсам). Популярность распределённых систем обусловлена быстрым ростом приложений, требующих распределённой обработки.

Пятое поколение (2005 г. - по н.в.)

Хотя идея объединения в одном устройстве и телефона и компьютера вынашивалась еще с 1970-х годов, первый настоящий смартфон появился только в середине 1990-х годов, когда Nokia выпустила свой N9000, представлявший собой комбинацию из двух отдельных устройств: телефона и КПК. В 1997 году в компании Ericsson для ее изделия GS88 «Penelope» был придуман термин «смартфон». Но действительно радикальные изменения произошли в 2005 году с появлением **многоядерных процессоров**. Очевидно, что для использования такого многоядерного процесса нужна многоядерная операционная система, являющаяся расширением одноядерной операционной системы. Популярные ОС Windows и Linux были успешно адаптированы для работы на многопроцессорной (многоядерной) архитектуре. На рынке мобильных устройств между различными операционными системами воцарилась жесткая конкуренция, исход которой еще менее ясен, чем в мире персональных компьютеров. На момент написания этих строк доминирующей была операционная система Google Android, а на втором месте находилась Apple iOS, но в следующие несколько лет ситуация может измениться. В мире смартфонов ясно только одно: долгое время оставаться на вершине какой-либо из операционных систем будет очень нелегко. В первое десятилетие после своего появления большинство смартфонов работало под управлением Symbian OS. Эту операционную систему выбрали такие популярные бренды, как Samsung, Sony Ericsson, Motorola и Nokia. Но долю рынка Symbian начали отбирать другие операционные системы, например, RIM Blackberry OS (выпущенная для смартфонов в 2002 году) и Apple iOS (выпущенная для первого iPhone в 2007 году). Многие ожидали, что RIM будет доминировать на рынке бизнес-устройств, а iOS завоюет рынок потребительских устройств. Для рынка популярность Symbian упала. В 2011 году Nokia отказалась от Symbian и объявила о своем намерении в качестве основной платформы сосредоточиться на Windows Phone. Некоторое время операционные системы от Apple и RIM всех устраивали (хотя и не приобрели таких же доминирующих позиций, какие были в свое время у Symbian), но вскоре всех своих соперников обогнала основанная на ядре Linux операционная система Android, выпущенная компанией Google в 2008 году. Для производителей телефонов Android обладала тем преимуществом, что имела открытый исходный код и была доступна по разрешительной лицензии. В результате компании получили возможность без особого труда подстраивать ее под свое собственное оборудование. Кроме того, у этой операционной системы имеется огромное сообщество разработчиков, создающих приложения в основном на общеизвестном языке программирования Java. Но при всем этом последние годы показали, что такое доминирование может и не продлиться долго и конкуренты Android постараются отвоевать часть ее доли на рынке. Широкомасштабное применение многоядерных процессоров привело к созданию многочисленных компьютеров большой мощности, использующихся для высокопроизводительных вычислений. Обычно такие системы собираются по кластерной технологии, состоят из множества автономных узлов с распределенной памятью, зачастую имеющих несколько процессоров и/или ядер, и отличаются способами соединения отдельных узлов. Широкую популярность приобрела технология **виртуализации** — представление вычислительных ресурсов, абстрагированное от аппаратной реализации. Виртуализации могут быть подвергнуты память системы, устройства ввода-вывода, операционная система, сети передачи данных, сети хранения данных, платформенное и прикладное программное обеспечение. Важным примером использования виртуализации являются облачные технологии, когда потребности клиентов в вычислениях и хранении данных прозрачным образом удовлетворяются специализированными дата-центрами. Обеспечиваемая виртуализацией изолированность позволяет нескольким клиентам одновременно пользоваться общей машиной и другими ресурсами.

Особенности современного этапа развития операционных систем

На современном этапе развития операционных систем на передний план вышли средства обеспечения **безопасности**. Это связано с возросшей ценностью информации, обрабатываемой компьютерами, а также с повышенным уровнем угроз, существующих при передаче данных по сетям, особенно по публичным, таким как Интернет. 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.

# Тема 1.2. Архитектура вычислительных систем

## **3. Компоненты архитектуры вычислительных систем, их назначение и взаимодействие**

Понятие общей шины. Контроллеры. Компоненты компьютера: процессор, память и др.

**Вычислительная система** — взаимосвязанная совокупность аппаратных средств вычислительной техники и программного обеспечения, предназначенная для обработки информации и удобства работы с ней.

**Контроллер** можно рассматривать как специализированный процессор, управляющий работой "вверенного ему" внешнего устройства по специальным встроенным программам обмена. Такой процессор имеет собственную систему команд. Например, контроллер на гибких магнитных дисках (дисковода) умеет позиционировать головку на нужную дорожку диска, читать или записывать сектор, форматировать дорожку и т.п. Результаты выполнения каждой операции заносятся во внутренние регистры памяти контроллера, которые могут быть в дальнейшем прочитаны центральным процессором накопителя. Таким образом, наличие интеллектуальных внешних устройств может существенно изменять идеологию обмена. Центральный процессор, при необходимости произвести обмен, выдает задание на его осуществление контроллеру. Дальнейший обмен информацией может протекать под руководством контроллера без участия центрального процессора. Последний получает возможность "заниматься своим делом", т.е. выполнять программу дальше (если по данной задаче до завершения обмена ничего сделать нельзя, то можно в это время решать другую...).

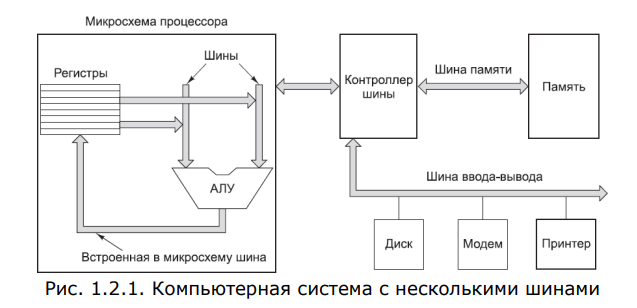
Для связи между отдельными функциональными узлами ЭВМ используется **общая шина** (англ. **bus**, часто ее называют **магистралью**). **Шина** — это несколько проводников, соединяющих несколько устройств. Шины можно разделить на категории в соответствии с выполняемыми функциями. Они могут быть внутренними по отношению к процессору и служить для передачи данных в АЛУ и из АЛУ, а могут быть внешними по отношению к процессору и связывать процессор с памятью или устройствами ввода-вывода. Каждый тип шины обладает определенными свойствами и к каждому из них предъявляются определенные требования.

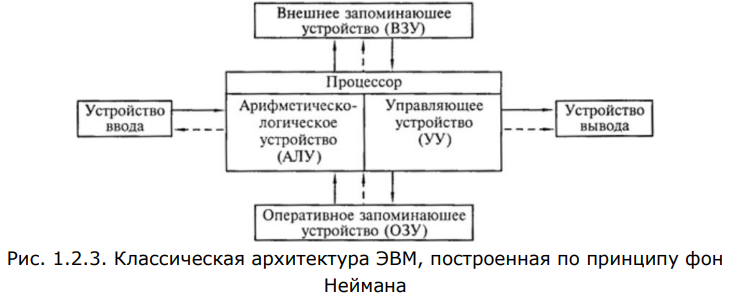
Любая шина состоит из трех частей:

• шина данных, по которой передается информация;

• шина адреса, определяющая, кому передаются данные;

• шина управления, регулирующая процесс обмена информацией. Также может присутствовать шина питания.





Фон-неймановская архитектура

ЭВМ, построенная по принципу фон Неймана, 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).

Центральный процессор

Центральный процессор Центральный процессор 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. Среди специальных регистров процессора стоит выделить счетчик команд IP (Instruction Pointer), который содержит адрес ячейки памяти со следующей выбираемой командой, указатель стека SP (Stack Pointer), который ссылается на вершину текущего стека в памяти и слово состояния программы PSW (Program Status Word).

Память

Второй основной составляющей любого компьютера является память. В идеале память должна быть максимально быстрой (работать быстрее, чем производится выполнение одной инструкции, чтобы работа центрального процессора не замедлялась обращениями к памяти), довольно большой и чрезвычайно дешевой. Никакая современная технология не в состоянии удовлетворить все эти требования, поэтому используется другой подход. Система памяти создается в виде иерархии уровней. 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.

Архитектуры процессоров с различными наборами команд

Аббревиатура CISC обозначает Complex Instruction Set Computer, а RISC — Reduced Instruction Set Computer. Процессоры CISC имеют большой набор разных команд, включая сложные, а длина команды в байтах для наиболее часто использующихся команд минимальна. Это было вызвано в первую очередь экономией оперативной памяти. 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.

Магнитные жесткие диски

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. На современных дисках внешние цилиндры содержат больше секторов, чем внутренние.

Твердотельные накопители

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 большим 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. SLC-ячейки выдерживают до 100000 циклов перезаписи, но хранят в одной ячейке ровно один бит. MLC - 2 бита на ячейку, TLC - 3 бита на ячейку, QLC - 4 бита на ячейку, PLC - 5 битов на ячейку.

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).

## 4. Система прерываний.

Аппаратная поддержка в процессорах x86. Различие работы в реальном и защищенном режимах процессора. Векторы прерываний.

Система прерываний

Для получения услуг от операционной системы пользовательская программа должна осуществить системный вызов, который перехватывается внутри ядра и вызывает операционную систему. Инструкция перехвата TRAP осуществляет переключение из пользовательского режима в режим ядра и запускает операционную систему. Когда обработка вызова будет завершена, управление возвращается пользовательской программе и выполняется команда, которая следует за системным вызовом. Системный вызов следует считать специальной разновидностью инструкции вызова процедуры, у которой есть дополнительное свойство переключения из пользовательского режима в режим ядра.

Все прерывания и особые ситуации имеют уникальные идентификационные номера. Эти номера называются векторами прерываний и лежат в пределах от 0 до 255. Векторы от 0 до 31 отведены для особых ситуаций и немаскируемого прерывания, причем некоторые из них зарезервированы и не должны использоваться программами. Векторы от 32 до 255 свободны для любого использования пользовательскими программами и внешними устройствами.

Классификация 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) Вложенных прерываний нет!

• 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.*

**Маскируемые прерывания** генерируются контроллером прерываний по заявке определенных периферийных устройств. Контроллер прерываний (в первых PC был выполнен в виде специальной микросхемы i8259A) поддерживает восемь уровней (линий) приоритета; к каждому уровню «привязано» одно периферийное устройство. Именно маскируемые прерывания часто называют аппаратными прерываниями. В ПК, начиная с IBM PC AT, построенных на базе микропроцессора i80286, используются два контроллера прерываний i8259A; они соединяются последовательно каскадным образом, что увеличивает количество внешних источников прерываний до 15 (каждая по 8).

**Немаскируемые прерывания** (говорят, что оно одно, т.к. подается на вывод микропроцессора NMI) инициируют источники, требующие безотлагательного вмешательства со стороны микропроцессора.

**Особые ситуации**, генерируемые процессором, подразделяются на три типа — ошибки, ловушки и сбои. В зависимости от типа особой ситуации различается реакция процессора на ее возникновение.(ошибка, ловушка, сбой).

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.

**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.

Фактически, на втором этапе микропроцессор:

• по номеру источника прерывания определяет смещение в таблице векторов прерываний;

• помещает первые два байта в регистр IP;

• помещает вторые два байта в регистр CS;

• передает управление по адресу CS:IP.

Далее выполняется сама программа обработки прерываний

*(Все источники прерывания имеют приоритеты)*

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.

Некоторые наиболее важные прерывания x86:

0 - ошибка деления

1 - прерывание пошагового режима (трассировка)

2 - аппаратное немаскируемое прерывание

3 - прерывание для трассировки (отладки)

8 - прерывание интервального таймера, возникает 18,2 раза в секунду

9 - прерывание от клавиатуры

13 - прерывание от IDE-контроллера

16-31 - обращение к BIOS и т.п.

32-95 - обращение к MS-DOS

Различие реального и защищенного режима и вектора прерываний

В **реальном** режиме для перехода к обслуживанию запроса прерывания или исключения процессор обращается к таблице ***векторов*** прерываний, которая располагается в начальной области памяти, начиная с адреса 0…0h. Элементами таблицы являются вектора прерываний – начальные адреса подпрограмм обслуживания различных типов прерываний. **Вектор** прерывания в реальном режиме содержит четыре байта (два байта база сегмента кода и два байта смещение). Размер всей таблицы может составлять 1024 байт, чтобы обслуживать до 256 возможных видов прерываний.

В **защищенном** режиме вызов подпрограмм обслуживания прерываний осуществляется через дескрипторную таблицу прерываний **IDT (Interrupt Descriptor Table)**, которая может располагаться в любом месте адресного пространства памяти. Базовый линейный адрес этой таблицы хранится в регистре IDTR.

# Тема 1.3. Архитектура операционных систем

## 5. Структура ядра и его функции. Объекты ядра. Основные операции над объектами ядра.

**Ядро** — низкоуровневая основа любой операционной системы, выполняемая аппаратурой в особом привилегированном режиме. Ядро загружается в память один раз и находится в памяти резидентно (постоянно), по одним и тем же адресам.

Возможны следующие типы структуры (архитектуры) ядра операционной системы:

• **Монолитная структура ядра.** Формируется из обширного комплекта абстракций оборудования. Все элементы монолитного ядра работают в едином адресном формате.

**• Модульная структура ядра.** Является современной и модифицированной версией структуры монолитного ядра операционной системы. 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 (элементы) ядра, которые поддерживают нужное аппаратное оборудование (как пример, загрузка драйвера), 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.

Преимущества: Производительность — следствие того, что количество переключений из режима контекста пользователя в режим ядра минимально;

Недостатки: Неустойчивость к сбоям —все базовые компоненты выполняются в режиме ядра, и если хотя бы в одном модуле или блоке ядра произойдет какой-либо сбой, то ему будет подвержена вся операционная система (все ядро), что потребует перезапуска операционной системы.

**• 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.

Преимущества:

• обладают переносимостью (весь машинно-зависимый код изолирован в микроядре, следовательно, нужно мало изменений при переносе системы на новый процессор, к тому же все изменения сгруппированы вместе);

• высокая степень расширяемости (для того, чтобы добавить новую подсистему требуется разработать новое приложение, для чего не требуется затрагивать микроядро; с другой стороны, пользователь легко может удалить ненужные подсистемы, удалять из ядра было бы сложнее);

• достаточная надежность (каждый сервер ОС выполняется в своем адресном пространстве и таким образом защищен от других серверов, кроме того, если происходит крах одного сервера, он может быть перезапущен без останова или повреждения других серверов).

• поддержка распределенных вычислений (серверы ОС могут работать на разных компьютерах - асимметричная ОС; возможен перенос однопроцессорных ОС на распределенные компьютеры).

Основной (и по сути *единственный*) **недостаток** микроядерной архитектуры — снижение производительности из-за накладных расходов.

**• Экзо структура ядра.** Ядро операционной системы, которое предоставляет только возможность взаимного обмена между процессами и надёжного распределения и высвобождения ресурсов. Экзоядро распределяет ресурсы между виртуальными машинами и отслеживает попытки их использования, чтобы ни одна из машин не пыталась использовать чужие ресурсы

**• Нано структура ядра.** Очень простое ядро решает лишь проблему обработки аппаратного прерывания программы, которое генерируют различные блоки компьютера. Когда обработка прерывания, например, при нажатии символа на клавиатуре, завершается, наноядро пересылает результаты программе, которая выше по рангу. При этом пересылка тоже выполняется посредством прерываний

**• Гибридная структура ядра.** Представляет собой модификацию микроядра, которая позволяет ускорить работу системы.

**• Комбинированная структура ядра.**

**Объекты ядра**

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.

