Оглавление

Вопрос 1 генерация сигнала, что-то про сигналы POSIX

Сигналы обеспечивают простейшую форму межпроцесного взаимодействия, позволяя уведомлять процесс или группу процессов о наступлении некоторого события. Обеспечивают механизм вызова некоторой процедуры по наступлению некоторого события. Две фазы: генерация или отправление сигнала и его доставка и обработка. Сигнал считается когда процесс, которому был отправлен сигнал, получает его и выполняет его обработку. В промежутке между этими двумя моментами сигнал ожидает доставки.

Отправление сигнала

Ядро генерирует и отправляет процессу сигнал в ответ на ряд событий, которые могут быть вызваны самим процессом, другим процессом, прерыванием или какими!либо внешними событиями. Можно выделить основные причины отправки сигнала:

- особые ситуации (например, деление на ноль)
- теминальные прерывания (Нажатие некоторых клавиш терминала, например CTRL + C вызывает отправление сигнала текущему процессу, свзанному с терминалом)
- другие процессы (например вызов kill вызывает отправку другому процессу указанного сигнала)
- управление заданиями (Командные интерпретаторы, поддерживающие систему управления заданиями, используют сигналы для манипулирования фоновым и текущими задачами.)
- квоты (Когда процесс превышает выделенную ему квоту вычислительных ресурсов или ресурсов фай!
- ловой системы, ему отправляется соответствующий сигнал.)
- уведеомления
- алармы (Если процесс установил таймер, ему будет отправлен сигнал, когда значение таймера станет равным нулю.)

Доставка и обработка сигнала

Процесс может использовать обработчик сигнала по умолчанию, может определить собственный обработчик сигнала, может игнорировать сигнал (для установления реакции на сигнал — signal(), SIG_IGN — игнор, SIG_DFL — умолчание; SIGKILL и SIGSTOP для них только умолчание). Поцесс не получит сигнал пока он не будет выбран планировщиком и ему не будут выделены выч. ресурсы, следовательно не порядок с точностью (если использовать сигналы для таймера). Ядро (от имени процесса) вызывает процедуру issig(), которая определяет, существуют ли сигналы адресованные данному процессу, если обработчик сигнала переопределн, то вызов sendsig(). Вывод — сигналы отстой: не очень информативны и ресрсоёмки.

Вопрос 2 генерация сигнала с клавиатуры

Допустим, пользователь, работая за терминалом, нажимает клавишу (DEL, CTRL +C) прерывания . Нажатие любой клавиши вызывает аппаратное прерывание (например, прерывание от последовательного порта), а драйвер терминала при обработке этого прерывания определяет, что была нажата специальная клавиша, генерирующая сигнал, и отправляет текущему процессу, связанному с терминалом, сигнал SIGINT. Когда процесс будет выбран планировщиком и запущен на выполнение, при переходе в режим задачи он обнаружит поступление сигнала и обработает его. Если же в момент генерации сигнала терминальным драйвером процесс, которому был адресован сигнал, уже выполнялся (т. е. был прерван обработчиком терминального прерывания), он также обработает сигнал при возврате в режим задачи после обработки прерывания.

Впрос 3 Генерация сигнала одним потоком с целью завершения другова потока.

```
#include <stdio.h>
#include <pthread.h>
#include <signal.h>
int flag=0,t2tokill=0;
pthread_t t1,t2;
int nit1count=0;
int nit2count=0;

void   *thread1(void);
void   *thread2(void);

void   *thread1(void)
{
    while(1){
        printf("\nthread1 - called %d times",nit1count++);
        sleep(5);
        if(flag==0)
        { pthread_kill(t2, SIGUSR1);
         flag=1; }
    }
}

void   *thread2(void)
{

    while(!t2tokill){
        printf("\nthread2 -called %d times",nit2count++);
        sleep(1);
    }
printf("\nthread2 is killed");
}


void   *SigHandlr(int s)
{
t2tokill=1;
}

main()
{
    signal(SIGUSR1,SigHandlr);
    pthread_create(&t1,NULL,thread1,NULL);
    pthread_create(&t2,NULL,thread2,NULL);
    pthread_join(t1,NULL);
    pthread_join(t2,NULL);
}
```

На пятой секунде работы приложения удаляется вторая нить при получении сигнала SIGUSR1.

