Конспект лекций по операционным системам.

Базовые определения

Прикладное ПО - это программы, выполняющие задачи, требуемые пользователю
Системное ПО - это программы, способствующие функционированию прикладных программ и упрощающие их разработку
Системное ПО - это операционные системы, драйверы и фреймворки
Драйвер - программа, управляющая работой периферийного устройства
Операционная система - это программа, обеспечивающая среду выполнения для других программ и облегчающая им доступ к устройствам, составляющим компьютер, процессору, жёсткому диску и т.д.
Фреймворк - это программная среда специального назначения, используемая для того, чтобы упростить написание программ и облегчить объединение отдельных программных компонентов.

Услуги, предоставляемые ОС

Упрощают использование аппаратных средств. Создаваемая ими виртуальная машина заметно отличается от реальной. ОС изолируют пользователей от аппаратной части компьютеров.
ОС обеспечивает распределение вычислительных ресурсов между программами.
Управляет файлами и папками. Для упрощения работы пользователей создаётся файловая система.
Предоставляет пользователю интерфейс для взаимодействия с программами и компьютером.

Интерфейс

Интерфейс - набор средств, используемые для взаимодействия двух систем.

Типы интерфейсов

Графический интерфейс. Управление осуществляется путём нажатия на различные виды кнопок, которые изображены на экране.
Интерфейс командной строки. Управление осуществляется путём набора текстовых команд.
Программный интерфейс. Через программный интерфейс программы взаимодействуют друг с другом. API - Application Programming Interface.
Голосовой интерфейс - команды даются с помощью голоса, речи.
Жестовый интерес - управление с помощью жестов: сенсорный экран, тачпад, джойстик, руль

Нейрокомпьютерный интерфейс - обмен данными между человеческим мозгом и электронным устройством осуществляется с помощью биологической обратной связи и встроенных электронных имплантов.
Аппаратный - предназначений для взаимодействия физических устройств друг с другом: тип разъёмов и параметры сигналов передаваемых через эти разъёмы.

Задание. Процесс.

Задание - совокупности программы и входных данных, необходимых для её выполнения.
Процесс - экземпляр программы во время выполнения, независимый объект, которому выделены системные ресурсы, например, процессорное время и память. Каждый процесс выполняется в отдельном адресном пространстве. Один процесс не может получить доступ к данным другого процесса.
Понятие процесса включает:
1. Множество внешней по отношению к процессу информации, используемой ОС для управления ресурсом типа «процесс». Состав данной информации зависит от ОС.
2. Структура и содержимое адресного пространства процесса. Т.е. части памяти, выделенной процессу.
3. Множество ресурсов, принадлежащих процессу или используемых процессом, а также состояние этих ресурсов.

1 Архитектура ОС

1.1 Монолитная ОС

В монолитной ОС система организуется как набор процедур, каждую из которых может вызывать пользовательская программа. Вся ОС расположена в едином адресном пространстве.
Пользовательский режим - процесс не может получить доступ к чужой области памяти, не все инструкции процессора ему доступны. Поток, исполняющийся в пользовательском режиме, может получить доступ к системным ресурсам только посредством вызова системных сервисов. Когда программа пользовательского режима вызывает системный сервис, вызов перехватывается и вызывающий процесс переключается в режим ядра.
Режим ядра - привилегированный режим работы процессора, в котором исполняется код ОС. Поток, исполняющийся в режиме ядра, имеет доступ ко всей памяти и аппаратуре.
Системные сервисы - набор программ, которые перехватывают обращение прикладных программ к системным ресурсам. Когда выполнение системного сервиса завершается, ОС переключает поток обратно в пользовательский режим.
Привилегия - это свойство, устанавливаемое при проектировании системы, которое определяет, какие компьютерные операции разрешены, какие доступы к памяти законны. Привилегии испльзуются для обеспечения без-

опасности в компьютерной системе и повышения надёжности её работы.
Привилегии реализуются путём присвоения процессам значения от 0 до 3. Значение 0 соответствует наибольшим привилегиям, тогда как значение 3 - наименьшим.
Привилегии реализуются на уровне процессора.

Достоинства монолитной ОС

1. Высокая скорость
2. Относительно простая разработка ОС:

Недостатки:

1. Поскольку всё ядро работает в одном адресном пространстве, сбой в одном из компонентов может нарушить работоспособность всей системы.
2. Усовершенствование ОС затруднено, т.к. внесение изменений в одну часть ОС может потребовать внесение изменений в другие части ОС.
Примеры монолитных ОС: MS DOS, первые версии MacOS

1.2 Модульная операционная система

Модульная ОС - это ОС, в которой каждый программный модуль (часть ОС) имеет законченное функциональное назначение с оговорёнными правилами взаимодействия. Все модули ОС равнозначны. В отличие от монолитной ОС, каждому модулю выделена своя область памяти. Все модули исполняются в режиме ядра.

Достоинства модульной ОС:

1. Упрощается усовершенствование ОС
2. Достаточно высокая надёжность ОС, т.к. сбой в одном модуле не влияет на другие

Недостатки модульной ОС:

1. Усложняется создание ОС
Модульную архитектуру имеет разновидность UNIX - FreeBSD. Linux имеет монолитно-модульную архитектуру.

1.3 Послойная ОС

Достоинства:
1. упрощается усовершенствование ОС
2. Достаточно высокая надёжность ОС
Недостатки:
1. Уменьшение быстродействия ОС.
Частично послойную архитектуру имеет ОС Windows.

1.4 Микроядерные ОС

В сложном программном продукте в среднем содержится 10 ошибок на 1000 строк кода. Следовательно, монолитная операционная ОС, состоящая из 5000000 строк кода, скорее всего, содержит от 10000 до 50000 ошибок. Таким образом, для уменьшения количества ошибок надо уменьшать размер кода, работающего в привилегированном режиме.
MacOS X использует микроядерную архитектуру, которая основана на микроядре Mach. При этом используются некоторые модули, взятые из ОС FreeBSD.
Harmony OS - микроядерная ОС, устанавливается на смартфоны компании Huawei.
Микроядро MINIX 3 занимает всего лишь около 12000 строк кода на языке С и 1400 строк кода на ассемблере.

Достоинства микроядерной ОС:

1. Достаточно высокое быстродействие
2. Высокая надёжность. Микроядерную архитектуру используют ОС, работающие в реальном масштабе времени в промышленных устройствах, авиации и военной технике.

Недостатки микроядерной ОС:

1. Сложность разработки
2. При увеличении числа процессов значительно падает быстродействие, т.к. увеличивается число обращений к ядру.

1.5 Клиент-серверная ОС

Клиент-серверная ОС разделяет процессы на два типа:
1. Процесс сервера, каждый из которых предоставляет какую-нибудь службу
2. Процесс клиента, который пользуется этими службами.
Для связяи клиентов с серверами используется ядро (микроядро).
Связь между клиентами и серверами организуется с помощью передачи сообщений следующим образом:
1. Клиентский процесс составляет сообщение, в котором говорится, что именно ему нужно, и отправляет его ядру или микроядру.
2. Ядро или микроядро ОС определяет, какой сервер должен ответить на сообщение, и доставляет сообщение серверу.
3. Служба выполняет определённую работу и отправляет обратно ответ.
4. Ядро возвращает клиенту результат в виде другого сообщения.

Требования к ОС

Требования правительства США.

2.6 Совместимость с POSIX

Lorem Ipsum

2.7 Безопасность

Безопасность, то есть защита от несанкционированного доступа. Требования, выполнение которых делает систему многопользовательской.
А. Защита от несакционированного проникновения на компьютер.
Б. Защита ресурсов пользователя от других пользователей.
В. Возможность установления квот на системные ресурсы для предотвращения захвата одним пользователем всех системных ресурсов.
Г. Разграничение прав доступа пользователей к файлам и устройствам.
Д. Сохранение информации о действиях пользователей, связанных с безопасностью.

3 Особенности реализации ОС Windows

Ядро (kernel) - компонент операционной системы, непосредственно взаимодействующий с процессом и памятью. Выполняет планирование и переключение потоком, обработку прерываний и исключений, взаимодействие с драйверами и мультипроцессорную синхронизацию.
В Windows также реализована идея помещать в ядро исполнительный механизм, а не политику.

3.1 Поток

В Windows - процесс не является испольняемой сущностью, т.е. непосредственно не выполняется процессором. Это некий объект, который характе-

ризует задачу во время испольнения.
Исполняемая сущность внутри процесса - это поток выполнения (thread of execution).
Поток - это программа, выполняющаяся в адресном пространстве процесса и имеющая доступ ко всем ресурсам процесса. Каждый процесс должен содержать не менее одного потока. Как правило, в Windows процесс содержит несколько потоков.
Поток - это способ распараллеливания вычислительных операций и экономии памяти.
В то время как процесс - это логическое представление работы, которую должна выполнить программа, поток отображает одну из многих необходимых подзадач.