Каждый объект ядра на самом деле просто блок памяти, выделенный ядром и доступный только ему. Блок представляет собой структуру данных, в элементах которой содержится информация об объекте. Некоторые элементы (дескриптор защиты, счетчик числа пользователей и др.) присутствуют во всех объектах, но большая их часть специфична для объектов конкретного типа. Например, у объекта процесс может быть идентификатор, базовый приоритет и код завершения, а у объекта файл - смещение в байтах, режим разделения и режим открытия.

Объекты ядра принадлежат ядру, а не процессу. Это говорит о том, что, завершая работу с процессом, мы не обязательно разрушаем объект ядра. В большинстве случаев объект разрушается, но, если созданный вами объект ядра используется другим процессом, ядро запретит разрушение объекта до тех пор, пока от него не откажется последний пользователь.

Основные операции над объектами ядра

Создание новых объектов, или открытие по имени уже существующих, приложение может осуществить при помощи Win32-функций, таких, как 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. Прежде чем прочитать данные из файла, нужно получить описатель. Для этого используется «функция» CreateFile. На самом деле это макрос, который раскрывается в одну из функций: CreateFileA для имен в кодировке ANSI и CreateFileW для имен в кодировке Unicode. 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.

## 6. Утилиты. Системные обрабатывающие программы. Библиотеки процедур. Программы дополнительных услуг.

**Утилиты** — программы, которые решают отдельные задачи управления и сопровождения компьютерной системы (сжатие, дефрагментация …);

*Утилиты* — это системное программное обеспечение, которое помогает поддерживать правильное и бесперебойное функционирование компьютерной системы. Они помогают операционной системе управлять, организовывать, поддерживать и оптимизировать функционирование компьютерной системы.

Утилиты выполняют определенные задачи, такие как обнаружение, установка и удаление вирусов, резервное копирование данных, удаление ненужных файлов и т. д. Некоторыми примерами являются антивирусное программное обеспечение, инструменты управления файлами, инструменты сжатия, инструменты управления дисками и т. д., предоставляют доступ к возможностям (параметрам, настройкам, установкам), недоступным без их применения, либо делают процесс изменения некоторых параметров проще, могут входить в состав операционных систем, идти в комплекте со специализированным оборудованием или распространяться отдельно.

Типы утилит

1. Антивирусы

2. Системы управления файлами

3. Инструменты сжатия

4. Инструменты управления дисками

5. Инструменты очистки диска

6. Дефрагментация диска

7. Утилита резервного копирования

Системные обрабатывающие программы

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.

*Лингвистические процессоры* делятся на ***трансляторы*** и ***интерпретаторы***.

В результате работы **транслятора** алгоритм, записанный на языке программирования (исходная виртуальная программа), преобразуется в алгоритм, записанный на машинном языке. (На самом деле, машинная программа является результатом совместной работы нескольких лингвистических процессоров.)

**Интерпретатор** в отличие от транслятора не выдаёт машинную программу целиком. Выполнив перевод очередного оператора исходной программы в соответствующую совокупность машинных команд, интерпретатор обеспечивает их выполнение. Затем преобразуется тот исходный оператор, который должен выполняться следующим по логике алгоритма, и так далее.

Библиотеки процедур

Библиотеки процедур и функций различного назначения включены в категорию вспомогательных модулей операционной системы. К ним можно отнести библиотеки математических функций, библиотеки функций ввода-вывода и т.д.

**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 операционных системах Windows системные библиотеки хранятся в папках **System32** и **SysWOW64**. Первая папка для 64-хразрядных библиотек, вторая для 32-хразрядных (а не наоборот). В операционных системах на основе ядра Linux по умолчанию библиотеки находятся в двух папках. Это корневая папка **/lib** (в ней находятся библиотеки, которые используют программы, расположенные в корневой папке **/bin**) и вторая папка **/usr/lib** (в ней находятся библиотеки, которые используют программы расположенные **/usr/bin**). Пути к библиотекам указаны в файле **/etc/ld.so.conf**

Программы дополнительных услуг

В эту категорию входит большое количество разнообразных программ: специальный вариант пользовательского интерфейса, калькулятор, некоторые игры (какие, например, поставляются в составе ОС).\**(ахуеть, просто невероятно, самый важный подпункт в вопросе)*

# Тема 2.1. Процессы операционных систем

## 7. Понятие процесса. Системные и пользовательские процессы. Операции над процессами.

**Процесс** — это просто экземпляр выполняемой программы, включая текущие значения счетчика команд, регистров и переменных. Концептуально у каждого процесса есть свой, виртуальный, центральный процессор. Разумеется, на самом деле настоящий центральный процессор постоянно переключается между процессами, но, чтобы понять систему, куда проще думать о наборе процессов, запущенных в (псевдо) параллельном режиме, чем пытаться отслеживать, как центральный процессор переключается между программами. Это постоянное переключение между процессами называется **мультипрограммированием**, или **многозадачным** режимом работы.

Людям довольно трудно отслеживать несколько действий, происходящих параллельно. Поэтому разработчики операционных систем за прошедшие годы создали **концептуальную модель последовательных процессов**, упрощающую работу с параллельными вычислениями. В этой модели все выполняемое на компьютере программное обеспечение, иногда включая операционную систему, сведено к ряду последовательных процессов, или, для краткости, просто процессов.

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. При вызове системных функций (например, **read**(), **write**() или **open**()) пользовательский процесс выполняет инструкции операционной системы. При этом пользовательский процесс «удерживает» процессор до тех пор, пока не будет выполнен системный вызов. Для выполнения *системного вызова* процессор обращается к ядру операционной системы. В это время о пользовательском процессе говорят, что он пребывает в привилегированном режиме, или режиме ядра (kernel mode), и не может быть выгружен никаким другим пользовательским процессом.

**Фоновые** процессы, предназначенные для обработки какой-либо активной деятельности, называются демонами в Unix и службами в Windows.

Операции над процессами



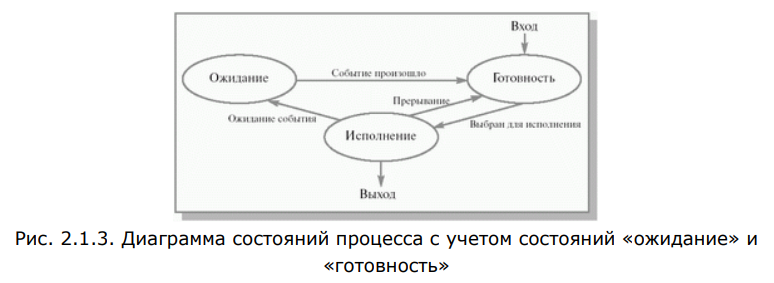
Процесс, находящийся в состоянии *«процесс исполняется»*, может быть либо завершен операционной системой, либо приостановлен и переведен в состояние *«процесс не исполняется»*.

Приостановка процесса происходит по двум причинам:

• для его дальнейшей работы требуется какое-либо событие (например, завершение операции ввода-вывода);

• истек временной интервал, отведенный операционной системой для работы данного процесса.

После приостановки процесса операционная система выбирает следующий процесс для исполнения, который находится в состоянии «процесс не исполняется» и переводит его в состояние «процесс исполняется» Новый процесс, появляющийся в системе, первоначально помещается в состояние «процесс не исполняется». Данная модель не отражает ситуацию ожидания выбранным процессом некоторого события. В процессе ожидания процесс к исполнению не готов.



Процесс не может перейти из одного состояния в другое самостоятельно. Изменением состояния процессов занимается ОС, совершая операции над ними. Удобно объединить их в три пары:

• **создание процесса – завершение процесса;**

**• приостановка процесса – запуск процесса;**

**• блокирование процесса – разблокирование.**

Еще одна операция, не имеющая парной: **изменение приоритета процесса**

Основные операции над процессами

**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**

В действительности же деятельность мультипрограммной операционной системы состоит из цепочек операций, выполняемых над различными процессами, и сопровождается переключением процессора с одного процесса на другой.

## 8. Организация межпроцессного взаимодействия в ОС. Сигналы. Каналы. Классические проблемы межпроцессного взаимодействия.

Организация межпроцессного взаимодействия в ОС

*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;

• удаленный;

• высокоуровневый.

**Средства локального уровня** IPC привязаны к процессору и возможны только в пределах компьютера. К этому виду IPC принадлежат практически все основные механизмы IPC UNIX, а именно, каналы, разделяемая память и очереди сообщений. Коммуникационное пространство этих IPC поддерживаются только в пределах локальной системы. Из-за этих ограничений для них могут реализовываться более простые и более быстрые интерфейсы.

**Удаленные IPC** предоставляют механизмы, которые обеспечивают взаимодействие как в пределах одного процессора, так и между программами на различных процессорах, соединенных через сеть. Сюда относятся удаленные вызовы процедур (Remote Procedure Calls - RPC), сокеты Unix, а также TLI (Transport Layer Interface - интерфейс транспортного уровня) фирмы Sun.

Под **высокоуровневыми IPC** обычно подразумеваются пакеты программного обеспечения, которые реализуют промежуточный слой между системной платформой и приложением.

Сигналы

**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.

*Сигналы могут возникать синхронно с ошибкой в приложении, например SIGFPE (ошибка вычислений с плавающей запятой) и SIGSEGV (ошибка адресации), но большинство сигналов является асинхронными. Сигналы могут посылаться процессу, если система обнаруживает программное событие, например, когда пользователь дает команду прервать или остановить выполнение, или получен сигнал на завершение от другого процесса. Сигналы могут прийти непосредственно от ядра ОС, когда возникает сбой аппаратных средств ЭВМ.*

Система определяет набор сигналов, которые могут быть отправлены процессу. В Linux применяется около 30 различных сигналов. При этом каждый сигнал имеет целочисленное значение и приводит к строго определенным действиям.

Отдельные сигналы подразделяются на три класса:

• системные сигналы (ошибка аппаратуры, системная ошибка и т.д.);

• сигналы от устройств;

• сигналы, определенные пользователем.

Механизм передачи сигналов состоит из следующих частей:

• установление и обозначение сигналов в форме целочисленных значений;

• маркер в строке таблицы процессов для прибывших сигналов;

• таблица с адресами функций, которые определяют реакцию на прибывающие сигналы.

Как только сигнал приходит, он отмечается записью в таблице процессов. Если этот сигнал предназначен для процесса, то по таблице указателей функций в структуре описания процесса выясняется, как нужно реагировать на этот сигнал. При этом номер сигнала служит индексом таблицы.

Известно три варианта реакции на сигналы:

• вызов собственной функции обработки;

• игнорирование сигнала (не работает для SIGKILL);

• использование предварительно установленной функции обработки по умолчанию.

Сигналы посылаются:

• с терминала, нажатием специальных клавиш или комбинаций (например, нажатие Ctrl-C генерирует SIGINT, Ctrl-\ SIGQUIT, а Ctrl-Z SIGTSTP);

• ядром системы:

o при возникновении аппаратных исключений (недопустимых инструкций, нарушениях при обращении в память, системных сбоях и т. п.);

o ошибочных системных вызовах;

o для информирования о событиях ввода-вывода;

• одним процессом другому (или самому себе):

o из С-кода с помощью системного вызова kill():

o из shell, утилитой /bin/kill (она позволяет задавать сигнал как числом, так и символьным обозначением)

Каналы

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).

Для создания канала используется системный вызов **pipe**(). Его формат следующий:

#include <inistd.h>

int pipe (int fd[2]);

После создания канала, процесс может при помощи обычного системного вызова **write**() выводить данные в него, а затем вводить их, вызывая соответственно функцию **read**(). При выполнении вызова **fork**() дескрипторы канала наследуются процессом-"потомком". Таким образом, оба процесса получают возможность обмениваться данными.

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 (race conditions).

**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.

*Если нельзя точно определить, какой из процессов запрашивает или возвращает свои данные в нужный компьютер первым, используется так называемое взаимодействие по модели "клиент-сервер". При этом используются один или несколько клиентов и один сервер. Клиент посылает запрос серверу, а сервер отвечает на него. После этого клиент должен дождаться ответа сервера, чтобы продолжать дальнейшее взаимодействие. Такое поведение называется* ***управлением потоком****. При одновременном доступе здесь также могут использоваться очереди с приоритетами.*

Классический **тупик** возникает, если процесс A получает доступ к файлу A и ждет освобождения файла B. Одновременно процесс B, получив доступ к файлу B, ждет освобождения файла A. Оба процесса теперь ждут освобождения ресурсов другого процесса и не освобождают при этом собственный файл.

# Тема 2.2. Потоки операционных систем

## 9. Концепция потока. Параллельное исполнение потоков. Главный поток процесса.

**Поток** **выполнения** (англ. thread — нить) — 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 (его код) и его *контекст* (значения переменных, которые они имеют в любой момент времени)

*В традиционных операционных системах у каждого процесса есть адресное пространство и единственный поток управления. Фактически это почти что определение процесса. Тем не менее нередко возникают ситуации, когда неплохо было бы иметь в одном и том же адресном пространстве несколько потоков управления, выполняемых квазипараллельно. Во многих приложениях одновременно происходит несколько действий, часть которых может периодически быть заблокированной. Модель программирования упрощается за счет разделения такого приложения на несколько последовательных потоков, выполняемых в квазипараллельном режиме, но разделяющих общее адресное пространство процесса.*

Преимущества потоков

1. Код программы может выполняться, даже если некоторые действия заблокированы.

2. Легкость и быстрота их создания (по сравнению с процессами).

3. Потоки довольно просто обмениваются данными по сравнению с процессами.

4. Улучшает общую производительность системы при ожиданиях завершения операций ввода-вывода.

5. На многопроцессорных системах могут повысить быстродействие за счет параллельных вычислений.

**Недостатки потоков**

1. При программировании приложения с множественными потоками необходимо постоянно думать о потокобезопасности (thread safety).

2. Один неправильно работающий поток может повредить остальные, так как потоки делят общее адресное пространство.

3. 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.

POSIX Threads

Это стандарт POSIX-реализации потоков. Стандарт POSIX.1c, Threads extensions (IEEE Std 1003.1c-1995) определяет API для управления потоками, их синхронизации и планирования. Библиотеки, реализующие этот стандарт (и функции этого стандарта), обычно называются pthreads (функции имеют приставку pthread\_). Некоторые из вызовов функций пакета Pthread: pthread\_create - создание нового потока, pthread\_exit - завершение работы вызвавшего потока, pthread\_join - ожидание выхода из указанного потока, pthread\_yield - освобождение центрального процессора, позволяющее выполняться другому потоку, pthread\_attr\_init - создание и инициализация структуры атрибутов потока, pthread\_attr\_destroy - удаление структуры атрибутов потока.

**Потоки и процессы в Windows**

В среде Microsoft Windows концепция иная, там процесс - это контейнер для потоков. Процесс-контейнер содержит как минимум один поток. Если потоков в процессе несколько, приложение (процесс) становится многопоточным. Процесс - это исполнение программы. Операционная система использует процессы для разделения исполняемых приложений. Поток - это основная единица, которой операционная система выделяет время процессора.

Параллельное исполнение потоков

Под словами *"параллельный поток"* или просто *"поток"* мы будем понимать объект, выполняемый параллельно с основным потоком приложения и с другими параллельными потоками. 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. Момент переключения параллельных потоков и адрес процессорной инструкции в момент переключения будем считать полностью недетерминированными. Реализация параллельных потоков опирается на возможность, предоставляемую операционной системой — "thread". Для именования параллельных потоков в литературе встречаются и другие термины: активные объекты, нити, параллельные процессы.