ы

Вопрос 4 Жизненный цикл сигнала. Описание каждой фазы. Системные вызовы в каждой фазе жизненного цикла.

Две фазы:

- отправление сигнала (генерация)

Возможно сигнал отправлен системным вызовом kill()

- доставка и обработка сигнала

Два вызова:

issig() - вызывает ядра от имени процесса для определения есть ли сигналы адресованные процессу.

Функция issig вызывается ядром в трех случаях:

1. Непосредственно перед возвращением из режима ядра в режим задачи после обработки системного вызова или прерывания.
2. Непосредственно перед переходом процесса в состояние сна с приори!
тетом, допускающим прерывание сигналом.
3. Сразу же после пробуждения после сна с приоритетом, допускающим прерывание сигналом.

Если процедура issig() обнаруживает ожидающие доставки сигналы, ядро вызывает функцию доставки сигнала, которая выполняет действия по умолчанию или вызывает специальную функцию sendsig() запускающую обработчик сигнала, зарегистрированный процессом. Функция возвращает процесс в режим задачи, передает управление обработчику сигнала, а затем восстанавливает контекст процесса для продолжения прерванного сигналом выполнения.

Вопрос 5 Подробная мотивайция ресурсоемкости взаимодействия посредственных сигналов на примере IPC Posix.

Ресурсоёмко потому что требуется много переключений из пользовательского режима в режим ядра из-за системных вызовов (issig, sendsig).

Доставка сигнала происходит после того, как ядро от имени процесса вызывает системную процедуру issig() которая проверяет, существуют лиожидающие доставки сигналы, адресованные данному процессу. Функция issig вызывается ядром в трех случаях:

1. Непосредственно перед возвращением из режима ядра в режим задачи после обработки системного вызова или прерывания.
2. Непосредственно перед переходом процесса в состояние сна с приори!
тетом, допускающим прерывание сигналом.
3. Сразу же после пробуждения после сна с приоритетом, допускающим прерывание сигналом.

Если процедура issig() обнаруживает ожидающие доставки сигналы, ядро вызывает функцию доставки сигнала, которая выполняет действия по умолчанию или вызывает специальную функцию sendsig() запускающую обработчик сигнала, зарегистрированный процессом. Функция возвращает процесс в режим задачи, передает управление обработчику сигнала, а затем восстанавливает контекст процесса для продолжения прерванного сигналом выполнения.

В качестве бонуса можно упоминуть про ненадежность (по временному критерию) сигналов. Сигнал будет доставлен только после выбора процесса планировщиком и выделения ему ресурсов, что нивилирует асинхронный характер сигнала.

Вопрос 6 Причины генерации сигналов.Примеры.Диспозиция. Пример кода С++ запуска

Причины:

- особые ситуации (например, деление на ноль)
- теминальные прерывания (Нажатие некоторых клавиш терминала, например CTRL + C вызывает отправление сигнала текущему процессу, свзанному с терминалом)
- другие процессы (например вызов kill вызывает отправку другому процессу указанного сигнала)
- управление заданиями (Командные интерпретаторы, поддерживающие систему управления заданиями, используют сигналы для манипулирования фоновым и текущими задачами.)
- квоты (Когда процесс превышает выделенную ему квоту вычислительных ресурсов или ресурсов фай!
- ловой системы, ему отправляется соответствующий сигнал.)
- уведеомления
- алармы (Если процесс установил таймер, ему будет отправлен сигнал, когда значение таймера станет равным нулю.)

Диспозиции:

- игнор - signal(номер сигнала, SIG_DFL)
- использование обработчика по умолчанию - (номер сигнала, SIG_IGN)
- использование собственного обработчика – (номер сигнала, имя функции обработчика)

Посылка сигнала — kill (PID, номер сигнала)

Что имеется ввиду под примером кода запуска ХЗ.