3.2 Прерывания

Прерывания (interrupt) - это запрос, поступающий от устройства ввода-вывода, исполняющейся программы или процессора с требованием прервать выполнение текущей программы и запустить на выполнение другую программу. Прерывания бывают внешние и внутренние.
Внешние прерывания являются внешними по отношению к процессору, поэтому так и называются. Внешние прерывания бывают друх типов: прерывания ввода-вывода и аппаратные прерывания.
Прерывания ввода-вывода создают переферийные устройства, которые сообщают, что они готовы выполнить операцию ввода или вывода. Для выполнения операции ввода-вывода надо прервать выполнение текущей программы и выполнить программу управления вводом-выводом. Программа, вызываемая при возникновении прерывания называется программоnй обработки прерывания.
Аппаратные прерывания поступают в процессор от различных устройств компьютера и не связаны с вводом-выводом. Они сигнализируют о наступлении какого-нибудь события или об обнаружении сбоев в работе какого-нибудь устройства. В этом случае тоже вызывается соответсвтующая программа обработки прерывания.
Внутренние прерывания происходят внутри процессора и бывают двух типов:
Программными прерываниями считаются запросы со стороны программы на начало выполнения операции ввода-вывода или при обращении к функциям операционной системы.
Исключительные ситуации (исключения) возникают в тех случаях, когда процессор не в состоянии выполнить предусмотренное в программе действие, например, деление на ноль. В этом случае он прерывает выполнение программы и сообщает операционной системе о наличии исключительной ситуации, а также выдаёт необходимую дополнительную информацию, которая позволяет более точно определить причину её возникновения.

Исключительные ситуации бывают следующих типов:

Деление на ноль
Пошаговая работа (трассировка)

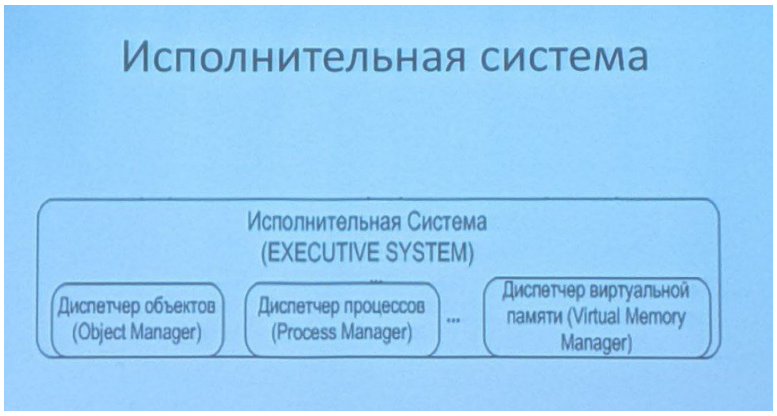
Переполнение порядка при работе со знаковым операндом
Выход индекса за границу, т.е. обращение к области памяти не принадлежащей к данной программе. В этом случае выдаётся ошибка General Protection Fault. (Нарушения основной защиты)
Другие исключительные ситуации

3.3 Уровень абстрагирования от оборудования

Ядро осуществляет взаимодействие с устройствами, но не непосредственно, а через уровень абстрагирования от оборудования. Чтобы работа ядра не зависела от аппаратного обеспечения введён слой абстрагирования от оборудования HAL (Hardware Abstraction Layer).
В HAL входит загружаемый модуль режима ядра (`C:\Windows\System32\hal.dll`, предоставляющий низкоуровневый интерфейс с аппаратной платформой, на которой выполняется Windows. Он скрывает от ОС специфику конкретной аппаратной платформы, т.е. все функции, зависимые от архитектуры и от конкретной машины. Таким образом, ядро работает с некоей стандартизированной виртуальной машиной. HAL преобразует команды ядра в команды, понятные конкретному оборудованию.
Также в HAL входят драйверы различных переферийных устройств.
DLL (Dynamic Link Library) - динамически подключаемая библиотека, подпрограмма, позволяющая многократное использование различными программными приложениями. К DLL относятся разные программные компоненты и в частности драйверы.
В системах UNIX также используются библиотеки. Они имеют расширение .so (shared object).

3.4 Исполнительная система

Исполнительная система (executive system) - спроектирована как уровень абстрагирования от ядра. Она обеспечивает специфические политики для управления объектами, памятью, процессами, файлами и устройствами. Таким образом, ядро реализует механизм, а исполнительная система - политику.
Ядро и исполнительная система включаются в один исполняемый модуль NTOSKRNL.EXE.

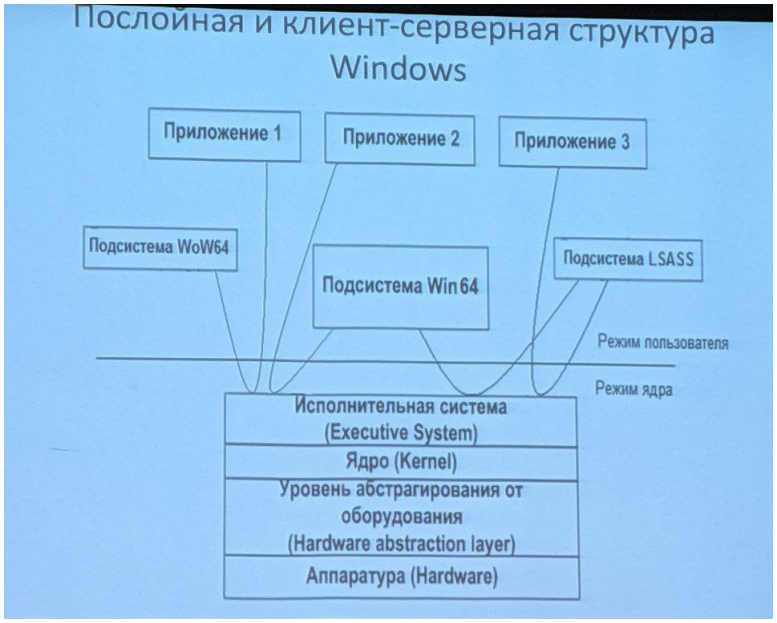


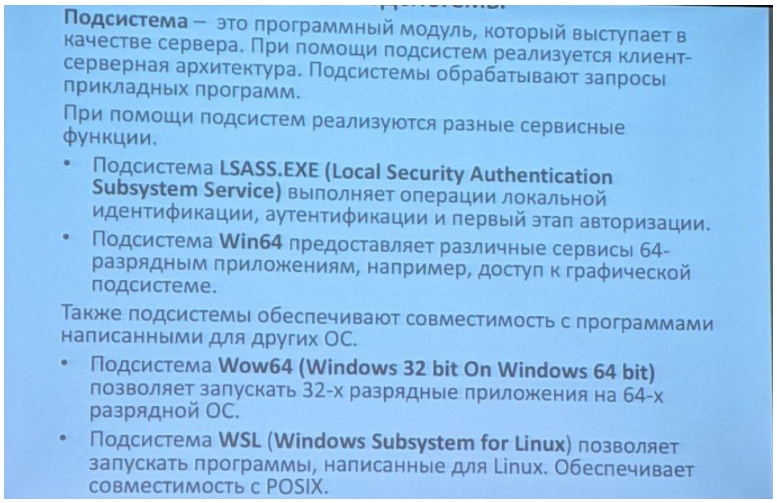
Основные модули исполнительной системы:

1. Диспетчер объектов (Object Manager).
2. Диспетчер процессов и потоков (Process and Thread Manager).
3. Диспетчер виртуальной памяти (Virtual Memory Manager).
4. Справочный монитор защиты (Security Reference Monitor).
5. Диспетчер ввода-вывода (I/O Manager).
6. Диспетчер кэша (Cache Manager).
7. Средство локального вызова процедур (Local Procedure Call).

3.5 Подсистема

Подсистема - это программный модуль, который выступает в качестве сервера. При помощи подсистем реализуется клиент-серверная архитектура.





Подсистема – это программный модуль, который выступает в качестве сервера. При помощи подсистем реализуется клиент-серверная архитектура. Подсистемы обрабатывают запросы прикладных программ.

При помощи подсистем реализуются разные сервисные функции.

- Подсистема **LSASS.EXE (Local Security Authentication Subsystem Service)** выполняет операции локальной идентификации, аутентификации и первый этап авторизации.
- Подсистема **Win64** предоставляет различные сервисы 64-разрядным приложениям, например, доступ к графической подсистеме.

Также подсистемы обеспечивают совместимость с программами написанными для других ОС.

- Подсистема **Wow64 (Windows 32 bit On Windows 64 bit)** позволяет запускать 32-х разрядные приложения на 64-х разрядной ОС.
- Подсистема **WSL (Windows Subsystem for Linux)** позволяет запускать программы, написанные для Linux. Обеспечивает совместимость с POSIX.