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

*Модель конкурентного выполнения сопрограмм (****concurrency****) - это немного более широкий термин, чем параллелизм (****parallelism****). Данная модель предполагает, что несколько задач могут выполняться одновременно, но не говорит, как это должно быть достигнуто. В англоязычном мире есть поговорка, "Concurrency does not imply parallelism". Асинхронный ввод-вывод, не являющийся ни многопроцессным, ни многопоточным, тем не менее также подпадает под формулировку конкурентного выполнения кода. Резюмируя вышесказанное, многопроцессность — это форма параллелизма (которого также можно добиться многопоточностью), а параллелизм в свою очередь, это подмножество конкурентного выполнения сопрограмм.*

Виды взаимодействия параллельных потоков

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.

Главный поток процесса

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.

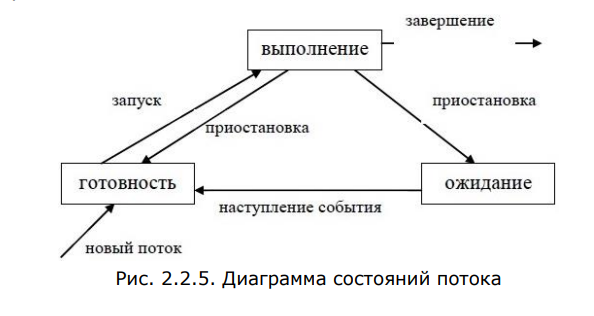
## 10. Диаграммы состояния потоков. Понятие контекста и переключения контекста.

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. Изменение состояния потока происходит в результате соответствующих действий. Удобно для этих целей использовать следующую диаграмму состояний и переходов:



Переходы между состояниями можно описать следующим образом:

• **«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.

*В состояниях готовности и ожидания может находиться несколько потоков, поэтому система создает для хранения их дескрипторов отдельные списковые структуры. Организация этих списков зависит от тех принципов, которые положены в основу планирования потоков для данной ОС.*

Понятие контекста и переключения контекста

**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) (x86) В ядре ОС с каждым потоком связаны следующие структуры:

• Общая информация pid, tid, uid, gid, euid, egid и т. д.

• Состояние процесса/потока

• Права доступа

• Используемые потоком ресурсы и блокировки

• Счетчики использования ресурсов (например, таймеры использованного процессорного времени)

• Регионы памяти, выделенные процессу

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

• Происходит очистка конвейера команд и данных процессора

• Очищается TLB, отвечающий за страничное отображение линейных адресов на физические. Кроме того, следует учесть следующие факты, влияющие на состояние системы:

• Содержимое кэша (особенно это касается кэша первого уровня), накопленное и «оптимизированное» под выполнение одного потока, оказывается совершенно неприменимым к новому потоку, на который происходит переключение.

• При переключении контекста на процесс, который до этого долгое время не использовался, многие страницы могут физически отсутствовать в оперативной памяти, что порождает подкачку вытесненных страниц из вторичной памяти.

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.

# Тема 2.3. Диспетчеризация потоков

## 11. 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. Режим многозадачности позволяет использовать центральный процессор более рационально.

При одновременном присутствии в памяти n процессов вероятность того, что все n процессов ожидают завершения вводавывода (в случае чего процессор простаивает), равна p·n. Тогда время задействования процессора вычисляется по формуле

*Время задействования центрального процессора = 1 – p^n*

Существует два типа многозадачности: **процессная многозадачность** (основанная на процессах — одновременно выполняющихся программах) и **поточная многозадачность** (основанная на потоках).

**Многопоточность** — специализированная форма многозадачности. К псевдопараллельной многозадачности можно отвести следующие типы:

***Простое переключение.*** 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.

***Совместная*** (или ***кооперативная***) многозадачность. Следующая задача выполняется только после того, как текущая задача явно объявит себя готовой отдать процессорное время другим задачам.

***Вытесняющая*** (или ***приоритетная***) многозадачность. 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. В этом виде многозадачности процессор может быть переключен с исполнения одной программы на исполнение другой без всякого пожелания первой программы и буквально между любыми двумя инструкциями в её коде. Распределение процессорного времени осуществляется планировщиком процессов

## 12. Иерархия, приоритеты и планирование потоков. Динамические уровни приоритетов.

**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. 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. (Поток обнуления страниц — это системный поток, отвечающий за обнуление всех свободных страниц, если нет других потоков, которые должны выполняться.)

Приоритет каждого потока определяется следующими критериями:

• Класс приоритета процесса

• Уровень приоритета потока в классе приоритета процесса.

Класс и уровень приоритета объединяются для формирования базового приоритета потока. 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

По умолчанию класс приоритета процесса — NORMAL\_PRIORITY\_CLASS.

Ниже приведены уровни приоритета в каждом классе приоритета.

• THREAD\_PRIORITY\_IDLE

• THREAD\_PRIORITY\_LOWEST

• THREAD\_PRIORITY\_BELOW\_NORMAL

• THREAD\_PRIORITY\_NORMAL

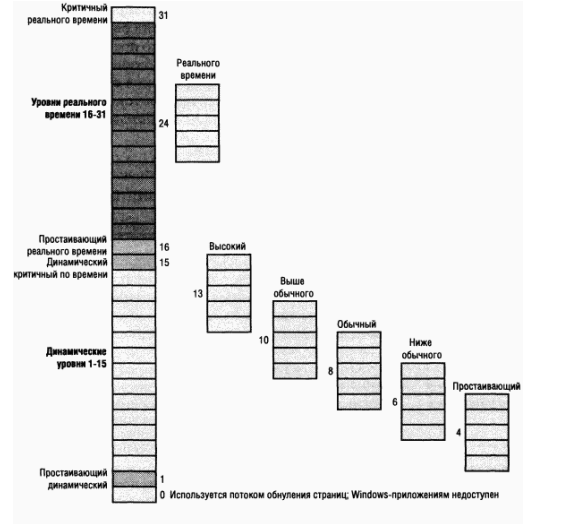
• THREAD\_PRIORITY\_ABOVE\_NORMAL

• THREAD\_PRIORITY\_HIGHEST

• THREAD\_PRIORITY\_TIME\_CRITICAL

Все потоки создаются с уровнем приоритета THREAD\_PRIORITY\_NORMAL. Это означает, что приоритет потока совпадает с классом приоритета процесса.

*Класс приоритета процесса и уровень приоритета потока объединяются для формирования базового приоритета каждого потока.*



Динамические уровни приоритетов в Windows

Каждый поток имеет **динамический** приоритет. Это приоритет, который использует планировщик, чтобы определить, какой поток следует выполнить. Изначально динамический приоритет потока совпадает с базовым приоритетом. Система может повысить и понизить динамический приоритет, чтобы гарантировать, что она реагирует и что потоки не голодают в течение времени процессора.

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.

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.

*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.*

# Тема 2.4. Синхронизация и взаимоблокировка ресурсов

## 13. Синхронизация и взаимоблокировка ресурсов. Механизмы синхронизации.

Состязательная ситуация. Понятия критического ресурса и области. Механизмы синхронизации: критические секции, мьютексы, семафоры, события, барьеры, атомарные операции

Состязательная ситуация

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.

Критическая область(секция)

Критическая область - часть программы, в которой есть обращение к совместно используемым данным.

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.

Чтобы использовать критические секции, сначала нужно объявить переменную типа **CRITICAL\_SECTION.**

Критический ресурс

Критический ресурс - ресурс, к которому осуществляется одновременный доступ.

**Условия для эффективной совместной работы процессов**

1. Два процесса не могут одновременно находиться в своих критических областях.

2. Не должны выстраиваться никакие предположения по поводу скорости или количества центральных процессоров.

3. Никакие процессы, выполняемые за пределами своих критических областей, не могут блокироваться любым другим процессом.

4. Процессы не должны находиться в вечном ожидании входа в свои критические области.

**Синхронизация** - Механизм упорядочивания выполнения программных блоков двух или более потоков.

**Атомарная операция** - Операция, которая не может быть прервана и выполняется как единое целое.

**Особенность синхронизации между процессами**

Синхронизация предполагает наличие общей памяти. В случае потоков проблемы нет, так как потоки разделяют адресное пространство процесса. Но у процессов разобщенные адресные пространства, поэтому нужно найти какое-то решение. 1. Общие структуры хранятся в ядре, для доступа используются системные вызовы. 2. Совместно используемая память. 3 Общий файл. В любом случае несколько процессов, использующих общее адресное пространство, никогда не будут столь же эффективными, как потоки, реализованные на пользовательском уровне, поскольку к управлению процессами неизменно привлекается ядро.

**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.

**Объект семафора** — это объект синхронизации, который поддерживает число от нуля до указанного максимального значения. Число уменьшается каждый раз, когда поток завершает ожидание объекта семафора, и увеличивается каждый раз, когда поток освобождает семафор. Когда число достигает нуля, больше потоки не могут ждать, пока состояние объекта семафора станет сигналом. Состояние семафора становится сигнальным, когда это число становится больше нуля, и несигнальным, когда равно нулю. Объект семафора полезен при управлении общим ресурсом, который может поддерживать ограниченное число пользователей.

Мьютексы

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. Упрощенная версия семафора.

**Объект мьютекса** — это объект синхронизации, состояние которого устанавливается в значение Signaled, если он не принадлежит ни одному потоку, и без знака, когда он принадлежит [3]. Только один поток за раз может владеть объектом мьютекса, имя которого происходит из-за того, что он полезен для координации взаимоисключающего доступа к общему ресурсу. Например, чтобы предотвратить запись двух потоков в общую память одновременно, каждый поток ожидает владения объектом мьютекса перед выполнением кода, который обращается к памяти. После записи в общую память поток освобождает объект мьютекса.

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.

**Барьер синхронизации** позволяет нескольким потокам ждать, пока все потоки не достигнут определенной точки выполнения, прежде чем поток продолжит работу. Барьеры синхронизации не могут быть разделены между процессами. Барьеры синхронизации полезны для поэтапных вычислений, в которых потоки, выполняющие один и тот же код в параллельном режиме, должны завершить один этап, прежде чем переходить к следующему. Чтобы создать барьер синхронизации, вызовите функцию **InitializeSynchronizationBarrier** и укажите максимальное количество потоков и количество раз, когда поток должен вращаться перед блокировкой. Затем запустите потоки, которые будут использовать барьер.

Событие

**Объект события** — это объект синхронизации, состояние которого может быть явно задано с помощью функции **SetEvent**. Ниже приведены два типа объекта события.

**Событие сброса вручную**. Объект события, состояние которого остается сигнальным до тех пор, пока функция **ResetEvent** явно не сбросит событие в несигнальное состояние. 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 несигнальное состояние. Если ожидающих потоков нет, состояние объекта события остается сигнальным. Если ожидается несколько потоков, выбирается ожидающий поток. Не предполагайте порядок "первым в очереди" (FIFO). Внешние события, такие как APC в режиме ядра, могут изменить порядок ожидания.

Объект события полезен при отправке в поток сигнала, указывающего на то, что произошло определенное событие.

## 14. 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.

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. (это рофл?ладно)

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.

# Тема 2.5. Компьютерное время

## 15. Компьютерное время. Ожидаемые таймеры.

Часовые пояса. Локальное время и UTC. Эпоха UNIX. Функции POSIX для работы с временем. Ожидаемые таймеры. Протокол NTP.

Понятие **часовой пояс** имеет два основных значения:

**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. Как правило, в понятие административного часового пояса включается ещё и совпадение даты — в этом случае, например, пояса UTC−10:00 и UTC+14:00 будут считаться различными, хотя в них действует одинаковое время суток.

Локальное время и UTC

**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.

Эпоха UNIX

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

Функции POSIX для работы с временем

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.

В стандарте ANSI C также определяется набор **функций** для манипулирования с календарным временем. Прототипы этих функций определены в заголовочном файле <time.h>.

Функция **time** позволяет получить текущее системное календарное время.

time\_t time(time\_t \*pval);

Функции **localtime**, **gmtime** преобразовывают дату из секундного представления **time\_t** в структурное представление **struct** **tm**. Они имеют следующие прототипы:

struct tm \*gmtime(const time\_t \*timep);

struct tm \*localtime(const time\_t \*timep);

extern char \*tzname[2];

long int timezone;

Функция **mktime** конвертирует календарное время местного часового пояса в развёрнутом структурном формате в секундное представление. Функция имеет следующий прототип:

time\_t mktime(struct tm \*timeptr);

Функции **ctime** и **asctime** конвертируют календарное время в символьную строку. Они имеют следующие прототипы:

char \*asctime(const struct tm \*timeptr);

char \*ctime(const time\_t \*timep);

Функция **ctime** преобразует календарное время timep в строку вида "Tue Nov 7 11:32:11 2000\n"

Таймеры ожидания

**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.

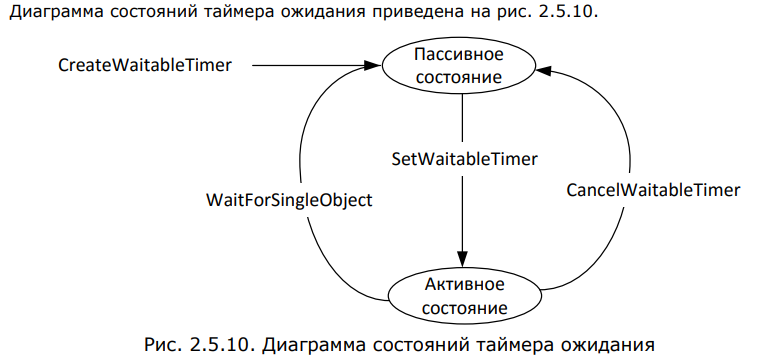
Таймер синхронизации

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.

Поток использует функцию **CreateWaitableTimer** или **CreateWaitableTimerEx** для создания объекта таймера. Поток создания указывает, является ли таймер таймером сброса вручную или таймером синхронизации. Создающий поток может указать имя объекта таймера. Потоки в других процессах могут открывать дескриптор для существующего таймера, указывая его имя в вызове функции **OpenWaitableTimer**. Любой поток с дескриптором объекта таймера может использовать одну из функций ожидания для ожидания, пока состояние таймера будет задано как сигнальное. Поток вызывает функцию **SetWaitableTimer** для активации таймера.



# Тема 3.1. Управление памятью

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

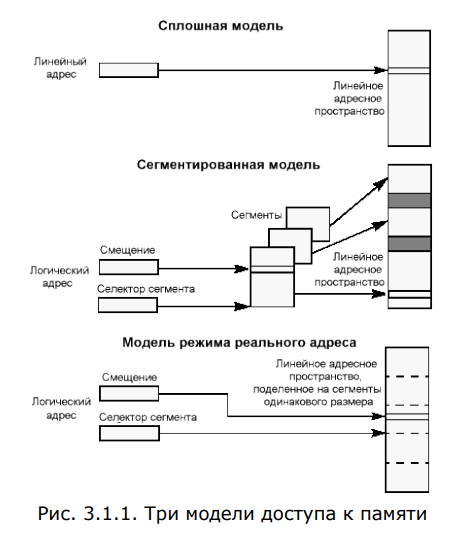
**Адресное** **пространство** — это набор адресов, который может быть использован процессом для обращения к памяти. У каждого процесса имеется собственное адресное пространство, независимое от того адресного пространства, которое принадлежит другим процессам (за исключением тех особых обстоятельств,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





Чтобы в этих моделях запустить другую программу, операционная система должна сохранить все содержимое памяти в файле на диске, а затем загрузить и запустить следующую программу. Эта концепция называется **заменой данных** (или **свопингом**).