Вопрос 7 Системные вызовы назначение и способы реализации в разных ОС. Синхронные и асинхронные системные вызовы.

Системные вызовы – возможность обращения к кодовым сегментам ОС из приложений. В результате обеспечивается возможность перехода в привилегированный режим.

Достоинства:

1) Высокая скорость вызова процедур ОС;
2) Универсальность обращения (одинаковые обращения для всех аппаратных платформ);
3) Контроль со стороны ОС за корректностью выполнения системных вызовов;
4) Легкая масштабируемость системных вызовов (22-32 для большинства ОС).

2 способа передачи управления при использовании программного прерывания:

- децентрализованный – каждому системному вызову соответствует свой вектор прерывания;
- централизованный – в большинстве ОС основан на использовании диспетчера системных вызовов.

Децентрализованный способ является более быстрым, но менее гибким (вектор прерывания указывается в запросе). Централизованный способ – всегда используется один и тот же вектор прерывания.

Пред выполнением программы прерывания приложение передает в ОС номер системного вызова. Номер является индексом в таблице системных вызовов (табличный способ реализации системных вызовов). Аргументы системных вызовов в таком случае передаются через стек. При выполнении системного вызова производится копирование пользовательского стека. Далее производится чтение аргументов из системного стека. Такой механизм используется в большинстве ОС. Достоинство такой реализации – легко расширяется набор системных вызовов.

Системные вызовы могут быть организованы как синхронно, так и асинхронно, в некоторых ОС – это как режим.

Синхронный вызов – более простой способ: процесс приостанавливается и ожидает окончания системного вызова (блокируется до завершения системной процедуры). После завершения системного вызова планировщик переводит процесс в очередь готовых процессов. Как только возобновится выполнение процесса, он сможет воспользоваться результатами системного вызова.

Асинхронный вызов – блокировки процесса не происходит, выполняется определенный ряд действий, а затем передается управление тому процессу, который выполнялся (такой способ удобен при обращении к ПУ – сохраняется возможность продолжения вычислений исходным процессом).

При синхронном вызове не нужно отслеживать момент завершения системной процедуры – эту функцию берет на себя планировщик, но в этом случае менее экономно расходуются ресурсы.

В универсальных ОС используется блокирующие (синхронные) системные вызовы. В микроядерных – асинхронные системные вызовы – это тенденция.

3. *Функции генерации сигналов (POSIX): названия, аргументы, принцип работы.*

функции генерации сигналов. Для генерации сигналов в Unix предусмотрены две функции – kill(2) и raise(3). Первая функция предназначена для передачи сигналов любым процессам, к которым владелец данного процесса имеет доступ, а с помощью второй функции процесс может передать сигнал самому себе. Как это обычно принято в мире Unix, семантика вызова функции kill()совпадает с семантикой одноименной команды ОС. У функции kill()два аргумента –PID процесса-приемника и номер передаваемого сигнала. С помощью функции kill()как и с помощью одноименной команды можно передавать сообщения не только конкретному процессу, но и группе процессов.

Таблица 2 демонстрирует поведение функции kill()в зависимости от значения PID:

PID > 1	Сигнал посылается процессу с соответствующим PID.
PID == 0	Сигнал посылается всем процессам из той же группы что и процесс-источник.
PID < 0	Сигнал посылается всем процессам, чей идентификатор группы равен абсолютному значению PID.
PID == 1	Сигнал посылается всем процессам системы.

Вопрос 8 Программные ресурсы, реентерабельные и повторно входимые модули, мотивация их применения при проектировании ОС и её компонентов

программные модули, т.к. именно они могут рассматриваться как программные ресурсы. Программные модули могут быть однократно и многократно или повторно используемыми. Однократно используемыми называют такие программные модули, которые могут быть правильно выполнены только один раз. Это означает, что в процессе выполнения они могут испортить себя: либо повреждается часть кода, либо – исходные данные, от которых зависит ход вычислений. Однократно используемые программные модули являются неделимым ресурсом. Обычно их не распределяют как ресурс системы, они используются только на этапе загрузки ОС. При этом собственно двоичные файлы, которые хранятся на системном диске и в которых записаны эти модули, не портятся, и могут быть повторно использованы при следующем запуске ОС.