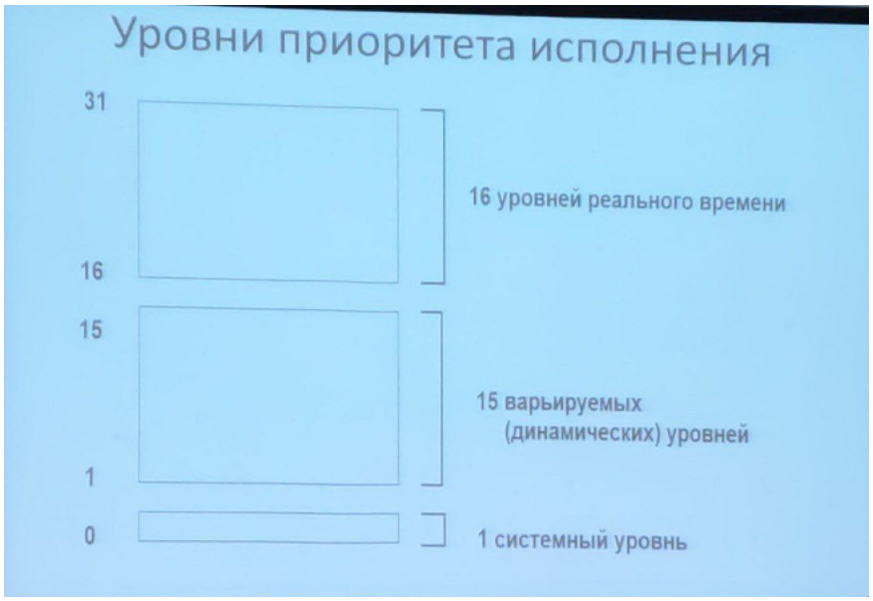
3.6 Приоритеты Windows

В Windows используется приоритетный, с вытеснением и кватованием времени планировщик.
Процессор выделяется потоку на квант времени, вычисляемый, как несколько тиков системных часов. Планировщик поддерживает 32 уровня приоритета и соответственно столько же различных очередей планировщика.
Алгоритм работы планировщика такой:

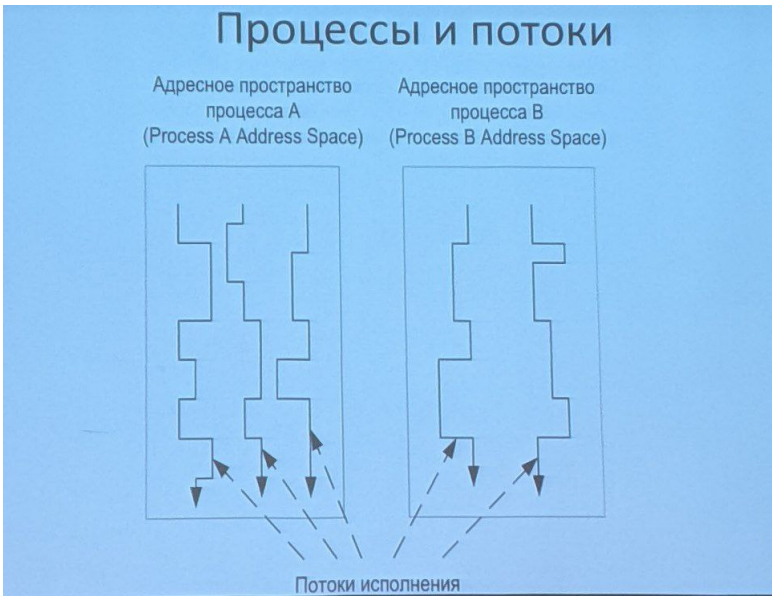
- В соответствии с приоритетом выполнения потоков создаётся несколько очередей. В каждой очереди потоки только с одинаковым приоритетом.

- Сначала выполняются потоки из очереди с самым высоким приоритетом.

- Если в этой очереди нет больше потоков, тогда планировщик будет обслуживать очередь второго по величине приоритета, потом третьего и т.д.



Самый высокий приоритет у потоков реального времени, он больше чем у потоков ОС. Такой приоритет нужен для обработки высокоскоростных потоков данных, которые нужно обрабатывать в реальном времени.
На динамическом уровне выполняются почти все прикладные системные процессы. Особенностью данного уровня является то, что приоритеты потоков могут меняться в течение времени в зависимости от ситуации и действий пользователя. Приоритет повышается для тех потоков, с которыми непосредственно в данный момент работает пользователь.
Системный уровень предназначен для одного системного процесса Idle. Этот процесс обладает самым низким приоритетом и работает, когда процессор простаивает. Процесс Idle удаляет ненужные данные из файла подкачки, также снижает энергопотребление процессора при простое. На каждом ядре процессора запускается по одному потоку процесса Idle.



Приоритеты процессов могут принимать не все возможные значения от 0 до 31, а только несколько определённых значений.

- Real Time Class (значение 14)
- High Class (значение 13)
- Above Normal class (значение 10)
- Normal Class (значение 8)
- Below Normal Class (значение 6)
- Idle Class (значение 4)

По умолчанию приоритет процесса наследуется от родительского процесса следующим образом:

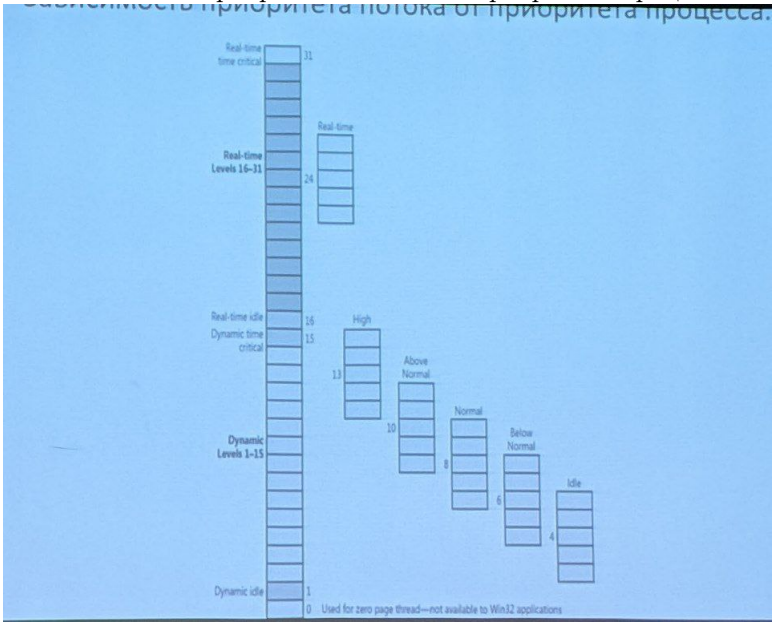
- Если приоритет родительского процесса Normal или ниже, то дочерний процесс имеет приоритет равный родительскому
- Если приоритет родительского процесса выше Normal, то дочерний процесс имеет приоритет Normal.

Приоритет потока - величина, складывающаяся из двух составных частей: приоритета породившего поток процесса и собственно приоритета потока. Приоритет потока задаётся относительно приоритета процесса.

- Normal: такой же какой и у процессора
- Above normal: +1 к приоритету процесса
- Below normal: -1;

- Highest: +2;
- Lowest: -2;
- Time critical: устанавливает базовый приоритет потока для Real Time в 31, для остальных классов в 15;
- Idle: устанавливает базовый приоритет потока для Real Time класса в 16, для остальных классов в 1.

Зависимость приоритета потока от приоритета процесса.



Многопоточность приложений нужна для следующих целей:
Распараллеливания вычислений для ускорения работы программы за счёт того, что разные потоки выполняются на разных ядрах.
Задания разным потокам разных приортетов для повышения удобства работы пользователя с программой. Таким образом базовый (начальный приоритет потока определяется процессом, но ОС может его менять для удобства работы пользвователя с программой. Когда пользователь непосредственно взаимодействует с потоком ОС повышает приоритет потока. Если пользователь перестаёт взаимодействовать с этим потоком, то приоритет понижается.

3.7 Изменение приоритетов

Используя системные функции Windows, программист может задать приоритет процесса при его создании и в дальнейшем при необходимости изменить.
Для создания процесса используется системная функция CreateProcess. Эта функция позволяет задать класс приоритета создаваемого процесса.

Приоритет процесса можно изменить и после его создания, используя функцию SetPriorityClass.
Команда Start позволяет создать процесс и задать его приоритет.
Пользователь получает доступ к функции SetPriorityClass, используя специальные утилиты, например, диспетчер задач и Process Explorer.
Изменение приоритетов индивидуальных потоков внутри процесса обычно не имеет смысла, так как программист определил приоритеты потоков в соответствии с логикой работы программы. Изменение относительных приоритетов потоков может привести к неадекватному поведения приложения.

4 Кеш-память

Кэш-память - это промежуточная память которая сглаживает разницу в быстродействии основных видом памяти. Использование кэш-памяти значительно ускоряет обмен данными между устройствами.

Экспериментально установлено, что после обработки некоторых данных процессор с большой вероятностью обращается к тем же самым данным или к данным, находящимся в непосредственной близости от них. Это явление называется принципом пространственной локализации.
Также установлено, что программе в ближайшее время, вероятнее всего, потребуются данные, которые использовались недавно. Эта закономерность называется принципом временной локализации.
Эти два принципа используются для ускорения обмена данными с дисковыми накопителями.
При чтении данных с диска читаются не только требуемые данные, но и данные находящиеся рядом, т.е. читается не один блок данных, а несколько. Прочитанные данные автоматически заносятся в кэш. Эта процедура называется упреждающее чтение.