Часть операционной системы, которая управляет иерархией памяти (или ее частью), называется *менеджером*, или **диспетчером** **памяти**. Он предназначен для действенного управления памятью и должен следить за тем, какие части памяти используются, выделять память процессам, которые в ней нуждаются, и освобождать память, когда процессы завершат свою работу. ***Физические*** ***адреса*** — это реальные адреса, используемые для выбора микросхем физической памяти, содержащих данные. Физическая память организована в виде последовательности 8-разрядных байтов. Каждому байту присвоен уникальный адрес, который может изменяться от 0 до 232-1 (или 236-1 у P6+). Физическая память составляет единое целое с компьютером и управляется процессором (с некоторой помощью средств прямого доступа к памяти). Организационные методы распределения памяти предоставляют программистам возможность эффективного использования всей компьютерной системы. При использовании средств процессора для управления памятью, программа может использовать одну из трех моделей доступа к памяти: **сплошная** ("плоская") модель памяти, **сегментированная** модель памяти, модель **режима** **реального** **адреса**.



Диспетчер памяти

Блок управления памятью (**MMU**) - это аппаратный компонент компьютера, который обрабатывает все операции с памятью и кэшированием, связанные с процессором. Другими словами, MMU отвечает за все аспекты управления памятью. Обычно он интегрирован в процессор, хотя в некоторых системах он занимает отдельную интегральную схему (ИС).

Работа **MMU** делится на три основные категории:

• Аппаратное управление памятью контролирует и регулирует использование процессором оперативной памяти и кэш-памяти.

• Управление памятью ОС обеспечивает наличие достаточных ресурсов памяти для объектов и структур данных каждой запущенной программы.

• Управление памятью приложений выделяет необходимую память каждой отдельной программе, а затем повторно использует освобожденное пространство памяти после завершения операции.

Управление памятью помогает выполнять следующие функции:

• **Распределение памяти.** MMU перемещает ресурсы памяти туда, где они могут наилучшим образом удовлетворить системные требования, обычно до и после выполнения системных процессов.

• **Мониторинг**. Он отслеживает все ресурсы памяти.

• **Эффективность**. Это сохраняет основную память и другие ресурсы памяти.

• **Целостность** **системы**. Это снижает вероятность фрагментации памяти и плохого распределения памяти.

• **Целостность** **данных**. Это гарантирует сохранение целостности данных.

• **Уменьшение** **рисков** **с** **данными**. Это сводит к минимуму риск повреждения данных.

• **Контроль** **за** **уровнем** **издержек**. Это снижает затраты, связанные с памятью.

• **Управление** **физической** **и** **логической** **памятью**

Основные различия между логической и физической памятью заключаются в следующем:

**Логическая память**. Адреса логической памяти, также называемые виртуальным адресным пространством, создаются ЦП и выполняют множество различных функций. Логическую память также называют виртуальной памятью, поскольку ее размер может динамически изменяться в зависимости от требований обработки данных пользователя.

**Физическая память.** Напротив, MMU вычисляет физические адреса. Затем они загружаются в регистр адреса памяти подсистемы памяти, который также называется физической памятью. Когда физические адреса содержат логические адреса, они называются физическим адресным пространством. В отличие от логических адресов, физические адреса не меняются.

Страничная организация памяти

Виртуальное адресное пространство состоит из блоков фиксированного размера, называемых страницами. Соответствующие блоки в физической памяти называются страничными блоками. Страницы и страничные блоки имеют, как правило, одинаковые размеры. Часто их размер составляет 4 Кбайт.

NX bit

Атрибут (бит) NX-Bit (no execute bit в терминологии фирмы AMD) или XD-Bit (execute disable bit в терминологии фирмы Intel) - бит запрета исполнения, добавленный в страницы для реализации возможности предотвращения выполнения данных как кода. Используется для предотвращения уязвимости типа переполнение буфера, позволяющей выполнять произвольный код на атакуемой системе локально или удалённо. Технология требует программной поддержки со стороны ядра операционной системы.

## 17. Классификация запоминающих устройств. Иерархия памяти. Оперативные и постоянные запоминающие устройства.

Оперативная память

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.

Запоминающее устройство

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 ЗУ

**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);

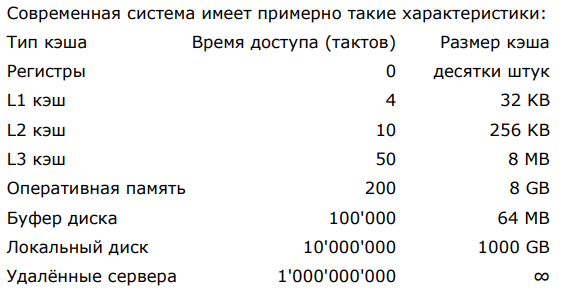
• с магнитной записью (ферритовые сердечники, магнитные диски, магнитные ленты, магнитные карты);

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

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

 Иерархия памяти

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**.



# Тема 3.2. Организация виртуальной памяти

## 18. Виртуальная память. Структуризация адресного пространства виртуальной памяти. Задачи управления виртуальной памятью.

Виртуальная память

В основе виртуальной памяти лежит идея, что у каждой программы имеется собственное адресное пространство, которое разбивается на участки, называемые страницами. Каждая страница представляет собой непрерывный диапазон адресов. Эти страницы отображаются на физическую память, но для запуска программы одновременное присутствие в памяти всех страниц необязательно. Когда программа ссылается на часть своего адресного пространства, находящегося в физической памяти, аппаратное обеспечение осуществляет необходимое отображение на лету. Когда программа ссылается на часть своего адресного пространства, которое не находится в физической памяти, операционная система предупреждается о том, что необходимо получить недостающую часть и повторно выполнить потерпевшую неудачу команду.

*Метод управления памятью процессора, предназначенный для выполнения программ, которым выделяется адресное пространство, превышающее доступный физический объем памяти компьютера.*

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 виртуальной памяти

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

## 19. 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 в Linux

Основная идея подкачки в Linux проста: процессу не обязательно находиться целиком в памяти для того, чтобы выполняться. Все, что нужно, — это пользовательская структура и таблицы страниц. Если они подкачаны в память, то процесс считается находящимся в памяти и может планироваться для выполнения. Страницы сегментов текста, данных и стека подкачиваются динамически (по одной) по мере появления ссылок на них. Если пользовательская структура и таблица страниц не находятся в памяти, то процесс не может выполняться до тех пор, пока процесс подкачки не доставит их в 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.

Подкачка в Windows

Windows поддерживает до 16 файлов подкачки. Обычно они распределены по нескольким дискам (для повышения производительности ввода-вывода). Все файлы имеют некий начальный размер и максимальный размер (до которого они могут увеличиться при необходимости), однако лучше создавать эти файлы максимального размера (при инсталляции системы). Если позднее (когда диски уже будут более заполнены) возникнет необходимость увеличения файла подкачки, то, скорее всего, новое пространство файла подкачки будет иметь высокую степень фрагментации (что снижает производительность).

**hiberfile.sys** - файл для сохранения памяти в режиме «сон» (гибернация);

**pagefile.sys** - файл подкачки;

**swapfile.sys** - файл подкачки отдельных (предварительно скаченных из магазина приложений UWP) для быстрого применения (в случае надобности).

**Гибернация** - энергосберегающее состояние компьютера, предназначенное в первую очередь для ноутбуков. Если в режиме «Сна» данные о состоянии системы и программ хранятся в оперативной памяти, потребляющей энергию, то при гибернации эта информация сохраняется на системном жестком диске в скрытом фале **hiberfil.sys**, после чего ноутбук выключается. При включении, эти данные считываются, и вы можете продолжить работу с компьютером с того момента, на котором закончили.

Алгоритмы замещения страниц

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**. 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. Компьютеры, как и люди, пытаются по возможности максимально отсрочить неприятные события.

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.

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:

1. Класс **0**: в последнее время не было ни обращений, ни модификаций.

2. Класс **1**: обращений в последнее время не было, но страница модифицирована.

3. Класс **2**: в последнее время были обращения, но модификаций не было.

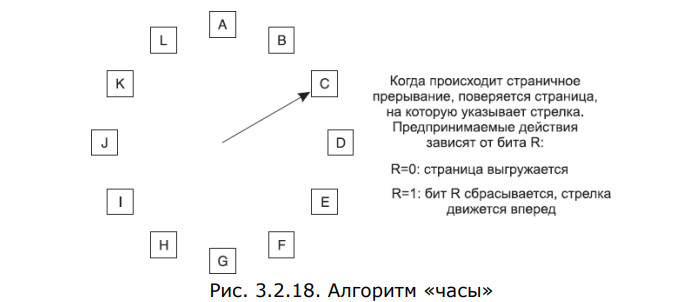
4. Класс **3**: в последнее время были и обращения, и модификации.

**Алгоритм исключения давно использовавшейся страницы.** 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»(FIFO).** 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.

**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 (рис. 3.2.18). Стрелка указывает на самую старую страницу.

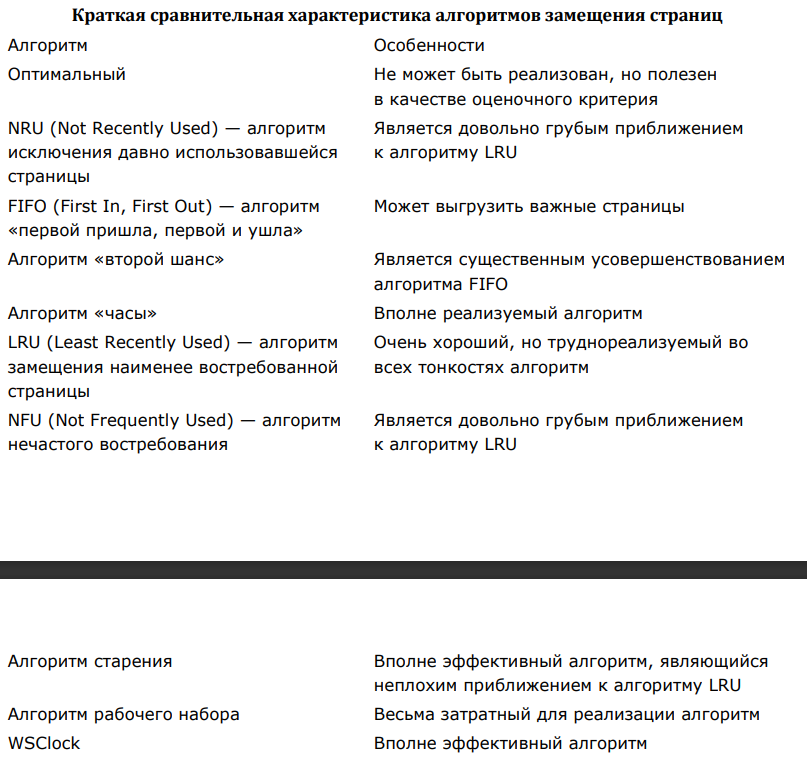


**Алгоритм замещения наименее востребованной страницы.** При возникновении ошибки отсутствия страницы нужно избавиться от той страницы, которая длительное время не была востребована. Эта стратегия называется замещением наименее востребованной страницы (Least Recently Used (LRU)).

**Алгоритм нечастого востребования.** 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.** При возникновении ошибки отсутствия страницы удаляется та страница, чей счетчик имеет наименьшее значение. Очевидно, что в счетчике страницы, к которой не было обращений за, скажем, четыре прерывания от таймера, будет четыре ведущих нуля, и поэтому значение ее счетчика будет меньшим, чем счетчика страницы, к которой не было обращений в течение трех прерываний от таймера.

**Алгоритм «рабочий набор».** Набор страниц, который процесс использует в данный момент, известен как рабочий набор. Для реализации модели рабочего набора необходимо, чтобы операционная система отслеживала, какие именно страницы входят в рабочий набор. Основной замысел алгоритма замещения страниц, основанного на рабочем наборе, состоит в том, чтобы найти страницу, не принадлежащую рабочему набору, и удалить ее из памяти.

****

Куча

**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**().

Управление кучей в Windows

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

**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**).

# Тема 3.3. Управление вводом-выводом и файлами

## 20. 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.

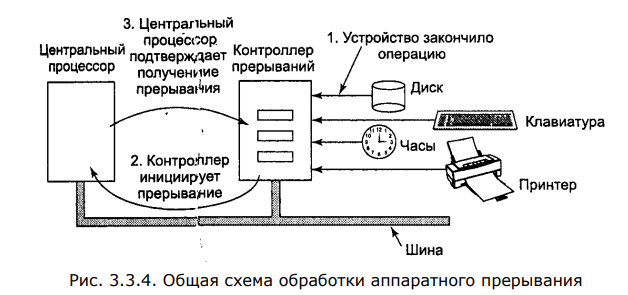
К **блочным** относятся такие устройства, которые хранят информацию в блоках фиксированной длины, у каждого из которых есть собственный адрес. Обычно размеры блоков варьируются от 512 до 65 536 байт. Вся передача данных ведется пакетами из одного или нескольких целых (последовательных) блоков. Важным 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.

Обработка аппаратных прерываний

На аппаратном уровне прерывания работают следующим образом: 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.



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.

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.

Синхронный и асинхронный ввод/вывод

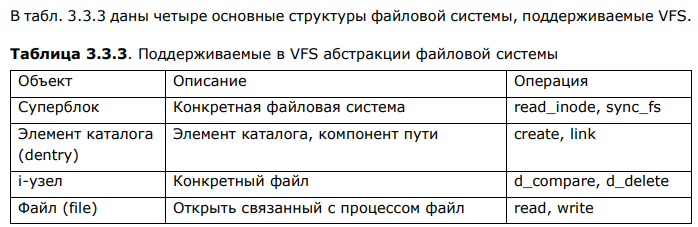
**Асинхронный** ввод-вывод используется там, где можно оптимизировать производительность приложения. При асинхронном вводе-выводе приложение инициирует операцию ввода-вывода, а затем может продолжить свою работу (во время выполнения этого запроса). При **синхронном** вводе-выводе приложение блокируется 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.

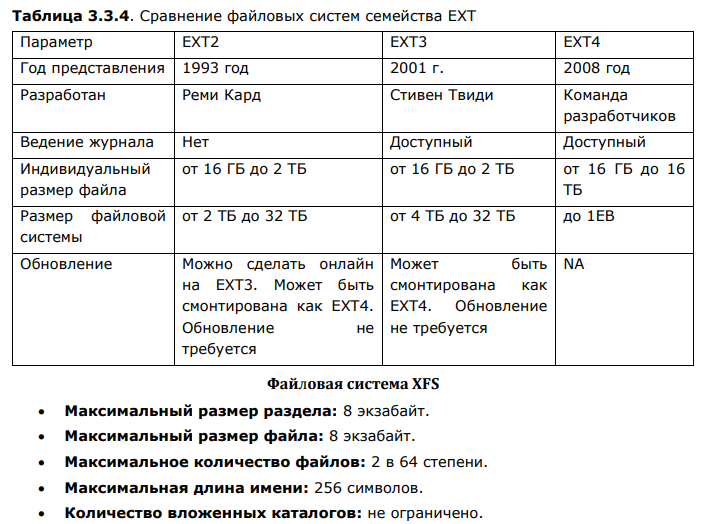
По умолчанию все файловые дескрипторы в Unix-системах создаются в «блокирующем» режиме. Это означает, что системные вызовы для ввода-вывода, такие как **read**, **write** или **connect**, могут заблокировать выполнение программы вплоть до готовности результата операции.

## 21. Файловые системы. Файлы и директории. Управление внешней памятью.

Файловые системы

**Файловая система** (file system) — 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.

****

****

Файловые системы в Windows

Типы файловых систем Windows включают:

Таблица размещения файлов (**FAT**), **FAT32** и расширенная таблица размещения файлов (**exFAT**).

Файловая система NT (**NTFS**).

Устойчивая файловая система (**ReFS**).

Файловая система:

• Предоставляет ряд функций, реализующих хранение и извлечение файлов на устройствах хранения.