Перед чтением данных с жёсткого диска выполняется попытка чтения данных из кэша. Если нужные данные действительно находятся в кеше, то они с высокой скоростью записываются в основную часть оперативной памяти, не относящейся к кешу. Эта ситуация носит название попадание в кеш. Если данные в кеше отсутствуют, то говорят, что имеет место промах кеша, и данные читаются с диска.
Если кеш заполнен полность, то при записи новых данных удаляются те данные, которые записаны давно и продолжительное время не использовались
Такое накопление данных называется кешированием.
При записи данных на диск сначала происходит запись в кеш. Существует 3 способа записи на диск:

1. Прямая запись - данные из кеша сразу записываются на жёсткий диск. Это самый надёжный, но самый медленный вариант.

2. Обратная запись - данные из кеша записываются на жёсткий диск в первом же свободном такте работы процессора. Эта схема использует некоторую задержку во времени относительно момента записи в кеш. Благодаря этому она более эффективна, чем схема прямой записи.

3. Отложенная запись - запись на жёсткий диск выполняется только при окончательном заполнении кеша, то есть когда для помещения в кеш нового значения не оказывается свободной области.

Использование кеш-памяти уменьшает количество обращений к жёсткому диску. Благодаря этому происходит ускорение обмена данными с дисковыми накопителями.
Обратная и отложенная запись значительно ускоряют работу с дисковыми накопителями, но существует риск потери данных при отключении питания компьютера или при потери связи с накопителем.
В Windows кэшем в оперативной памяти управляет соответствующий модуль в исполнительной системе - диспетчер кеша. Диспетчер кеша в частности определяет какая часть оперативной памяти выделяется под кеш.

5 Виртуальная память

Виртуальная память - метод управления памятью компьютера, позволяющий выполнять программы, требующие больше оперативной памяти, чем имеется в компьютере, путём автоматического перемещения частей программы между основной памятью и вторичным хранилищем, например, жёстким диском.
Достоинства использования виртуальной памяти:

- Программа не ограничена объёмом физической оперативной памяти. Успрощается разработка программ, поскольку можно задействовать

большие виртуальные пространства, не заботясь о размере используемой памяти.

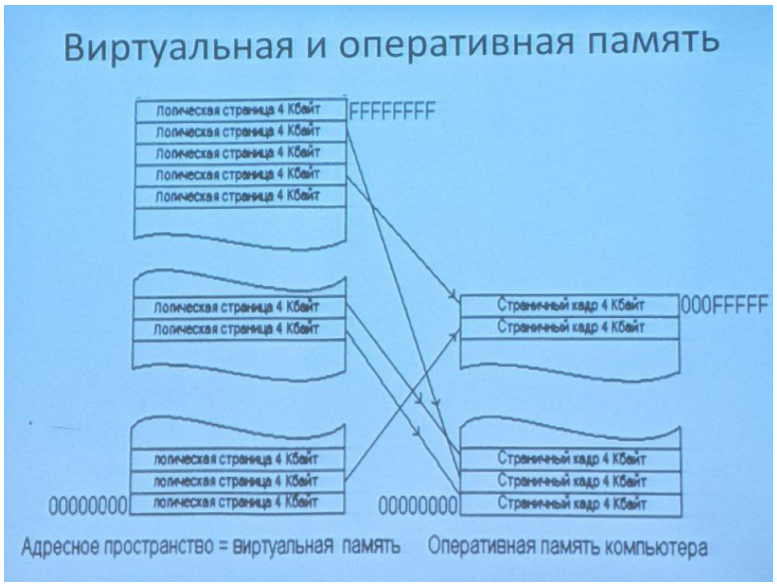
- Поскольку появляется возможность частичного помещения программы (процесса) в память и гибкого перераспределения памяти между программами, можно разместить в памяти больше программ, что увеличивает загрузку процессора и пропускную способность системы.

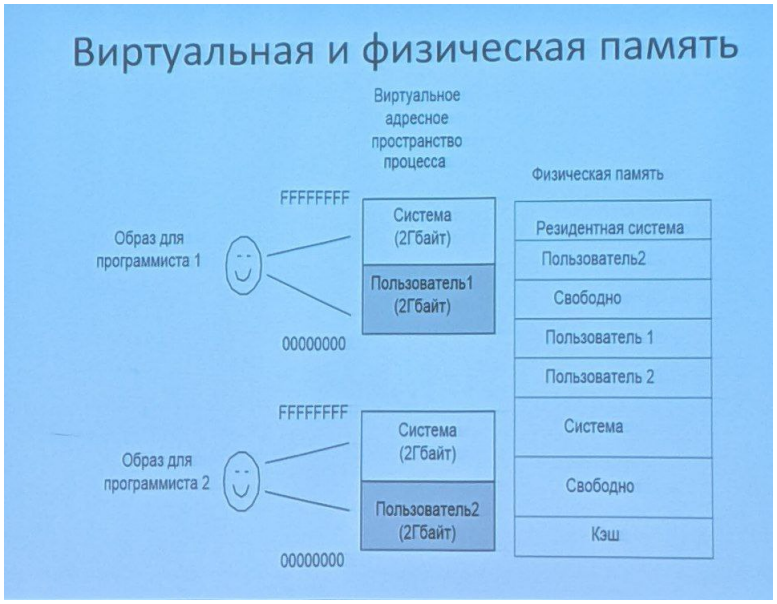
5.1 Файл подкачки.

Файл подкачки - это область на жёстком диске, которая повторяет структуру оперативной памяти и имеет такую же адресацию как оперативная память.
Файл подкачки можно назвать виртуальной памятью. Размер виртуальной памяти гораздо больше оперативной и ограничен только размером жёсткого диска и величиной адресного пространства процессора.
При запуске программы на выполнение создаётся процесс и этому процессу выделяется виртуальное адресное пространство. Таким образом, код и данные копируются сначала в виртуальную память, а потом по мере необходимости по частям копируются из виртуальной памяти в оперативную.
В Windows виртуальная память находится в фалй `pagefile.sys` или `swap.sys`. Этот файл называется файлом подкачки или своп-файлом. Размер этого файла обычно в 2 раза больше размера оперативной памяти.
В ОС Unix и Linux вместо файла подкачки используется раздел подкачки - область на жёстком диске, выделенная для виртуальной памяти.
В MacOS используется несколько файлов подкачки.
`/private/var/vm/swapfile0,.../swapfile1,.../swapfile2,...`
В операционных системах iOS и Android файла подкачки нет.

5.2 Страничная организация памяти

Для организации виртуальной памяти адресное пространство, подразделяется на блоки памяти 4 КБ или 4 МБ каждый. Такие блоки называются логическими страницами.
Фактически имеющаяся в компьютере оперативная память также делится на участки длиной 4 КБ или 4 МБ, которые называются страничными кадрами.
При объёме оперативной памяти 4ГБ образуется 1024 страничных кадра.





При любом обращении к памяти определяется, находится ли связанная с сегментом логическая страница в оперативной памяти. Если страница находится в памяти, то происходит прямое обращение к ней.
В противном случае нужная логическая страница автоматически загружается (подкачивается) операционной системой с жёсткого диска в один из неиспользуемых или редко используемых страничных кадров оперативной памяти.
Адрес в оперативной памяти отличаается от адреса в виртуальной памяти. Для доступа к нужной ячейке памяти нужно осуществить преобразование логического адреса в виртуальной памяти в физический адрес в оперативной памяти. Это преобразование осуществляется диспетчером виртуальной

памяти.

5.3 Адресация памяти

Следует понимать разницу между возможностью адресации виртуальной памяти и возможностью адресации физической памяти. Адресация виртуальной памяти определяется типом процессора.
Максимальный размер адресуемой физической памяти зависит ещё и от операционной системы. Так 32-разрядная ОС Windows может адресовать не более 4ГБ физической памяти. Windows 10 x64 Home может адресовать 128 ГБ, остальные 64-х разрядные версии 2ТБ.

5.4 Диспетчер виртуальной памяти.

В Windows виртуальной памятью управляет диспетчер виртуальной памяти. Диспетчер виртуальной памяти выполняет следующие действия:

1. Выделяет и освобождает виртуальное адресное пространство
2. Обеспечивает своевременную подкачку данных из виртуальной памяти в физическую
3. Обеспечивает отображение логических адресов виртуальной памяти в физические адреса оперативной памяти.

6 Диспетчер объектов

Объектом является любая именованная составляющая компьютерной системы с которой работает ОС Windows за исключением процессора и памяти.
Диспетчер объектов управляет объектами и организует доступ процессов к объектам.

1. Создаёт и удаляет объект
2. Предоставляет доступ к объекту
3. Изменяет информацию об объекте

Объекты бывают двух видов:

1. Объекты, отражающие реальные сущности, существующие в компьютерной системе: файлы, папки, принтеры и т.д.
2. Объекты, не отражающие реальные сущности и необходимые для корректной работы процессов: семафоры, мьютексы и т.д.

Если процессу нужен доступ к какому-нибудь объекту, то происходит обращение к диспетчеру объектов. Диспетчер объектов выполняет следующие действия:

1. Определяет существует ли требуемый объект

2. Если объект не существует, то он создаётся

3. Для доступа процесса к объекту создаётся описатель объекта (handle). В Windows он называется дескриптор.

4. Описатель возвращается процессу.

В дальнейшем при обращении к объекту прикладной процесс передаёт описатель соответствующей функции API, через которую происходит обращение к объекту.
Когда другой процесс попытается обратиться к уже существующему объекту, то диспетчер объектов создаёт второй описатель объёкта для этого процесса. Таким образом, число описателей в операционной системе гораздо больше числа объектов.
В Windows существует 15 типов объектов. При создании объектов, диспетчер объектов создаёт запись, содержащую информацию об объекте. Эта запись содержит заголовок и тело. Заголовок используется диспетчером объектв для управления объектом. Тело содержит зависимою от типа объекта, информацию, позволяющую получить доступ к объекту. Заголовок включает в себя следующие данные:

- Имя объекта. Позволяет различным процессам ссылаться на объект.
- Дескриптор защиты. Содержит разрешения доступа.
- Информация об открытых описателях. Содержит информацию о том, какие процессы используют описатели объекта.
- Тип объекта. Один из 15 типов.
- Счётчик ссылок. Содержит количество открытых описателей объекта.

7 Диспетчер потоков и процессов.

Диспетчер процессов и потоков выполняет следующие действия:

- Создание и уничтожение процессов и потоков.
- Выделяет память созданному процессу.
- Управляет изменениями состояния процессов и потоков.
- Сохраняет необходимую информацию о каждом процессе и потоке.
- Удаляет зависшие процессы.
- Ведёт подсчёт потребляемыми ресурсами.

Для управления процессами и потоками диспетчер процессов сохраняет информацию о каждом процессе и потоке в памяти.
Для сохранения информации о процессе создаётся дескриптор процесса. Он называется **executive process control block (EPROCESS)**. При создании потока создаётся дескриптор потока **executive thread process control block (ETHREAD)**.

8 Справочный монитор защиты

Справочный монитор защиты выполняет второй этап авторизации - проверяет права доступа процесса к объекту.
Когда поток пытается получить досутп к объекту, его запрос передаётся справочному монитору защиты. Каждый объект имеет дескриптор безопасности, который содержит **владельца объекта и список контроля доступа (access conrol list, ACL)**.
Список контроля доступа содержит список пользователей, которым разрешён доступ к объекту. Также в этом списке содержится информация с какими правами разрешён доступ пользователя к объекту. В компьютере от имени пользователя действует процесс.
Справочный монитор защиты определяет индентификатор процесса и тип требуемого им доступа, а затем проверяет, разрешён ли ему заданный доступ.

9 Средство локального вызова процедур

Средство локального вызова процедур - механизм исполнительной системы для обмена сообщениями между клиентом и сервером. LPC реализует клиент-серверную архитектуру следующим образом.

1. После запуска процесса сервера, он создаёт объект типа именованный порт (named pipe). Через этот порт можно соединиться с сервером.
2. Клиент запрашивает соединение с объектом типа порт, отправив соответствующее сообщение.
3. Если сервер отвечает на запрос, создаются два безымянных порта.
   (a) Коммуникационный порт клиента используется процессом клиентом для связи с сервером;
   (b) Коммуникационный порт сервера для связи с клиентом, по одному порту на каждого клиента;
4. Клиент получает описатель (handle) серверного порта от LPC, а сервер получает описатель клиентского порта. Таким образом, устанавливается связь между процессами.

LPC поддерживает 3 способа передачи сообщений между клиентом и сервером:

1. Сообщение короткое - меньше или равно 256 байт. Ядро копирует сообщение из адресного пространства одного процесса в системное адресное пространство, а оттуда в адресное пространство дрегого процесса.
2. Сообщение длиной более 256 байт:
   (a) LPC создаёт область памяти, разделяюемую между двумя разными процессами.
   (b) Первый процесс записывает данные в разделяемую область памяти.

(c) Первый процесс отправляет методом 1 сообщение, что можно считать данные из разделяемой области памяти.

(d) Второй процесс считывает данные.

3. Если объём передаваемых данных очень большой, и эти данные надо передавать с минимальной задержкой, то сервер читает и записывает данные прямо в адресное пространство клиента.

10 Блок управления процессом и потоком, контекст процесса

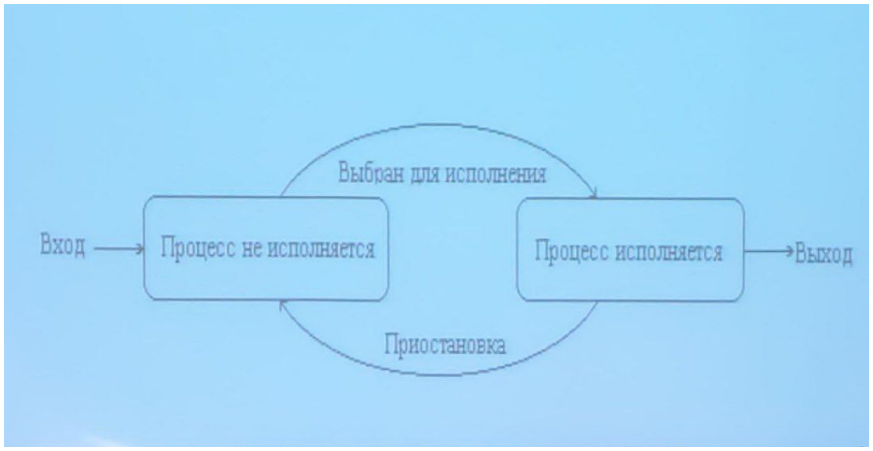
Информация о процессе хранится в специальной структуре данных, которая называется **Process Control Block (PCB) - блок управления процессом**. Есть аналогичный **блок управления потоком Thread Control Block (TCB)**
В этих блоках содержится следующая информация:

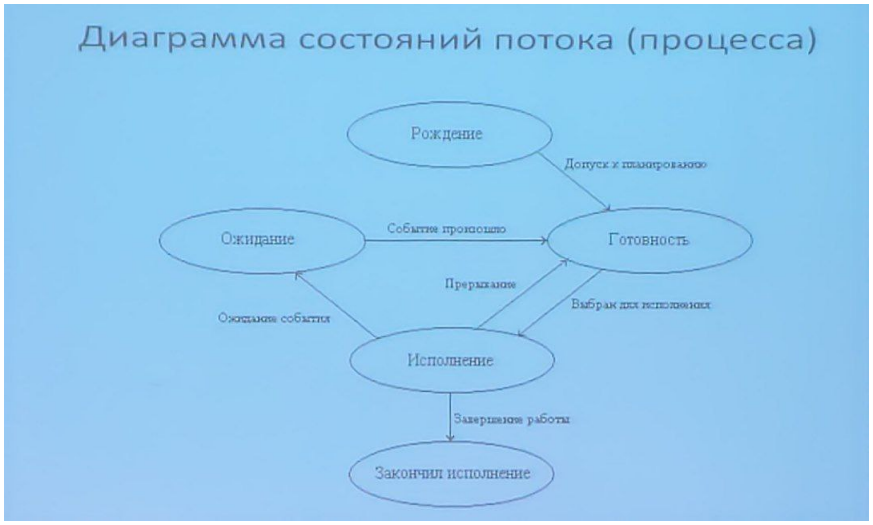
1. Идентификатор процесса или потока - уникальный номер, используется для идентификации процесса или потока.
2. Групповые параметры процесса: родительский процесс, дочерние процессы, процессы, принадлежащие одному заданию и т.д.
3. Приоритет процесса или потока.
4. Состояние процесса или потока.
5. Статистические данные: время создания процесса; время процессора, использованное всеми потоками процесса; время процессора, использованное каждым потоком и т.д.
6. Описание виртуального адресного пространства процесса.
7. Контекст ввода-вывода - информация, определяющая возможности процесса по взаимодействию с устройствами ввода-вывода. Для процесса может быть указан объект с которого он может получать входные данные, и объект, куда процесс должен осуществлять вывод.
8. Контекст безопасности: владелец процесса, группы к которым принадлежит владелец процесса.
9. Текущие системные параметры выполнения: логический диск, рабочий каталог, системные переменные и т.д.
10. Код завершения процесса или потока.

Блок управления процессом содержит всю необходимую операционной системе информацию для управления процессом, поэтому данный блок называют системным контекстом процесса.
Содержимое всех регистров процессора называется регистровый контекст процесса.
Код и данные, находящиеся в адресном пространстве процесса, - это пользовательский контекст процесса.

Код и данные, находящиеся в адресном пространстве процесса - это пользовательский контекст процесса.
Совокупность регистрового, системного и пользовательского контекстов процесса принято называть контекстом процесса.