• Позволяет организовывать файлы в иерархическую структуру и контролировать их формат и соглашение об именах.

• Поддерживает широкий спектр устройств хранения данных. Все файловые системы, доступные в операционной системе Windows, состоят из следующих компонентов хранения:

• Файлы

• Каталоги

• Объемы

Файлы и директории

**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 NULL.

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, расположенный в каталоге ast, который в свою очередь находится в каталоге usr.

***Абсолютные*** имена путей часто бывают длинными и неудобными. По этой причине операционная система Linux позволяет пользователям и процессам обозначить каталог, в котором они работают в данный момент, как рабочий каталог (working directory). Имена путей могут указываться относительно рабочего каталога. Путь, заданный относительно рабочего каталога, называется ***относительным*** путем (relative path)

*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 ~, то есть, /home/user1/file1, ~/file1 и $HOME/file1 эквивалентны*

Управление внешней памятью (*оч сомнительно описано в лк но тем не менее*)

В основе файловой системы NFS лежит представление о том, что пользоваться общей файловой системой может произвольный набор клиентов и серверов. Во многих случаях все клиенты и серверы располагаются в одной и той же локальной сети, но это не требуется. Файловая система NFS может также работать через Глобальную сеть (если сервер находится далеко от клиента). Для простоты мы будем говорить о клиентах и серверах так, как если бы они работали на различных компьютерах, хотя файловая система NFS позволяет каждой машине одновременно быть и клиентом, и сервером [1]. 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.

# Тема 4.1. Введение в операционную систему Windows

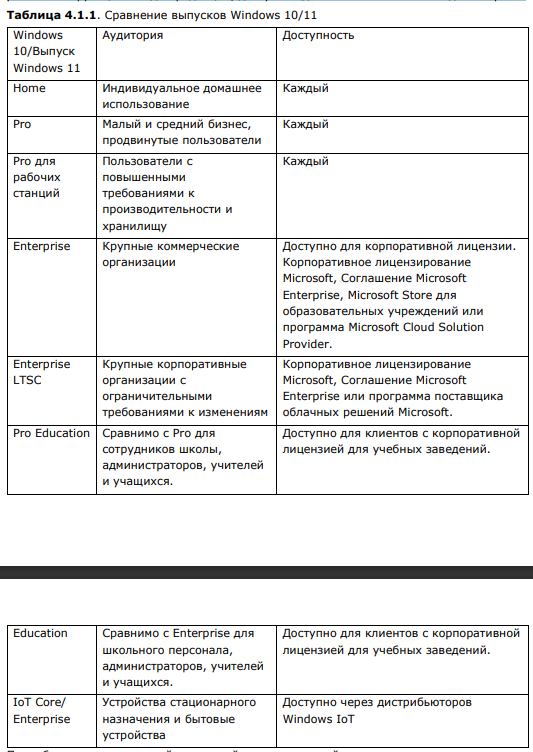
## 22. Принципы организации и структура ОС Windows. Обзор версий Windows. Методы инсталляции ОС Windows.

Принципы организации и структура ОС Windows

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

Обзор версий Windows (*тут какая-то тоска, поэтому даже разбираться не буду*)

Сначала рассмотрим клиентские версии Windows. Прежде чем устанавливать Windows, важно выбрать наиболее подходящую для вашей организации редакцию. Различные выпуски Windows удовлетворяют потребности потребителей, от частных лиц до крупных предприятий. В этом модуле описаны различные функции каждой редакции, доступной для Windows и более поздних версий.



Enterprise LTSC

Enterprise Long Term Service Channel (LTSC) — это специальная версия Enterprise, которую Microsoft не будет обновлять новыми функциями. Enterprise LTSC получает только обновления безопасности и другие важные обновления. Вы можете установить Enterprise LTSC на устройства, работающие в известной неизменяемой среде. Типичным примером может служить ПК, используемый как часть медицинской или промышленной системы. Эти среды обычно представляют собой закрытые системы, разработанные с учетом определенной спецификации, на которую могут повлиять традиционные обновления.

Различия между Enterprise LTSC и стандартной версией Enterprise включают в себя:

• Не получает обновлений функций

• Нет браузера Microsoft Edge (можно установить отдельно)

• Нет клиента Microsoft Store

• Многие встроенные универсальные приложения для Windows отсутствуют.

Серверные версии Windows Server

Последняя LTSC-версия Windows Server сегодня — это Windows Server 2022. Также поддерживается и Windows Server 2019. Windows Server 2022 обеспечивает расширенную многоуровневую безопасность, гибридные возможности с Azure и гибкую платформу приложений . Он имеет встроенные возможности защищенного ядра, помогающие защитить оборудование, встроенное ПО и возможности ОС Windows Server от расширенных угроз безопасности. Серверы с защищенным ядром основаны на таких технологиях, как System Guard в Защитнике Windows и безопасности на основе виртуализации, что позволяет снизить риск, связанный с уязвимостями встроенного ПО и современными вредоносными программами. Соединение также более защищено, чем раньше, благодаря более быстрым и защищенным зашифрованным соединениям HTTPS, стандартному шифрованию SMB AES 256 и ряду других.

Существует три редакции Windows Server 2022:

• Standard

• Datacenter

• Datacenter: Azure Edition

Редакция **Standard** включает основную функциональность, кроме Hotpatching и SDN. Но Storage Replica ограничен одной группой ресурсов до 2TB и количество автоматически лицензируемых машин ограничено двумя.

Редакция **Datacenter** поддерживает все функции, кроме Hotpatching и SMB over QUIC.

Редакция **Datacenter**: **Azure** **Edition** является самым премиальным из трех и, как следует из названия, поддерживается в Azure (Azure IaaS или Azure Stack HCI — 21H2) и предлагает определенные функции, недоступные за пределами этих сред (горячее обновление, SMB через QUIC) и расширенная сеть Azure). Особенно полезна здесь функция горячего исправления (Hotpatching), которая позволяет администраторам применять исправления без перезагрузки, а также SMB через QUIC для безопасного доступа к общим файлам через Интернет без хлопот, связанных с настройкой VPN.

Методы инсталяции ОС Windows

**Ручная установка с использованием установочного носителя**. Устанавливать систему вручную не очень сложно. Нужно воспользоваться установочным носителем и пройти все этапы установки, документируя по мере продвижения все настройки. Этот способ иногда необходим, когда у администратора нет образа, подходящего для определенной аппаратной платформы, или когда регулярно развертывается только небольшое количество систем, а время, необходимое на создание автоматических установок или установок с использованием образов, не ограничено и не является ключевым фактором для организации.

**Автономная (необслуживаемая) установка**. Автономные установки могут быть полезными при развертывании большого количества настольных компьютеров и серверов, на которых установлено одинаковое оборудование. Файл автономной установки представляет собой файл, содержащий ответы на все вопросы, задаваемые во время ручной установки. Конфигурационные файлы автономной установки обычно называют файлами ответов.

**Установка, сопровождаемая производителем.** Некоторые производители поставляют носитель автономной установки, который сразу же после запуска предлагает администратору ответить на несколько вопросов, и оставшаяся часть установки происходит в автономном режиме. Этот распространенный сценарий встречается в секторе розничных продаж для домашних пользователей, а также для серверов и настольных компьютеров, поставляемых с установленными операционными системами.

**Создание копий или образов систем.** Создание копий или образов систем может быть полезно при развертывании нескольких идентичных настольных компьютеров и серверов. Вы создаете настольный компьютер или сервер, подготавливаете систему для клонирования или создания образа и копируете или захватываете образ системы с помощью средств сторонних разработчиков или средств развертывания от Microsoft, таких как WDS. Продукты Microsoft поддерживают клонирование и создание образов серверов и настольных компьютеров только в том случае, если для генерирования идентификаторов безопасности (Security ldentifier — SID) нового компьютера используется утилита Sysprep.ехе.

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

Организация рабочей среды пользователя

Рабочая среда пользователя состоит из настроек рабочего стола, например, цвета экрана, настроек мыши, размера и расположения окон, из настроек процесса обмена информацией по сети и с устройством печати, переменных среды, параметров реестра и набора доступных приложений. Для управления средой пользователя предназначены следующие средства Windows:

**Сценарий входа в сеть** (сценарий регистрации) представляет собой командный файл, имеющий расширение .bat, или исполняемый файл с расширением .ехе, который выполняется при каждой регистрации пользователя в сети. Сценарий может содержать команды операционной системы, предназначенные, например, для создания соединения с сетью или для запуска приложения. Кроме того, с помощью сценария можно устанавливать значения переменных среды, указывающих пути поиска, каталоги для временных файлов и другую подобную информацию.

**Профили пользователей**. В профиле пользователя хранятся все настройки рабочей среды компьютера, на котором работает Windows 2000, определенные самим пользователем. Это могут быть, например, настройки экрана и соединения с сетью.

**Сервер сценариев Windows** (Windows Scripting Host, WSH). Сервер сценариев независим от языка и предназначен для работы на 32-разрядных платформах Windows. Он включает в себя как ядро сценариев Visual Basic Scripting Edition (VBScript), так и JScript. Сервер сценариев Windows предназначен для выполнения сценариев прямо на рабочем столе Windows или на консоли команд. При этом сценарии не надо встраивать в документ НТМL.

Большинство настроек рабочей среды пользователя можно настроить с помощью **групповых политик**.

Работа с учетными записями пользователей и групп

Учетные записи локальных пользователей по умолчанию — это встроенные учетные записи, которые создаются автоматически при установке операционной системы.

Учетные записи локальных пользователей по умолчанию не могут быть удалены или удалены и не предоставляют доступ к сетевым ресурсам.

Учетные записи локальных пользователей по умолчанию используются для управления доступом к ресурсам локального устройства на основе прав и разрешений, назначенных учетной записи. Учетные записи локальных пользователей по умолчанию и создаваемые локальные учетные записи пользователей находятся в папке **Пользователи**. Папка **Пользователи** находится в папке Локальные пользователи и группы в локальной консоли управления Майкрософт (MMC). Управление компьютером — это набор средств администрирования, которые можно использовать для управления локальным или удаленным устройством.

**Администратор**. У каждого компьютера есть учетная запись администратора (SID S-1-5-домен-500, отображаемое имя — Администратор). Учетная запись администратора — это первая учетная запись, которая создается при установке Windows. Учетная запись администратора имеет полный контроль над файлами, каталогами, службами и другими ресурсами на локальном устройстве. Учетная запись администратора может создавать других локальных пользователей, назначать права пользователей и назначать разрешения. Учетная запись администратора может в любое время управлять локальными ресурсами, изменив права и разрешения пользователя. Учетную запись администратора по умолчанию нельзя удалить или заблокировать, но ее можно переименовать или отключить.

Участники групп "Администраторы" могут запускать приложения с повышенными разрешениями, не используя параметр Запуск от имени администратора. Быстрое переключение пользователей более безопасно, чем использование runas или повышение прав пользователей.

**Гость**. Гостевая учетная запись позволяет случайным или разовым пользователям, у которых нет учетной записи на компьютере, временно войти на локальный сервер или клиентский компьютер с ограниченными правами пользователя. По умолчанию учетная запись гостя отключена и имеет пустой пароль. Так как учетная запись гостя может предоставлять анонимный доступ, она считается угрозой безопасности. По этой причине рекомендуется оставить учетную запись гостя отключенной, если только ее использование не требуется.

Управление учетной записью локального пользователя

Учетные записи локальных пользователей по умолчанию и созданные вами учетные записи локальных пользователей находятся в папке «Пользователи». Папка "Пользователи" находится в папке "Локальные пользователи и группы". Вы можете использовать локальные пользователи и группы для назначения прав и разрешений только на локальном сервере, чтобы ограничить возможность локальных пользователей и групп выполнять определенные действия. Право разрешает пользователю выполнять определенные действия на сервере, такие как резервное копирование файлов и папок или выключение сервера. Разрешение на доступ — это правило, связанное с объектом, обычно с файлом, папкой или принтером. Он регулирует, какие пользователи могут иметь доступ к объекту на сервере и каким образом. Вы также можете управлять локальными пользователями с помощью NET.EXE USER и управлять локальными группами с помощью NET.EXE LOCALGROUP или с помощью различных командлетов PowerShell и других технологий сценариев.

Учетные записи для служб

Учетная запись **SYSTEM** используется операционной системой и службами, работающими под управлением Windows.

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

Учетная запись **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

Профили пользователей предоставляют следующие преимущества:

• Когда пользователь входит на компьютер, система использует те же параметры, которые использовались при последнем выходе пользователя.

• При совместном использовании компьютера с другими пользователями каждый пользователь получает настроенный рабочий стол после входа в систему.

• Параметры в профиле пользователя уникальны для каждого пользователя. Другие пользователи не могут получить доступ к параметрам. Изменения, внесенные в профиль одного пользователя, не влияют на профили других пользователей или других пользователей.

Локальные профили пользователей

Система автоматически создает локальный профиль пользователя для каждого пользователя при первом входе пользователя на компьютер. Система автоматически поддерживает параметры рабочей среды каждого пользователя в профиле пользователя на локальном компьютере. Управление профилями пользователей в Windows Vista и более поздних версиях осуществляется с помощью элемента панели управления ***Учетные записи пользователей***.

Профиль пользователя не загружается автоматически при входе пользователя с помощью функции **LogonUser**. Чтобы загрузить профиль пользователя программным способом, используйте функцию **LoadUserProfile**. Чтобы выгрузить профиль пользователя, загруженный **LoadUserProfile**, вызовите функцию **UnloadUserProfile**.

Перемещаемые профили пользователей

Профили, которые хранятся на сервере. Перемещаемые профили пользователей имеют следующие преимущества.

• Автоматическая доступность ресурсов. Уникальный профиль пользователя автоматически становится доступным при входе на любой компьютер в сети. Пользователям не нужно создавать профиль на каждом компьютере, который они используют в сети.

• Упрощенная замена и резервное копирование компьютеров. Когда необходимо заменить компьютер пользователя, его можно легко заменить, так как все данные профиля пользователя хранятся отдельно в сети, независимо от отдельного компьютера. Когда пользователь впервые входит на новый компьютер, серверная копия профиля пользователя копируется на новый компьютер.

Профиль пользователя не загружается автоматически при входе пользователя с помощью функции **LogonUser**. Аналогия с локальными профилями.

Обязательный профиль пользователя

Обязательный профиль пользователя — это особый тип предварительно настроенного перемещаемого профиля пользователя, который администраторы могут использовать для указания параметров для пользователей. С помощью обязательных профилей пользователей пользователь может изменять свой рабочий стол, но изменения не сохраняются при выходе пользователя из системы. При следующем входе пользователя будет загружен обязательный профиль пользователя, созданный администратором. Существует два типа обязательных профилей: обычные обязательные профили и супер-обязательные профили. Профили пользователей становятся обязательными, когда администратор переименовывает файл NTuser.dat (куст реестра) на сервере в NTuser.man. Расширение .man приводит к тому, что профиль пользователя будет доступен только для чтения. Профили пользователей становятся супер обязательными, если имя папки пути к профилю заканчивается на .man; например, \\server\share\mandatoryprofile.man\. Суперобязательные профили пользователей похожи на обычные обязательные профили, за исключением того, что пользователи с супер-обязательными профилями не могут войти в систему, если сервер, на котором хранится обязательный профиль, недоступен.

# Тема 4.2. Администрирование в ОС Windows

## 24. Системный реестр ОС Windows, его назначение и использование.

Системный реестр 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,

HKEY\_CURRENT\_CONFIG.