11 Состояние потока (процесса)



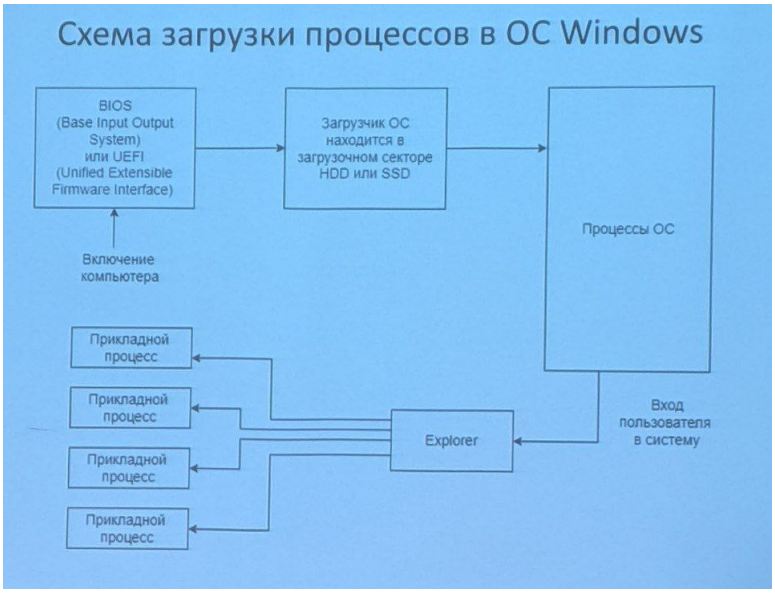


11.1 Рождение процесса

1. Создаётся Process Control Block процесса.
2. Задаётся ID процесса.
3. Процессу выделяется адресное пространство в виртуальной памяти и другие ресурсы.
4. Программа и данные загружаются в виртуальное адресное пространство процесса.

5. Устанавливается начальное значение программного счётчика процесса.

Родившийся процесс переводится в состояние "готовность".
Инициатором рождения нового процесса является другой процесс: системный или пользовательский. Для создания процесса нужно выполнить специальный системный вызов.
Процесс, инициировавший создание нового процесса, называется родительским процессом. Созданный процесс называется дочерним процессом. Дочерние процессы могут создавать другие процессы, в результате возникает дерево процессов. Если таких деревьев возникает несколько, то они называются лесом.



Для выполнения своих функций новому процессу требуются определённые ресурсы: память, файлы, устройства ввода-вывода и т.д. Существует два подхода к их выделению:

1. Новый процесс получает в своё пользование ресурсы родительского процесса. В Windows таков подход используется при создании потоков.

2. Новый процесс получает свои ресурсы непосредственно от операционной системы. В Windows такой подход используется при создании процессов.

Информация о выделенных ресурсах заносится в PCB (Process Control Block). После выделения процессу адресного пространства нужно занести туда код и данные. При этом может использоваться один из двух методов:

1. Дочерний процесс становится дубликатом родительского процесса по регистровому и пользовательскому контекстам. В дальнейшем требуемые код и данные загружаются из файла, меняется также регистровый контекст. Такой подход используется в UNIX и Linux.

2. Дочерний процесс сразу загружает пользовательский контекст из файла. Такой подход используется в Windows.

После того как процесс наделён содержанием, в PCB дописывается оставшаяся информация и состояние нового процесса меняется на **готовность**.
Рассмотрим как ведут себя родительские процеесы после создания дочерних процессов:

1. Родительский процесс продолжает своё выполнение одновременно с дочерним процессом.

2. Родительский процесс ожидает завершение дочернего процесса.

11.2 Завершение процесса.

Рассмотрим, что происходит после того, как процесс завершил свою работу.

1. Операционная система переводит его в состояние **"закончил исполнение"**

2. Освобождает все ассоциированные с ним ресурсы, делая соответсвующие записи в блоке управления процессом.

При этом PCB завершённого процесса не уничтожается, а остаётся в системе ещё некоторое время. Это связано с тем, что родительский процесс после завершения дочернего процесса может запросить операционную систему информацию о работе дочернего процесса. В первую очередь может понадобиться код завершения процесса. Также может понадобиться статистичечкая информация по работе дочернего процесса.
Подобная информация сохраняется в PCB завершённого процесса до запроса родительского процесса или до конца его работы, после чего PCB удаляется. В операционной системе UNIX процессы, находящиеся в состоянии **закончил исполнение** , принято называть **процессами зомби**.
В некоторых операционных системах, например, в системах реального времени, завершение родительского процесса приводит к завершению работы всех дочерних. В Windows, UNIX, MacOS дочерние процессы продолжают своё существование и после окончания работы родительского процесса.
В Windows после завершения родительского процесса дочерний утрачивает связь с деревом процессов и становится началом нового дерева. В UNIX для такого процесса родительским процессом становится системный процесс **init**. В результате дерево процессов сохраняет свою целостность.

11.3 Запуск потока (процесса).

Из числа потоков, находящийся в состоянии готовность, операционная ситема выбирает один поток для последующего исполнения

1. Для выбранного потока операционная система обеспечивает наличие в оперативной памяти информации, необходимой для его дальнейшего исполнения.

2. Состояние потока меняется на исполнение.

3. Восстанавливаются значения регистров для данного потока, и управление передаётся команде, на которую указывает счётчик команд потока.

Все данные, необходимые для восстановления контекства, извлекаются из PCB потока.

11.4 Приостановка потока (процесса).

Работа потока, находящегося в состоянии исполения, приостанавливается в результате какого-либо прерывания.

1. Процессор сохраняет счётчик команд и содержимое регистров в стеке потока.

2. Процессор передаёт управление программе обработки прерывания. Начало программы обработки прерывания должно находится по определённому адресу.

3. Программа обработки прерывания сохраняет системный и регистровый контекст в его PCB.

4. Программа обработки прерывания переводит потокк в состояние готовность.

5. Программа обработки прерывания приступает к обработке прерывания, то есть к выполнению определённых действий, связанных к возникшим прерываниям.

11.5 Блокировка и разблокировка потока (процесса).

Поток блокируется, когда он уже не может продолжать свою работу, не дождавшись возникновения какого-либо события в вычислительной системе. Блокирование происходит, когда поток обращается к операционной системе с помощью системного вызова. Операционная система обрабатывает системный вызов следующим образом:

1. Сохраняет контекст потока.

2. Переводит поток из состояния исполнение в состояние ожидание.

3. Добавляет поток в очередь потоков, дожидающийся возникновения события.

4. Инициализирует операцию ввода-вывода или другое действие.

После возникновения в системе какого-либо события, операционная система определяет, какой поток ожидает это событие. Данный поток переводится в состояние готовность.

11.6 Переключение контекста

Для корректного переключения процессора с одного потока на другой необходимо **сохранить контекст** исполняющегося потока и **восстановить контекст** потока, на который будет переключен процессор. Такая процедура сохранениявосстановления работоспособности потоков называется **переключением контекста**. Время, затраченное на переключение контекста, не используется вычислительной системой для совершения полезной работы и представляет собой накладные расходы, снижающие производительность системы. Оно определяется быстродействием компьютера и находится в диапазоне от 1 до 1000 микросекунд.

12 Вытесняющее и невытесняющее планирование.

Процесс планирования осуществляется частью операционной системы, называемой **планировщиком**. В большинстве операционных систем в том числе Windows **планировщик** является частью ядра. **Планировщик** может принимать решения о выборе для исполнения нового потока, из числа находящийся в состоянии **готовность**, в следующих четырёх случаях:

- Когда поток переводится из состояния **исполнение** в состояние **завершение**
- Когда поток переводится из состояния **исполнение** в состояние **ожидание**
- Когда поток переводится из состояния **исполнение** в состояние **готовность** (например, после прерывания от таймера)
- Когда поток переводится из состояния **ожидание** в состояние **готовность** (завершилась операция ввода-вывода или произошло другое событие)

Следует отметить, что в двух случаях прерывания поток самостоятельно останавливает свою работу. В друх других - работа потока останавливает извне.
Когда используется все четыре варианта остановки потока для осуществления планировани, такой механизм называется **вытесняющее планирование**. **Вытесняющее планирование** используется во всех современных ОС разделения времени.
Если в операционной системе не используется механизм прерывания извне, то планирование осуществляется только в том случае, когда поток сам прерывает свою работу. Такой механизм называется **не вытесняющее планирование**. Не вытесняющее планирование использовалось в MS DOS, MacOS ранних версий, а также в 16-разрядных версиях Windows.
Не вытесняющее планирование относительно просто реализуется и достаточно эффективено, так как позволяет использовать большую часть процессорного времени на работу самих процессов и до минимума снизить затраты на переключение контекстов. Однако при не вытесняющем планировании

возникает проблема возможности полного захвата процессора одним зависшим процессом. В таком ситуации необходимо использовать перезагрузку всей вычислительной системы. При вытесняющем планировании такой проблемы нет.
Рассмотрим реализацию механизма прерывания в случае окончания кванта времени, выделенного на выполнение потока.

1. Системный таймер отсчитывает квант времени.
2. По истечении времени, отведённого на выполнение потока сигнал подаётся на специальный вход процессора. Создаётся аппаратное прерывание.
3. Процессор получает сигнал прерывания и останавливает выполнение команд потока. В стеке сохраняется содержимое регистров процессора.
4. Сигнал прерывания содержит номер прерывания. По этому номеру процессор вызывает нужную программу обработки прерывания. В данном случае вызывается планиройщик
5. Планировщик записывает регистровый контекст из стека в специальную область памяти
6. При помощи диспетчера процессов делаются необходимые изменения в блоке управления потоком.
7. Выбирается поток для запуска.
8. При помощи диспетчера процессов делаются необходимые изменения в блоке управления потоком.
9. Поток запускается на выполнение.