HKEY\_CLASSES\_ROOT

Этот куст содержит информацию о ассоциации файлов и определяет, какое приложение открывается, когда пользователь дважды щелкает определенный тип файла в файловой системе. Например, он определяет, что приложением для файлов .xlsx является Microsoft Excel. Этот куст заполняется на основе настроек, связанных с компьютером и пользователем, которые хранятся в разделах HKEY\_LOCAL\_MACHINE\Software\Classes и HKEY\_CURRENT\_USER\Software\Classes. Обычно вы не будете вносить изменения в этот куст.

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.

Ключи и подразделы

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, как показано в следующем примере:

• HKCU\Control Panel\Desktop\Wallpaper — это значение (Wallpaper), в котором хранится имя и расположение обоев рабочего стола пользователя.

• HKLM\Software\Microsoft\Windows\CurrentVersion\Run — это ключ, содержащий значения, относящиеся к программам, которые запускаются автоматически при запуске компьютера и входе пользователя в систему. Обычно эти программы находятся на панели задач.

Значения

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.

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

Списки управления доступом (ACL)

Списки управления доступом (ACL) состоят из записей управления доступом (ACE). Каждый ACE в ACL идентифицирует доверенного лица и указывает права доступа, разрешенные, запрещенные или регистрируемые для этого доверенного лица.

Дескриптор безопасности для защищаемого объекта может содержать два типа списков управления доступом: **DACL** и **SACL**.

**Список управления доступом на уровне пользователей** (**DACL**) определяет доверенных лиц, которым разрешен или запрещен доступ к защищаемому объекту. Когда процесс пытается получить доступ к защищаемому объекту, система проверяет ACE в DACL объекта, чтобы определить, следует ли предоставить доступ к нему. Если у объекта нет DACL, система предоставляет полный доступ всем пользователям. Если DACL объекта не имеет ACE, система отклоняет все попытки доступа к объекту, так как DACL не разрешает какие-либо права доступа. Система проверяет ACE в последовательности, пока не найдет один или несколько ACE, которые разрешают все запрошенные права доступа, или пока не будет отказано в любом из запрошенных прав доступа.

**Системный список управления доступом (SACL)** позволяет администраторам регистрировать попытки доступа к защищенному объекту. Каждый ACE указывает типы попыток доступа со стороны указанного доверенного лица, которые приводят к созданию системой записи в журнале событий безопасности. ACE в SACL может создавать записи аудита при сбое попытки доступа и/или при успешном выполнении. Предоставляется несколько функций, которые извлекают сведения об управлении доступом из списка управления доступом (ACL). К ним относятся функции для определения прав доступа, которые ACL предоставляет или проверяет для указанного доверенного лица.

## 26. Средства автоматической настройки в ОС Windows.

BAT-файлы. Powershell. WMI. ETW. Групповые политики. Административные шаблоны.

BAT-файлы

Текстовые файлы с расширением .cmd или .bat (от слова batch) Комментарии начинаются с REM (remark) Строки, начинающиеся с @, не выводятся на экран

Допускаются программные конструкции, например:

for %%i in (1 1 5) do echo %%i

if a equ 0 echo 0 else echo notzero

Для вызова другого скрипта используется команда call

Для создания переменной используется команда set set a=0

Оболочки командной строки Windows

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. Командная оболочка cmd была первой оболочкой, встроенной в Windows, для автоматизации повседневных задач, с пакетными (.bat или .cmd) файлами. С помощью Windows Script Host можно запускать более сложные скрипты в командной оболочке (команды cscript и vscript). 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.

*Примеры команд командной строки: dir – вывод содержимого каталога copy – копирование файла или папки ren – переименование файла или папки cd – сменить текущий каталог mkdir – создать подкаталог*

WMI

Инструментарий управления Windows (WMI) — это инфраструктура для управления данными и операциями в операционных системах Windows. Это расширенная и адаптированная под Windows реализация стандарта WBEM, принятого многими компаниями, в основе которого лежит идея создания универсального интерфейса мониторинга и управления различными системами и компонентами распределённой информационной среды предприятия с использованием объектно-ориентированных идеологий и протоколов HTML и XML. В основе структуры данных в WBEM лежит Common Information Model (CIM), реализующая ООП-подход к представлению компонентов системы. CIM является расширяемой моделью, что позволяет программам, системам и драйверам добавлять в неё свои классы, объекты, методы и свойства. WMI, основанный на CIM, также является открытой унифицированной системой интерфейсов доступа к любым параметрам операционной системы, устройствам и приложениям.

ETW

Event Tracing for Windows (ETW) — это служба, которая позволяет получать события от одного или нескольких поставщиков событий в режиме реального времени или из файла \*.etl за некоторый временной период. Архитектура ETW включает в себя 4 элемента:

1. **поставщики событий (providers),**
2. **потребители событий (consumers),**
3. **контроллеры ETW (controllers),**
4. **сессии ETW (event tracing sessions).**

**Поставщики событий** — это приложения, содержащие инструменты отслеживания событий. **Контроллер** — это приложение, которое отвечает за функционирование одной или нескольких сессий ETW. **Потребители** — это приложения, которые получают и обрабатывают события от одного или нескольких сеансов трассировки одновременно. Сеансы отслеживания событий (**сессии ETW**) записывают события от одного или нескольких провайдеров, которые разрешает контроллер. Сессия также отвечает за управление и очистку буферов.

Групповые политики

**Групповая** **политика** — важный элемент любой среды Microsoft Active Directory (AD). Её основная цель — дать ИТ-администраторам возможность централизованно управлять пользователями и компьютерами в домене . Групповая политика, в свою очередь, состоит из набора политик, называемых объектами групповой политики (GPO).

При создании домена AD автоматически создаются два объекта групповой политики:

• политика домена по умолчанию устанавливает базовые параметры для всех пользователей и компьютеров в домене в трех плоскостях: политика паролей, политика блокировки учетных записей и политика Kerberos;

• политика контроллеров домена по умолчанию устанавливает базовые параметры безопасности и аудита для всех контроллеров домена в рамках домена.

Применение групповой политики

Для вступления настроек в силу, объект групповой политики необходимо применить (связать) с одним или несколькими контейнерами Active Directory: сайт, домен или подразделение (OU). Например, можно использовать групповую политику, чтобы потребовать от всех пользователей в определённом домене использовать более сложные пароли или запретить использование съемных носителей на всех компьютерах только в финансовом подразделении данного домена. Объект групповой политики не действует, пока не будет связан с контейнером Active Directory, например, сайтом, доменом или подразделением. Любой объект групповой политики может быть связан с несколькими контейнерами, и, наоборот, с конкретным контейнером может быть связано несколько объектов групповой политики. Кроме того, контейнеры наследуют объекты групповой политики, например, объект групповой политики, связанный с подразделением, применяется ко всем пользователям и компьютерам в его дочерних подразделениях. Аналогичным образом, объект групповой политики, применяемый к OU, применяется не только ко всем пользователям и компьютерам в этом OU, но и наследуется всем пользователям и компьютерам в дочерних OU.

**Порядок применения групповой политики**

Настройки различных объектов групповой политики могут перекрываться или конфликтовать. По умолчанию объекты групповой политики обрабатываются в следующем порядке (причем примененные позднее имеют приоритет над примененными ранее): Локальный (индивидуальный компьютер), Сайт, Домен, Организационное подразделение. В эту последовательность можно и нужно вмешиваться, выполнив любое из следующих действий. Изменение последовательности GPO. Объект групповой политики, созданный позднее, обрабатывается последним и имеет наивысший приоритет, перезаписывая настройки в созданных ранее объектах. Это работает в случае возникновения конфликтов. Блокирование наследования. По умолчанию дочерние объекты наследуют все объекты групповой политики от родительского, но вы можете заблокировать это наследование. Принудительное игнорирование связи GPO. По умолчанию параметры родительских политик перезаписываются любыми конфликтующими политиками дочерних объектов. Вы можете переопределить это поведение. Отключение связей GPO. По умолчанию, обработка включена для всех связей GPO. Вы можете предотвратить применение объекта групповой политики для конкретного контейнера, отключив связь с объектом групповой политики этого контейнера.

**Административные шаблоны**

Административные шаблоны представляют собой XML-файлы на разных языках, которые определяют основанные на значениях реестра параметры групповых политик, отображаемые в редакторе локальных групповых политик. Существует два вида административных шаблонов: **ADMX** — не зависящий от языка файл установки, который указывает количество и тип параметров политики, а также расположение по категориям согласно отображению файла в редакторе локальных групповых политик, **ADML** — файл установки на определенном языке, который предоставляет связанные с языком сведения для ADMX-файла. Этот файл позволяет параметру политики отображаться в редакторе локальных групповых политик на нужном языке. Вы можете добавлять новые языки, добавляя новые ADML-файлы на нужных языках.

# Тема 4.3. UNIX-подобные и другие POSIX-совместимые операционные системы

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

Основные понятия системы UNIX.

Основное отличие Unix-подобных систем от других операционных систем заключается в том, что это изначально многопользовательские **многозадачные** системы. То есть в один и тот же момент времени сразу множество людей может выполнять множество вычислительных задач (процессов). Вторая колоссальная заслуга Unix в её **мультиплатформенности**. Ядро системы написано таким образом, что его легко можно приспособить практически под любой микропроцессор.

Unix имеет и другие характерные особенности:

• использование простых текстовых файлов для настройки и управления системой;

• широкое применение утилит, запускаемых из командной строки;

• взаимодействие с пользователем посредством виртуального устройства — терминала;

• представление физических и виртуальных устройств и некоторых средств межпроцессного взаимодействия в виде файлов;

• использование конвейеров из нескольких программ, каждая из которых выполняет одну задачу.

Существует два основных объекта операционной системы UNIX, с которыми приходится работать пользователю – файлы и процессы. Эти объекты сильно связаны друг с другом, и в целом организация работы с ними как раз и определяет архитектуру операционной системы. Все данные пользователя хранятся в файлах; доступ к периферийным устройствам осуществляется посредством чтения и записи специальных файлов; во время выполнения программы, операционная система считывает исполняемый код из файла в память и передает ему управление. С другой стороны, вся функциональность операционная определяется выполнением соответствующих процессов. В частности, обращение к файлам на диске невозможно, если файловая подсистема операционной системы (совокупность процессов, осуществляющих доступ к файлам) не имеет необходимого для этого кода в памяти. В операционной системе содержится большое количество стандартных программ, некоторые из них указаны в стандарте POSIX 1003.2, тогда как другие могут различаться от версии к версии. К этим программам относятся командный процессор (оболочка), компиляторы, редакторы, программы обработки текста и утилиты для работы с файлами. Именно эти программы и запускает пользователь с клавиатуры. Графические интерфейсы пользователя поддерживает оконная система X Windowing System, которую обычно называют Х11 (или просто Х). Она определяет обмен и протоколы отображения для управления окнами на растровых дисплеях UNIX-подобных систем.

Пользователи системы

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). Каждый пользователь имеет уникальное или регистрационное имя, но система различает пользователей по идентификатору пользователя UID. Идентификаторы также должны быть уникальны. Пользователи являются членами одной или нескольких групп. Информация о пользователях обычно хранится в специальном файле: /etc/passwd. Этот файл доступен только для чтения. Писать в него может только администратор.

Атрибуты пользователя

name

Регистрационное имя пользователя. Это имя пользователь вводит в ответ на приглашение системы login. Для небольших систем имя пользователя достаточно произвольно. В больших системах, в которых зарегистрированы сотни пользователей, требования уникальности заставляют применять определенные правила выбора имен.

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. В случае совпадения пользователю разрешается войти в систему. Даже в закодированном виде доступность пароля представляет некоторую угрозу для безопасности системы. 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. Обычно такой пароль имеют псевдопользователи.

**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 (например, при запуске программы или чтении файла). Суперпользователь имеет UID=0, что дает ему неограниченные права в системе.

**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. Группа определяет общие для всех членов права доступа и тем самым обеспечивает возможность совместной работы (например, совместного использования файлов).

**comments**

Обычно, это полное "реальное" имя пользователя. Это поле может содержать дополнительную информацию, например, телефон или адрес электронной почты. Некоторые программы (например, finger(1) и почтовые системы) используют это поле.

**home-dir**

Домашний каталог пользователя. При входе в систему пользователь оказывается в этом каталоге. Как правило, пользователь имеет ограниченные права в других частях файловой системы, но домашний каталог и его подкаталоги определяют область файловой системы, где он является полноправным хозяином.

**shell**

Имя программы, которую UNIX использует в качестве командного интерпретатора. При входе пользователя в систему UNIX автоматически запустит указанную программу. Обычно это один из стандартных командных интерпретаторов /bin/sh (Bourne shell), /bin/csh (С shell) или /bin/ksh (Korn shell), позволяющих пользователю вводить команды и запускать задачи. В принципе, в этом поле может быть указана любая программа, например, командный интерпретатор с ограниченными функциями (restricted shell), клиент системы управления базой данных или даже редактор. Важно то, что, завершив выполнение этой задачи, пользователь автоматически выйдет из системы. Некоторые системы имеют файл /etc/shells, содержащий список программ, которые могут быть использованы в качестве командного интерпретатора.

Файловая структура ОС

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. В этом же месте можно обнаружить все другие дополняющие модули ядра, следовательно, данный файл в плане размерности может быть ощутимо больше самого ядра Линукс.

В файле **system.map** не составит труда обнаружить различные функций. Программы устанавливаются в различные директории в файловой системе.

Одна из наиболее распространенных директорий для установки программ — это **/usr/bin**. Здесь обычно размещаются исполняемые файлы программ, доступные для всех пользователей системы. Эта директория имеет преимущество перед другими в пути поиска исполняемых файлов, что позволяет пользователю запускать программы из любой директории.

Другая важная директория для установки программ — это **/opt**. Она служит для установки приложений, которые не являются частью основной операционной системы. Здесь можно разместить программы, которые вы загрузили из Интернета или с которыми не связаны зависимости от других программ. Установка программ в эту директорию обеспечивает лучшую изоляцию и восстановление системы после обновления или удаления программ.

Наконец, важно упомянуть о **/usr/local** — директории, в которой обычно размещаются локально установленные программы. Здесь могут находиться программы, установленные самим пользователем или администратором системы и которые не входят в стандартный пакет установки Linux. Размещение программ в этой директории позволяет отделить их от программ, поставляемых с операционной системой, и облегчает их управление и обновление.

## 28. ОС UNIX: особенности процессов, сигналы, обработка сигналов.

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). Linux представляет собой многозадачную систему и 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.

На большинстве однопользовательских рабочих станций работают десятки фоновых процессов, называемых **демонами** (daemons). Они запускаются при загрузке системы из сценария оболочки.

Типичным демоном является **cron**. Он просыпается раз в минуту, проверяя, не нужно ли ему что-то сделать. Если у него есть работа, он ее выполняет, а затем отправляется спать дальше (до следующей проверки). Этот демон позволяет планировать в системе Linux активность на минуты, часы, дни и даже месяцы вперед.

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**.

Сигналы

Сигналы — это способ информирования процесса со стороны ядра о происшествии некоторого события. 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**.

Обработка сигналов

Процессы могут сообщить системе, какие действия следует предпринимать, когда придет входящий сигнал. Варианты такие:

1. проигнорировать сигнал,
2. перехватить его,
3. 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). Процесс может посылать сигналы только членам своей группы процессов (process group), состоящей из его прямого родителя (и других предков), братьев и сестер, а также детей (и прочих потомков). Процесс может также послать сигнал сразу всей своей группе за один системный вызов.*

# Тема 4.4. Операционная система ОС Linux

## 29. Основные принципы функционирования Linux. Основные компоненты Linux. Дистрибутивы Linux. Файловая система Linux.

Основные принципы функционирования 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.

Основные компоненты Linux

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.

Ядро

В ядре собрана основная функциональность для работы с памятью, управления процессами и т.д. Когда ядру требуется дополнительные функции, он обращается к модулям. В модулях, например, может содержаться код для работы с оборудованием. Файлы ядра Linux находятся в /boot.