13 Процессы и потоки в Windows.

13.1 Процессы в Windows

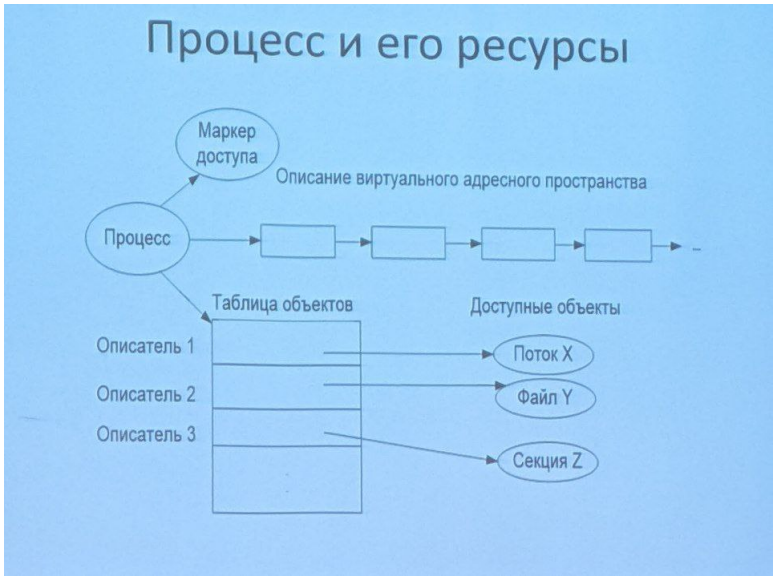
Диспетчер процессов управляет созданием и удалением процессов, а также отношениями между ними: родительский - дочерний процессы. Таким образом определяется древовидная структура процессов.
Особенности процессов Windows:

- Процессы реализованы как объекты, и доступ к ним осуществляется посредством объектных сервисов (по принципу ООП).
- В адресном пространстве процесса может исполняться несколько потоков.
- Объект процесс и объект поток имеют встроенные возможности синхронизации.

Процесс состоит из:

- Исполняемой программы (код и данные).

- Закрытого адресного пространства, т.е. набора адресов виртуальной памяти, который процесс может использовать.
- Системных ресурсов, выделяемых операционной системой процессу во время выполнения программы (семафоров, файлов и т.д.).
- По крайней мере одного потока управления.
- Блока управления процессом (Process Control Block).



13.2 Потоки в Windows

Поток - это единица исполнения, отдельный счётчик команд или подлежащая планированию сущность внутри процесса. В то время как процесс - это логическое представление работы, которую должна выполнить программа, поток отображает одну из многих необходимых подзадач.
Блок управления потоком:

- Уникальный идентификатор.
- Содержимое набора регистров, отражающее состояние процессора.
- Два стека: один используется потоком при работе в пользовательском режиме, а другой - в режиме ядра.
- Собственная область памяти, предназначенная для использования библиотеками (Run-time Library. Dynamic Link Library).

Run-time Library (библиотека времени выполнения) - специальная программная библиотека, используемая компилятором для реализации функций, встроенных в ЯП, во время выполнения компьютерной программы. В эти библиотеки входят функции ввода и вывода, управления паматью или математические функции.

Run-time Library (RTL) можно внедрять в программу, но при этом увеличивается размер программы. RTL можно устанавливать в JC и вызывать при выполнении программы. Обширный набор RTL образует среду выполнения, устнавливаемую поверх ОС.
Содержимое регистров, стека и собственной области памяти называется контекстом потока.
Поток находится в адресном пространстве процесса, используя его для хранения данных во время выполнения. Если в одном процессе существует несколько потоков, то они совместно используют адресное пространство и все ресурсы, включая маркер доступа, базовый приоритет и описатели объектов из таблицы объектов процесса.

13.3 Динамический приоритет потоков Windows

Для выбора порядка в котором исполняются потоки, ядро NT использует приоритеты. Windows динамически повышает приоритет некоторых потоков для оптимизации общей пропускной способности и отзывчивости системы в следующих случаях:

1. При переводе потока в состояние ожидание ввода/вывода приоритет потока повышается на 2. После разблокирования потока и запуска его на выполнение, в течение одного кванта поток выполняется с этим приоритетом. После этого приоритет уменьшается на один уровень. После выполнения в течение ещё одного кванта приоритет уменьшается ещё на 1.

2. При выводе окна на передний план приоритет потока, отвечающего за это окно повышается на 2. При переводе окна в фон приоритет окна снижается на 2.

3. По окончании ожидания на каком-либо объекте исполнительной системы приоритет потока увеличивается на один уровень. После разблокирования потока и запуска его на выполнение, в течение одного кванта поток выполняется с этим приоритетом. После этого приоритет уменьшается на один уровень.

4. Периодически планировщик сканирует очереди готовых потоков и ищет потоки, которые находились в состоянии **Ready** более 3 секунд. Обнаружив такой поток, диспетчер повышает его приоритет до 15 и выделяет ему квант вдвое больше обычного. По истечении двух квантов приоритет потока снижается до исходного уровня.

13.4 Достоинства и недостатки многопоточного программирования

Достоинства:

- Ускорение вычислений за счёт распараллеливания вычислительных операций.

- Повышение комфорта работы пользователя с программой за счёт разделения разных операций по потокам и повышения приоритета у того потока, который отвечает за взаимодействие с пользователем..

- Быстрое создание потоков по сравнению с созданием процессов.
- По сравнению с многопроцессными приложениями экономится память за счёт совместного использования памяти несколькими потоками.
- Простой и быстрый обмен данными между потоками (по сравнению с обменом данными между процессами).

Недостатки

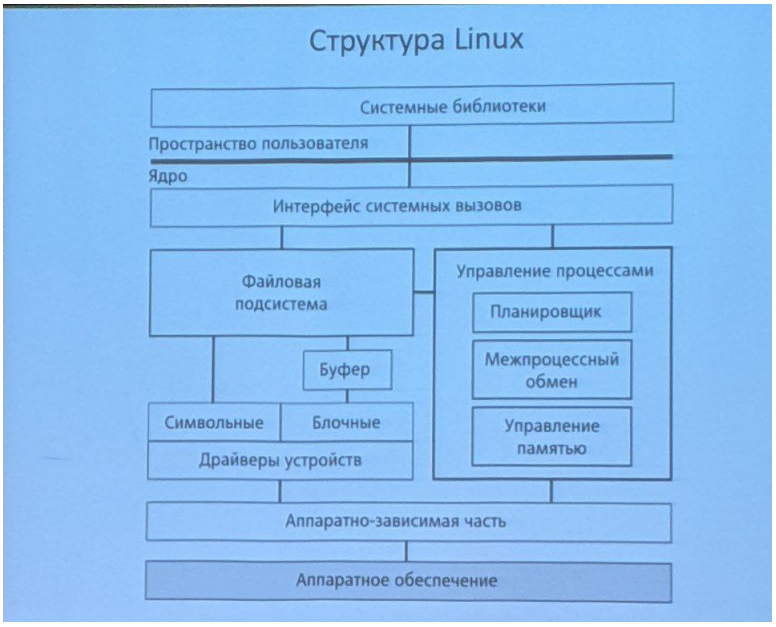
- В большинстве случаев невозможно разпараллелить данные для проведения параллельных вычислений.
- Сложность программирования многопоточных приложений из-за необходимости синхронизцаии работы потоков.
- Многопоточные приложения требуют больше оперативной (не виртуальной) памяти по сравнению с однопоточными.

14 Операционные системы семейства UNIX/Linux

Операционная система UNIX относится к категории многопользовательских многопрограммных ОС, работающих в режиме разделения времени. На основе UNIX создана ОС Linux. На основе Linux - ОС Android. Существует множество разновидностей UNIX и Linux.
UNIX - FreeBSD, OpenBSD, Solaris.
Linux - Ubuntu, CentOS, Fedora, openSUSE. Для того, чтобы унифицировать API используется стандарт **POSIX** (Portable Operating System Interface for UNIX). **POSIX** - набор стандартов описывающих интерфейсы между операционной системой и прикладной программой (системный API), библиотеку языка C и набор приложений и их интерфейсов. Стандарт создан для обеспечения совместимости различных UNIX-подобных операционных систем переносимости прикладных программ на уровне исходного кода, но может быть использован и для не-UNIX систем.

14.1 Архитектура Linux

Демон - это системная служба, выполняющая определённую функцию. Как правило демоны запускаются в процессе старта операционной системы.
Системные утилиты предоставляют пользователю определённые сервисные функции. Вызываются по мере необходимости.

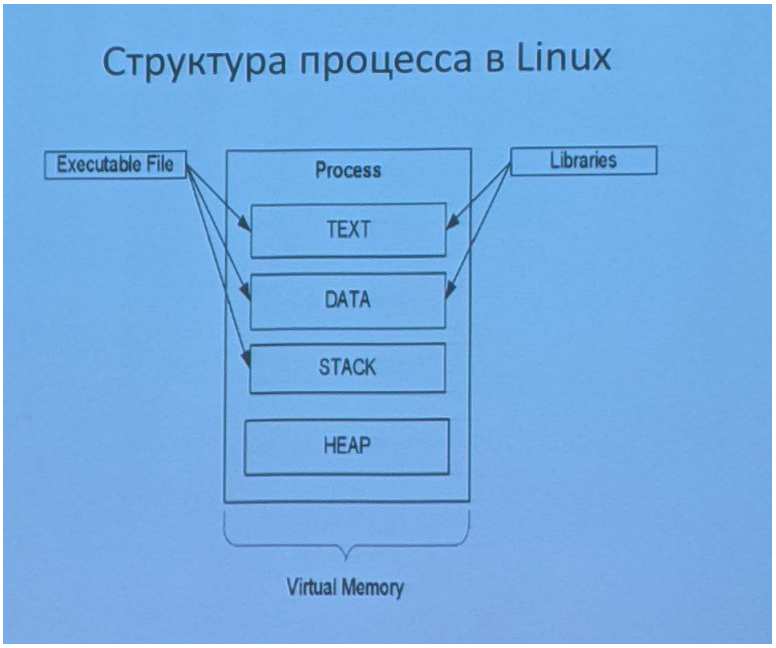


Ядро ОС UNIX состоит из двух основных частей: управления процессами и управления устройствами. Управление процессами резервирует ресурсы, определяет последовательность выполнения процессов и принимает запро-

сы на обслуживание. Управление устройствами контролирует передачу данных между оперативной памятью и периферийными устройствами.
Символьные устройства обрабатывают данные последовательно посимвольно. Произвольный доступ к данным невозможен.
Блочное устройство хранит и обрабатывает информацию отдельными блоками. Используется произвольный доступ к данным, хранящимся в блоках.
Пользователь взаимодействует с компьютером через один или несколько терминалов. Управляет терминалами **X Windowing System**.
Один из этих терминалов имеет графический интерфейс. Наиболее известные графические оболочки GNOME, KDE и другие.
В текстовых терминалах можно использовать различные командные интерпретаторы. Наиболее известные: bash, tcsh, zsh.

14.2 Процессы в Linux

Организация работ в ОС Linux основана на понятии **процесса как единицы работы, управления и потребления ресурсов**. Взаимодействие процессов с ядром происходит по принципу сопрограмм, т.е. процесс системных функций и ядра используется API.



Разделы памяти.

TEXT - программный код. Доступ к этому участку памяти только для чтения.
DATA - данные программы. Данные можно изменять, хотя могут быть данные только для чтения.

STACK - стек нужен для организации функциональных блочных вычислений. При вызове подпрограммы в стеке запоминается адрес основной программы, откуда производился вызов, и содержимое регистров процессора. При вызове подпрограммы из вызванной подпрограммы запоминается аналогичная информация. После завершения работы подпрограммы из стека извлекается адрес памяти, откуда надо продолжить выполнение программы.
HEAP - куча. Динамические данные программы, под которые выделяется память.
Программа фактически состоит из двух частей: самой программы, находящейся в исполняемом файле, и библиотек, которые требуются для выполнения программы. Библиотеки загружаются двумя способами.
При запуске исполняемого файла. Это происходит, когда библиотека обязательно потребуется при работе программы.
Программа в процессе выполнения может потребовать загрузить библиотеку. Это происходит тогда, когда библиотека нужна только при выполнении некоторых необязательных операций.
В Linux, как правило, одна библиотека зависит от другой, и одна программа может зависеть от другой.

14.3 Потоки в Linux

Легковесные (лёгкие) процессы - это аналог потоков в Windows. Они создаются для совместного использования адресного пространства. Такие легковесные процессы образуют группу процессов.
Программа на стадии выполнения называется **заданием**. Задание может содержать несколько процессов, образующих группу процессов.
Группы нумеруются по ID первого созданного процесса. Даже если процесс завершает свою работу, то ID группы не меняется.

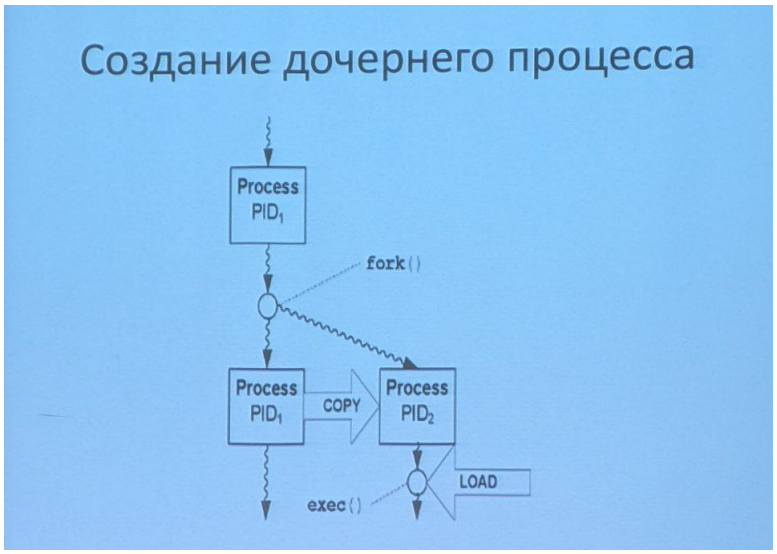
14.4 Планирование в Linux

В UNIX также используется приоритетный метод планирования выполнения процессов. Приоритет называется **nice** (любезность). В отличие от Windows в UNIX чем выше число тем ниже приоритет. Всего 40 уровней приоритета от -19 до 20.
По умолчанию процесс получает значение 0. Дочерний процесс наследует приоритет родительского.
Используя специальные команды, можно устанавливать приоритет запускаемой программы или изменять приоритет уже запущенной программы.
nice - запуск программы с определённым приоритетом.
renice - изменение приоритета уже запущенной программы.

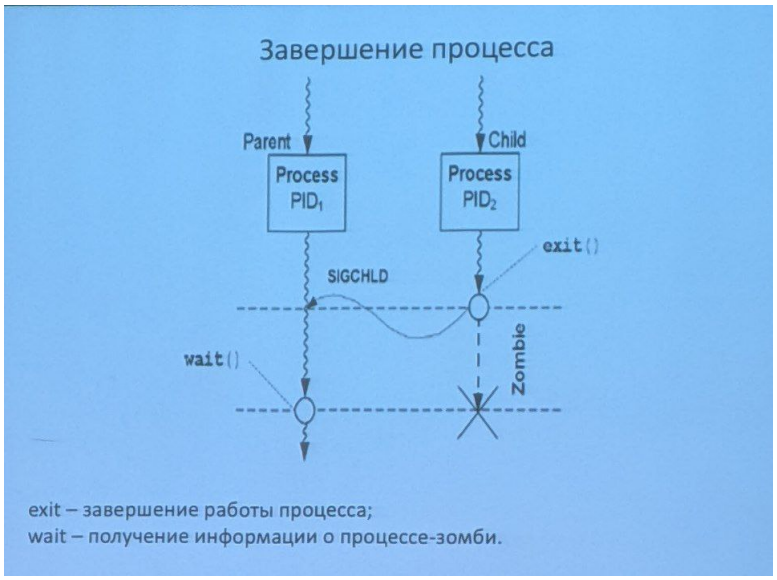
14.5 Рождение и завершение процесса в Linux

В UNIX рождение процесса происходит путём дублирования родительского процесса. В результате у дочернего процесса пользовательский контекст

такой же как и у родительского процесса.



Для замены контекста используется специальная команда ОС. В результате выполнения этой команды происходит загрузка в память исполняемого файла и библиотек. После этого процесс переходит в состояние готовность.
fork - создание обычного процесса.
clone - создание легковесного процесса. Не создаются копии адресного пространства родительского процесса. Потоки используют совместно адресное пространство вместе с другими ресурсами, включая дескрипторы файлов.
exec - загрузка программы в адресное пространство процесса.



14.6 Псевдофайловая система

В Windows используется объектная модель, а в Linux - файловая. Все сущности в Linux представляются файлами. В ОС Linux/UNIX используется шесть типов файлов:

- **Обычный файл** - именованная область на внешнем носителе. Обычные файлы содержат данные пользователей.
- **Символьная ссылка** - содержит путь к некоторому файлу.
- **Каталог** - это файл, содержащий информацию о файлах и других каталогах, находящийся внутри каталога. Каталоги используются системой для поддержания файловой структуры.
- **FIFO файлы (First Input - First Output** - используются для обмена данными между процессами. Данные передаются через ядро, при этом создаётся именованный канал Named Pipe.
- **Файл сокет (Socket)** - тоже используется для межпроцессного взаимодействия, но именованные каналы не создаются, создаются сокеты. Сокеты можно адресовать по IP адресу и номеру порта. Через один файл-сокет могут передавать данные несколько процессов.