**Планировщик**

Так как ядро Linux обеспечивает одновременную работу нескольких процессов от нескольких пользователей (с поддержкой нескольких процессоров), операционная система нуждается в средствах управления многопоточностью. Планировщик Linux назначает процессам приоритеты и определяет, какой процесс выполняется на конкретном процессоре (если в системе установлено несколько процессоров). Планировщик можно настроить для работы в системах различного типа. При правильной настройке более важные процессы получают более быструю реакцию процессора. Например, планировщик Linux на настольном компьютере предоставляет больший приоритет задаче перемещения окна и меньший — задаче фонового копирования файлов.

**Файл подкачки**

Ядро Linux старается держать работающие в данный момент процессы в оперативной памяти. Простаивающие процессы перемещаются в файл подкачки, представляющий собой выделенную область на жестком диске, которая используется для хранения не перемещающихся в оперативную память данных и процессов. При переполнении оперативной памяти процессы выносятся в файл подкачки. При переполнении файла подкачки (но этого допускать нельзя) новые процессы не запускаются.

**Модули**

Ядро Linux поддерживает тысячи аппаратных устройств. При этом за счет включения в работающее ядро только актуальных драйверов размер ядра удается сохранять на приемлемом уровне. Использование загружаемых модулей позволяет добавить в ядро поддержку дополнительных устройств. Модули можно загружать и выгружать по запросу в результате подключения или отключения устройства.

**Файловые системы**

Как и UNIX, операционная система Linux изначально создавалась для обеспечения, одновременного многопользовательского доступа. Для защиты пользовательских ресурсов каждому файлу назначаются наборы разрешений на чтение, запись и выполнение, которые определяют права доступа. В стандартной системе Linux пользователь root имеет доступ ко всей системе без ограничений, специальные регистрационные записи могут управлять определенными службами (например, службами Web-сервера Apache), а пользователям могут присваиваться разрешения по отдельности или в составе групп. Последние нововведения, например, Security-Enhanced Linux (SELinux), поддерживают более тонкую настройку и защиту безопасных сред обработки информации.

**Механизмы защиты**

Как и UNIX, операционная система Linux изначально создавалась для обеспечения, одновременного многопользовательского доступа. Для защиты пользовательских ресурсов каждому файлу назначаются наборы разрешений на чтение, запись и выполнение, которые определяют права доступа. В стандартной системе Linux пользователь root имеет доступ ко всей системе без ограничений, специальные регистрационные записи могут управлять определенными службами (например, службами Web-сервера Apache), а пользователям могут присваиваться разрешения по отдельности или в составе групп. Последние нововведения, например, Security-Enhanced Linux (SELinux), поддерживают более тонкую настройку и защиту безопасных сред обработки информации.

**Инструменты администрирования**

Инструменты администрирования включают в себя сотни (а возможно и тысячи) команд и графических утилит, которые позволяют добавлять пользователей, управлять дисками, следить за состоянием сети, устанавливать программное обеспечение, а также гарантировать безопасность и управлять ресурсами компьютера.

**Серверные возможности**

Серверные возможности позволяют компьютеру под управлением Linux предоставлять службы для клиентов в сети. Иными словами, кроме установки Web-обозревателей для просмотра Web-страниц, компьютер можно превратить в сервер, который предоставляет Web-страницы другим компьютерам. При этом среди популярных серверных функций можно назвать Web-серверы, серверы электронной почты, баз данных, печати, файловые серверы, серверы DNS и DHCP.

**Графический интерфейс пользователя**

Графический интерфейс пользователя состоит из графической инфраструктуры (обычно это X WindowSystem), оконных менеджеров, панелей, пиктограмм и меню. Графический интерфейс пользователя позволяет применять комбинацию мыши и клавиатуры вместо простого ввода команд с клавиатуры.

**Система управления пакетами**

Система управления пакетами - это набор инструментов, предназначенных для автоматизации процессов установки, обновления, конфигурирования и удаления пакетов программного обеспечения определенного формата. Наиболее известными (или распространенными) системами управления пакетами являются: RPM/YUM — менеджер пакетов Red Hat; dpkg/APT — система управления пакетами \*.deb дистрибутива Debian; tgz или tar.gz — стандартный набор из двух программ tar + gzip; система портежей дистрибутива Gentoo; YaST - утилита, разработанная Novell и используемая в дистрибутиве SuSE.

**Дистрибутивы Linux**

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.

**Классификация дистрибутивов Linux**

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.

**Файловая система Linux**

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**

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.

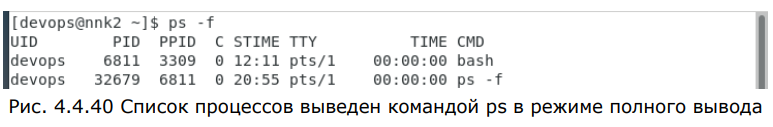
FHS (Filesystem Hierarchy Standard)

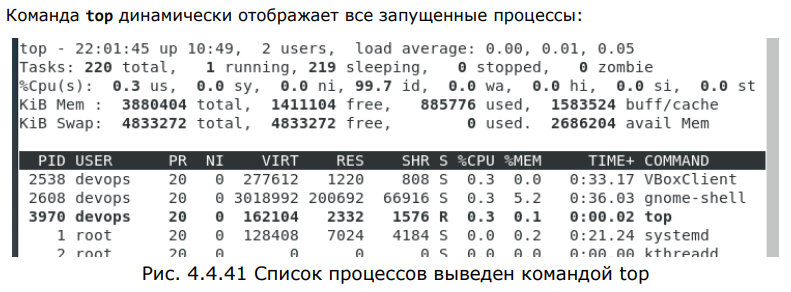
FHS (Filesystem Hierarchy Standard) – стандарт файловой системы, специфичной для GNU/Linux, создан в 1994-1996 гг. Текущая версия стандарта FHS 3.0 выпущена 03.06.2015 г. Некоторые Linux-системы отвергают FHS и следуют своему собственному стандарту.В FHS все файлы и каталоги находятся внутри корневого 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.

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

Управление процессами

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.





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

Команда поддерживает три стиля параметров, причем они не совпадают. Например, ps aux не эквивалентна ps –aux, но ps -e эквивалентна ps ax.

Иерархию процессов можно увидеть с помощью команды **pstree**.

Завершить процесс можно с помощью команды **kill**.

Выполнение команд в фоновом режиме

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

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.

**Утилита renice**

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.

*$ renice [значение приоритета] -p [id процесса]*

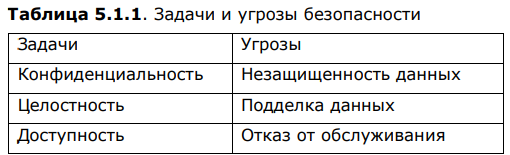
# Тема 5.1. Безопасность и контроль доступа

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

Классификация угроз. Zero trust. Авторизация и аутентификация пользователей. Криптографическая защита данных. Права доступа

Понятие безопасности ОС

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). Они показаны в табл. 5.1.1 и составляют основу свойств безопасности, которыми мы должны защититься от взломщиков и шпионов.



*Позже было решено, что трех основных свойств для всех возможных сценариев недостаточно, и были добавлены дополнительные свойства, такие как аутентичность (authenticity), идентифицируемость (accountability), неотвергаемость (nonrepudiability), закрытость (privacy) и др. Конечно, неплохо было бы обладать всеми этими свойствами. Но даже при этом три исходных свойства по-прежнему занимают особое место в умах и сердцах большинства экспертов по вопросам безопасности.*

Основные угрозы безопасности

Угрозы безопасности операционной системы можно классифицировать по различным аспектам их реализации.

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.

Методы и защитные механизмы операционных систем

Операционную систему называют **защищенной**, если она предусматривает средства защиты от основных классов угроз. Защищенная операционная система обязательно должна содержать средства разграничения доступа пользователей к своим ресурсам, а также средства проверки подлинности пользователя, начинающего работу с операционной системой. Кроме того, защищенная операционная система должна содержать средства противодействия случайному или преднамеренному выводу операционной системы из строя.

Если операционная система предусматривает защиту не от всех основных классов угроз, а только от некоторых, такую ОС называют частично защищенной.

Существует два основных подхода к созданию защищенных операционных систем – **фрагментарный** и **комплексный.**

При **фрагментарном** подходе вначале организуется защита от одной угрозы, затем от другой. Примером фрагментарного подхода может служить ситуация, когда за основу берется незащищенная операционная система, на нее устанавливают антивирусный пакет, систему шифрования, систему регистрации действий пользователей и т. д.

При **комплексном** подходе защитные функции вносятся в операционную систему на этапе проектирования архитектуры операционной системы и являются ее неотъемлемой частью.

Архитектура подсистемы защиты операционной системы

Подсистема защиты ОС выполняет следующие основные функции: идентификация и аутентификация; разграничение доступа; аудит; управление политикой безопасности; криптографические функции; сетевые функции. Подсистема защиты обычно не представляет собой единый программный модуль. Как правило, каждая из перечисленных функций подсистемы защиты решается одним или несколькими программными модулями. Некоторые функции встраиваются непосредственно в ядро ОС.

**Идентификация**

Идентификация субъекта доступа заключается в том, что пользователь (субъект) сообщает операционной системе идентифицирующую информацию о себе (имя, учетный номер) и таким образом идентифицирует себя.

Аутентификация

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**

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. Авторизация субъекта не относится напрямую к подсистеме защиты операционной системы. В процессе авторизации решаются технические задачи, связанные с организацией начала работы в системе уже идентифицированного и аутентифицированного субъекта доступа.

Криптография

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). Криптография использует некоторые низкоуровневые криптографические алгоритмы для достижения одной или нескольких из этих целей информационной безопасности. Среди этих инструментов – алгоритмы шифрования, алгоритмы цифровой подписи, алгоритмы хэширования и другие функции.

Алгоритм шифрования

Алгоритм шифрования – это процедура, которая преобразует сообщение в формате неформатированного текста в зашифрованный текст. Современные алгоритмы используют сложные математические вычисления и один или несколько ключей шифрования. Благодаря этому можно относительно легко зашифровать сообщение, но практически невозможно расшифровать его, не зная ключей. В зависимости от того, как действуют ключи, технологии шифрования делятся на две категории: симметричные и асимметричные.

Криптография с симметричным ключом

Алгоритмы шифрования с симметричным ключом используют одни и те же криптографические ключи для шифрования простого текста и расшифровки зашифрованного. При использовании симметричного шифрования все получатели сообщения должны иметь доступ к общему ключу.

Асимметричная криптография (с открытым ключом)

Схема шифрования называется асимметричной, если в ней один ключ (открытый) используется для шифрования данных, а другой, но математически связанный (частный) – для их расшифровки. Необходимо, чтобы было невозможно вычислить частный ключ, если известен только открытый. Поэтому общий открытый ключ можно передавать, а частный – держать в тайне и в безопасности. Эти ключи называются парой ключей.

Права доступа

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.

**Объект доступа**

Объектом доступа (или просто объектом) называют любой элемент операционной системы, доступ к которому пользователей и других субъектов доступа может быть произвольно ограничен. Возможность доступа к объектам ОС определяется не только архитектурой операционной системы, но и текущей политикой безопасности. Под объектами доступа понимают, как ресурсы оборудования, так и программные ресурсы. В качестве примера ресурсов оборудования можно привести процессор, принтер, жесткие диски и ленты. Каждый объект имеет уникальное имя, отличающее его от других объектов в системе, и может быть доступен через хорошо определенные и значимые операции.

Метод доступа к объекту

Методом доступа к объекту называется операция, определенная для объекта. Тип операции зависит от объектов. Например, процессор может только выполнять команды, сегменты памяти могут быть записаны и прочитаны, считыватель магнитных карт может только читать, а для файлов могут быть определены методы доступа «чтение», «запись» и «добавление» (дописывание информации в конец файла).

# Тема 5.2. Механизмы безопасности в операционных системах семейства Windows

## 32. Механизмы безопасности в операционных системах семейства Windows.

Идентификаторы безопасности (SID). Маркеры защиты. Списки управления доступом (ACL). Доменные службы Active Directory. Виды групп пользователей.

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. Если идентификатор SID используется в качестве уникального идентификатора пользователя или группы, его нельзя использовать повторно для идентификации другого пользователя или группы.

Безопасность Windows использует идентификаторы безопасности в следующих элементах безопасности:

• в дескрипторах безопасности для идентификации владельца объекта и основной группы;

• в записях контроля доступа для определения доверенного лица, для которого разрешен, запрещен или проверен доступ;

• в маркерах доступа — для идентификации пользователя и групп, к которым принадлежит пользователь

Маркеры защиты

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.

Система использует маркер доступа для идентификации пользователя, когда поток взаимодействует с защищаемым объектом или пытается выполнить системную задачу, требующую привилегий. 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.

Каждый процесс имеет основной маркер, описывающий контекст безопасности учетной записи пользователя, связанной с процессом. По умолчанию система использует первичный маркер, когда поток процесса взаимодействует с защищаемым объектом.

Списки управления доступом (ACL)

Списки управления доступом (ACL) состоят из записей управления доступом (ACE). Каждый ACE в ACL идентифицирует доверенного лица и указывает права доступа, разрешенные, запрещенные или регистрируемые для этого доверенного лица.

Дескриптор безопасности для защищаемого объекта может содержать два типа списков управления доступом: **DACL** и **SACL**.

**Список управления доступом на уровне пользователей** (**DACL**) определяет доверенных лиц, которым разрешен или запрещен доступ к защищаемому объекту. Когда процесс пытается получить доступ к защищаемому объекту, система проверяет ACE в DACL объекта, чтобы определить, следует ли предоставить доступ к нему. Если у объекта нет DACL, система предоставляет полный доступ всем пользователям. Если DACL объекта не имеет ACE, система отклоняет все попытки доступа к объекту, так как DACL не разрешает какие-либо права доступа. Система проверяет ACE в последовательности, пока не найдет один или несколько ACE, которые разрешают все запрошенные права доступа, или пока не будет отказано в любом из запрошенных прав доступа.

**Системный список управления доступом (SACL)** позволяет администраторам регистрировать попытки доступа к защищенному объекту. Каждый ACE указывает типы попыток доступа со стороны указанного доверенного лица, которые приводят к созданию системой записи в журнале событий безопасности. ACE в SACL может создавать записи аудита при сбое попытки доступа и/или при успешном выполнении. Предоставляется несколько функций, которые извлекают сведения об управлении доступом из списка управления доступом (ACL). К ним относятся функции для определения прав доступа, которые ACL предоставляет или проверяет для указанного доверенного лица.

Доменные службы Active Directory

Служба каталогов, например, службы домен Active Directory (AD DS), предоставляет методы хранения данных каталога и предоставления этих данных сетевым пользователям и администраторам. Например, AD DS хранит сведения об учетных записях пользователей, таких как имена, пароли, номера телефонов и т. д., а также позволяет другим авторизованным пользователям в той же сети получить доступ к этой информации.

Active Directory хранит сведения об объектах в сети и предоставляет эту информацию администраторам и пользователям, которые могут легко найти и использовать ее. Active Directory использует структурированное хранилище данных в качестве основы для логической иерархической организации сведений в каталоге. Это хранилище данных, также известное как каталог, содержит сведения о объектах Active Directory. Эти объекты обычно включают общие ресурсы, такие как серверы, тома, принтеры и сетевые учетные записи пользователя и компьютера.

Безопасность интегрирована с Active Directory с помощью проверки подлинности входа и управления доступом к объектам в каталоге. С помощью одного входа в сеть администраторы могут управлять данными каталога и организацией по всей сети, а авторизованные пользователи сети могут получать доступ к ресурсам в любой точке сети. Администрирование на основе политики облегчает управление даже очень сложной сетью.

Active Directory также включает следующие компоненты:

• Схема — набор правил, который определяет классы объектов и атрибутов, содержащихся в каталоге, ограничения и ограничения экземпляров этих объектов, а также формат их имен.

• Глобальный каталог, содержащий сведения о каждом объекте в каталоге. Это позволяет пользователям и администраторам находить сведения о каталоге независимо от того, какой домен в каталоге фактически содержит данные.

• Механизм запроса и индекса, чтобы объекты и их свойства могли быть опубликованы и найдены сетевыми пользователями или приложениями.

• Служба репликации, которая синхронизирует данные каталога по сети. Все контроллеры домена в домене участвуют в репликации и содержат полную копию всех сведений о каталоге для своего домена. Любые изменения данных каталога реплицируются в домене на все контроллеры домена.

**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.

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.

• Группы рассылки: создание списков рассылки электронной почты.

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.

# Тема 5.3. Механизмы безопасности в операционных системах семейства Linux

## 33. Механизмы безопасности в операционных системах семейства Linux.

Типы файлов в ОС Linux. Владельцы файлов. Управление правами доступа в файловой системе. Атрибуты файлов. Управление свойствами файлов.

Типы файлов в ОС Linux

В GNU/Linux, как и других Unix-подобных операционных системах, понятие типа файла не связано с расширением файла (несколькими буквами после точки в конце имени), как это обстоит в Windows. Поэтому тип файла в Linux – это скорее тип объекта, но не тип данных, как в Windows. 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

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). В Unix-подобных системах один и тот же файл может существовать под разными именами и/или в разных каталогах: все имена будут связаны с одним и тем же индексным дескриптором (механизм жестких ссылок). Также следует, что файлы всегда должны содержаться в каталогах, иначе станут недоступны, так как нигде не будет содержаться записи о них.

Символьные ссылки

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 по-прежнему будет указывать на файл, которого уже нет. Символьные ссылки не содержат атрибутов файлов, на которые они указывают. У них есть собственные атрибуты (свое время создания, размер, права доступа).

**Команда file**

Линукс не делает предположение о типе данных в обычном файле, но есть специальная утилита, которая выполняет эту задачу, – программа file. Для этого она анализирует начало содержимого файла и находит в нем специальные "сигналы", характерные для определенного типа – бинарного файла, текстового, изображения и др.

Владельцы файлов

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.

Разрешения на доступ к файлу

По сути, разрешения задаются в восьмеричной системе. В числовом представлении они записываются как трех- (или иногда четырех-)значное число. В символьном представлении каждая цифра записывается как три символа: старший бит r, средний бит w, младший бит x, их численные значения соответственно 4, 2, 1. Три крайних правых цифры в числовом представлении относятся к следующим категориям пользователей: первая относится к пользователю-владельцу, вторая – к группе-владельцу, третья – ко всем остальным пользователям. Применяется наиболее точное разрешение. В этом же порядке следуют группы разрешений в символьном представлении, отсутствующие разрешения обозначаются дефисом. 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).

Управление разрешениями

Команда chmod позволяет изменять разрешения двумя способами: символьным и числовым.

Специальные разрешения

Эффект для файлов: u+s (suid) – файл выполняется от имени владельца файла, а не фактического пользователя; g+s (sgid) – файл выполняется от имени группы-владельца файла; o+t (sticky) – нет эффекта. Эффект для каталогов: u+s (suid) – нет эффекта; g+s (sgid) – создаваемые в каталоге файлы в качестве группы-владельца получат группу-владельца каталога; o+t (sticky) – пользователи с разрешением write могут удалять только свои собственные файлы.

**Атрибуты файлов**

В Linux атрибуты файла — это свойства метаданных, которые описывают поведение файла. Например, атрибут может указывать, сжат ли файл, или указывать, можно ли удалить файл. 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.

**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.

# Теоретический минимум

**Операционная система** - это комплекс программ, которые служат интерфейсом между модулями вычислительных систем и прикладными программными приложениями, а также управляют компьютерным оборудованием и процессами вычислений, эффективным распределением вычислительных мощностей среди процессов вычислений.

**Стек вызовов** хранит информацию для возврата управления из подпрограмм (процедур, функций) в программу (или подпрограмму, при вложенных или рекурсивных вызовах) и/или для возврата в программу из обработчика прерывания (в том числе при переключении задач в многозадачной среде).

При вызове подпрограммы или возникновении прерывания, в стек заносится адрес возврата — адрес в памяти следующей инструкции приостановленной программы и управление передается подпрограмме или подпрограмме-обработчику. При последующем вложенном или рекурсивном вызове, прерывании подпрограммы или обработчика прерывания, в стек заносится очередной адрес возврата и т. д.

При возврате из подпрограммы или обработчика прерывания, адрес возврата снимается со стека и управление передается на следующую инструкцию приостановленной (под-)программы.

**Динамически связываемые модули**

**DLL** (Dynamic Link Library — «библиотека динамической компоновки», «динамически подключаемая библиотека») в операционных системах Microsoft Windows — динамическая библиотека, позволяющая многократное использование различными программными приложениями.

Эти библиотеки обычно имеют расширение DLL, OCX (для библиотек содержащих ActiveX), или DRV (для ряда системных драйверов). Формат файлов для DLL такой же, как для EXE-файлов Windows, то есть Portable Executable (PE) для 32-битных и 64-битных приложений Windows и New Executable (NE) — для 16- битных. Так же, как EXE, DLL могут содержать секции кода, данных и ресурсов. В системах Unix аналогичные функции выполняют так называемые общие объекты (англ. shared objects).

Библиотеки не предназначены для запуска напрямую. Загружаются в память процесса загрузчиком программ операционной системы либо при создании процесса, либо по запросу уже работающего процесса, то есть динамически. В UNIX-системах библиотеки имеют расширение so (shared object), в Windows — расширение dll (dynamic link library).

Динамические библиотеки могут использоваться двумя способами:

• динамическая компоновка (dynamic linking)

• динамическая загрузка (dynamic loading) При динамической загрузке программа сама загружает конкретную библиотеку, указывая путь к ней, затем находит в библиотеке нужную функцию по имени и вызывает её. Это частый паттерн использования в программах, которые поддерживают плагины.

**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) (x86) В ядре ОС с каждым потоком связаны следующие структуры:

• Общая информация pid, tid, uid, gid, euid, egid и т. д.

• Состояние процесса/потока

• Права доступа

• Используемые потоком ресурсы и блокировки

• Счетчики использования ресурсов (например, таймеры использованного процессорного времени)

• Регионы памяти, выделенные процессу

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

• Происходит очистка конвейера команд и данных процессора

• Очищается TLB, отвечающий за страничное отображение линейных адресов на физические. Кроме того, следует учесть следующие факты, влияющие на состояние системы:

• Содержимое кэша (особенно это касается кэша первого уровня), накопленное и «оптимизированное» под выполнение одного потока, оказывается совершенно неприменимым к новому потоку, на который происходит переключение.

• При переключении контекста на процесс, который до этого долгое время не использовался, многие страницы могут физически отсутствовать в оперативной памяти, что порождает подкачку вытесненных страниц из вторичной памяти.

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.

Механизмы синхронизации

Критическая область(секция)

Критическая область - часть программы, в которой есть обращение к совместно используемым данным.

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.

Чтобы использовать критические секции, сначала нужно объявить переменную типа **CRITICAL\_SECTION.**

Критический ресурс

Критический ресурс - ресурс, к которому осуществляется одновременный доступ.

**Синхронизация** - Механизм упорядочивания выполнения программных блоков двух или более потоков.

**Атомарная операция** - Операция, которая не может быть прервана и выполняется как единое целое.

**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.

**Объект семафора** — это объект синхронизации, который поддерживает число от нуля до указанного максимального значения. Число уменьшается каждый раз, когда поток завершает ожидание объекта семафора, и увеличивается каждый раз, когда поток освобождает семафор. Когда число достигает нуля, больше потоки не могут ждать, пока состояние объекта семафора станет сигналом. Состояние семафора становится сигнальным, когда это число становится больше нуля, и несигнальным, когда равно нулю. Объект семафора полезен при управлении общим ресурсом, который может поддерживать ограниченное число пользователей.

Мьютексы

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. Упрощенная версия семафора.

**Объект мьютекса** — это объект синхронизации, состояние которого устанавливается в значение Signaled, если он не принадлежит ни одному потоку, и без знака, когда он принадлежит [3]. Только один поток за раз может владеть объектом мьютекса, имя которого происходит из-за того, что он полезен для координации взаимоисключающего доступа к общему ресурсу. Например, чтобы предотвратить запись двух потоков в общую память одновременно, каждый поток ожидает владения объектом мьютекса перед выполнением кода, который обращается к памяти. После записи в общую память поток освобождает объект мьютекса.

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.

**Барьер синхронизации** позволяет нескольким потокам ждать, пока все потоки не достигнут определенной точки выполнения, прежде чем поток продолжит работу. Барьеры синхронизации не могут быть разделены между процессами. Барьеры синхронизации полезны для поэтапных вычислений, в которых потоки, выполняющие один и тот же код в параллельном режиме, должны завершить один этап, прежде чем переходить к следующему. Чтобы создать барьер синхронизации, вызовите функцию **InitializeSynchronizationBarrier** и укажите максимальное количество потоков и количество раз, когда поток должен вращаться перед блокировкой. Затем запустите потоки, которые будут использовать барьер.

Событие

**Объект события** — это объект синхронизации, состояние которого может быть явно задано с помощью функции **SetEvent**. Ниже приведены два типа объекта события.

**Событие сброса вручную**. Объект события, состояние которого остается сигнальным до тех пор, пока функция **ResetEvent** явно не сбросит событие в несигнальное состояние. 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 несигнальное состояние. Если ожидающих потоков нет, состояние объекта события остается сигнальным. Если ожидается несколько потоков, выбирается ожидающий поток. Не предполагайте порядок "первым в очереди" (FIFO). Внешние события, такие как APC в режиме ядра, могут изменить порядок ожидания.

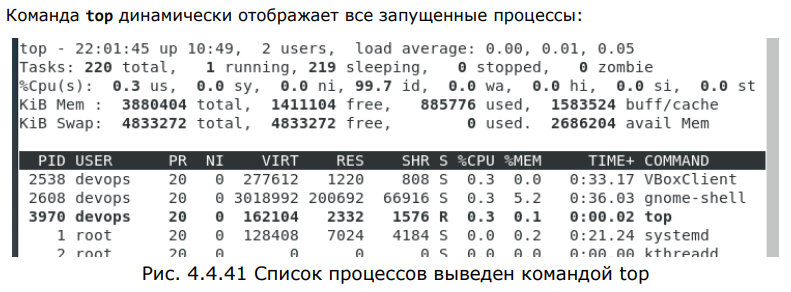
Объект события полезен при отправке в поток сигнала, указывающего на то, что произошло определенное событие.

**Адресное** **пространство процесса** — 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)

Виртуальная память

В основе виртуальной памяти лежит идея, что у каждой программы имеется собственное адресное пространство, которое разбивается на участки, называемые страницами. Каждая страница представляет собой непрерывный диапазон адресов. Эти страницы отображаются на физическую память, но для запуска программы одновременное присутствие в памяти всех страниц необязательно. Когда программа ссылается на часть своего адресного пространства, находящегося в физической памяти, аппаратное обеспечение осуществляет необходимое отображение на лету. Когда программа ссылается на часть своего адресного пространства, которое не находится в физической памяти, операционная система предупреждается о том, что необходимо получить недостающую часть и повторно выполнить потерпевшую неудачу команду.

*Метод управления памятью процессора, предназначенный для выполнения программ, которым выделяется адресное пространство, превышающее доступный физический объем памяти компьютера.*



**SID и ACL**

**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. Если идентификатор SID используется в качестве уникального идентификатора пользователя или группы, его нельзя использовать повторно для идентификации другого пользователя или группы.

Безопасность Windows использует идентификаторы безопасности в следующих элементах безопасности:

• в дескрипторах безопасности для идентификации владельца объекта и основной группы;

• в записях контроля доступа для определения доверенного лица, для которого разрешен, запрещен или проверен доступ;

• в маркерах доступа — для идентификации пользователя и групп, к которым принадлежит пользователь

Списки управления доступом (ACL)

Списки управления доступом (ACL) состоят из записей управления доступом (ACE). Каждый ACE в ACL идентифицирует доверенного лица и указывает права доступа, разрешенные, запрещенные или регистрируемые для этого доверенного лица.

Дескриптор безопасности для защищаемого объекта может содержать два типа списков управления доступом: **DACL** и **SACL**.

**Список управления доступом на уровне пользователей** (**DACL**) определяет доверенных лиц, которым разрешен или запрещен доступ к защищаемому объекту. Когда процесс пытается получить доступ к защищаемому объекту, система проверяет ACE в DACL объекта, чтобы определить, следует ли предоставить доступ к нему. Если у объекта нет DACL, система предоставляет полный доступ всем пользователям. Если DACL объекта не имеет ACE, система отклоняет все попытки доступа к объекту, так как DACL не разрешает какие-либо права доступа. Система проверяет ACE в последовательности, пока не найдет один или несколько ACE, которые разрешают все запрошенные права доступа, или пока не будет отказано в любом из запрошенных прав доступа.

**Системный список управления доступом (SACL)** позволяет администраторам регистрировать попытки доступа к защищенному объекту. Каждый ACE указывает типы попыток доступа со стороны указанного доверенного лица, которые приводят к созданию системой записи в журнале событий безопасности. ACE в SACL может создавать записи аудита при сбое попытки доступа и/или при успешном выполнении. Предоставляется несколько функций, которые извлекают сведения об управлении доступом из списка управления доступом (ACL). К ним относятся функции для определения прав доступа, которые ACL предоставляет или проверяет для указанного доверенного лица.

Права доступа к файлам в Linux

Права доступа могут быть представлены как в буквенном, так и цифровом виде, то есть к примеру  
запись «777» является полностью идентична записи «drwxrwxrwx» или «-rwxrwxrwx».

Существует 3 вида прав доступа:

1. чтение — обозначается как **r** (read),
2. запись — обозначается как **w** (write),
3. выполнение — обозначается как **x** (execute).

И 3 типа пользователей, которым эти права раздаются:

1. Владелец файла или директории
2. Группа пользователей, которой принадлежит владелец
3. Все остальные

Рассмотрим символьное представление прав доступа.

к примеру «drwxrwxrwx» (полный доступ всем) можно поделить на 4 секции: d rwx rwx rwx

**1 секция:** можеть быть либо «d»- директория, либо «-» — файл.

**2 секция:** показывает права доступа владельца файла или директории.

r-чтение, w-запись, x- выполнение.

**3 секция:** показывает права доступа группы в которой находится владелец, rwx — аналогично

**4 секция:** показывает права доступа все остальных пользователей системы.

Теперь рассмотрим цифровое значение прав доступа.

3 цифры прав доступа это 3 секции:владелец, группа владельца, все остальные.

У каждого типа доступа есть свое значение:

r (чтение) — **4**

w (запись) — **2**

x (выполнение) — **1**

Чтобы получить цифру доступа  просто складываем права для каждой секции отдельно, то есть:

7 (4+2+1) — чтение, запись, выполнение;

6(4+2) — чтение, запись;

5(4+1) — чтение, выполнение;

4 — только чтение;

1 — только выполнение.

Права доступа будут выглядеть следующим образом:

script.pl  **755** или **-rwxr-xr-x**

text.txt  **764** или **-rwxrw-r—**

test2  **740** или **drwxr——**

Чтобы изменить права доступа в Linux необходимо воспользоваться командой **chmod**

Например: **chmod 755 test.txt**