Повторно используемые программные модули могут быть непривилегированными, привилегированными и реентерабельными.

Привилегированные программные модули работают в так называемом привилегированном режиме, т.е. при отключенной системе прерываний, так, что никакие внешние события не могут нарушить порядок вычислений. В результате программный модуль выполняется до своего конца, после чего он может быть вновь вызван на исполнение из другой задачи (другого вычислительного процесса). Со стороны по отношению к вычислительным процессам, которые попеременно в течение срока жизни вызывают некоторый привилегированный программный модуль, такой модуль будет выступать попеременно разделяемым ресурсом.

Непривилегированные программные модули – это обычные программные модули, которые могут быть прерваны во время работы. В общем случае их нельзя считать разделяемыми, потому что если после прерывания выполнения такого модуля, исполняемого в рамках одного вычислительного процесса, запустить его еще раз по требованию другого вычислительного процесса, то промежуточные результаты для прерванных вычислений могут быть потеряны.

В противоположность этому, реентерабельные программные модули допускают повторное многократное прерывания своего исполнения и повторный их запуск из других задач. Для этого реентерабельные программные модули должны быть созданы таким образом, чтобы было обеспечено сохранение промежуточных вычислений для прерываемых вычислений и возврат к ним, когда вычислительный процесс возобновляется с прерванной ранее точки. Это может быть реализовано двумя способами: с помощью статических и динамических методов выделения памяти под сохраняемые значения.

Кроме реентерабельных программных модулей существуют еще повторно входимые. Это программные модули, которые допускают многократное параллельное использование, но в отличие от реентерабельных их нельзя прерывать.

Вопрос 9 Мультипроцессорная обработка.Конфигурация

. Мультипроцессированные ОС.

ОС может быть однопроцессорной или многопроцессорной.

Симметричные ВС:

В ВС входит набор однородных процессоров (одинаковая конструкция). Такую систему можно наращивать по вертикали до 4…8 процессоров.

Ассиметричные ВС:

Также в ВС может быть набор неоднородных процессоров – каждый со своими свойствами. Для вычислительных систем это лучше, но для ОС сложнее.

Симметричные ОС – только для симметричной аппаратуры. Все функции ОС распределены между процессорами. Идеальный вариант – динамическое распределение. Такие системы из-за своей сложности часто ненадежны.

Ассиметричные ОС – для любой аппаратуры. Один из вариантов ОС, ассиметричной по мультипроцессированию – когда один процессор главный, и он распределяет задачи между остальными. Это надежно.

Вопрос 10 Совместимость и множественные прикладные среды.Реализация множественных прикладных сред.

Совместимость ОС – свойство, позволяющее выполнять приложения, написанные для других ОС.

- Двоичная совместимость – исполняемый файл. Определяется архитектурой процессора, совпадение API. Внутренняя структура файла должна соответствовать структуре, используемой в данной ОС. При несовпадении структуры необходима эмуляция двоичного кода. Эмуляция используется в системах кросс-разработки.
- На уровне исходных кодов – совместимость на уровне компиляторов, совместимость библиотек, системных вызовов.

Эмуляция двоичного кода используется в системах кросс-разработки – системах, предназначенных для разработки программ в 2х машинной конфигурации.

Состав системы кросс-разработки:

- Средства редактирования
- Средства компиляции
- Средства отладки

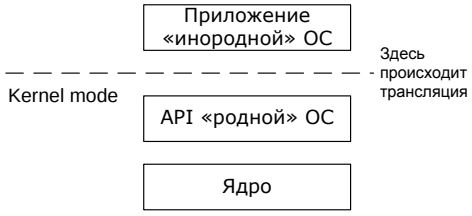
Все это находится на инструментальной машине, а в готовом виде передается на целевую. Использование кросс-средств: системы программирования МК (Intel, Atmel и др.). ОС WinCE, PalmOS. Такие ОС включают в себя набор компиляторов и ассемблеров на инструментальной машине под ее ОС, библиотеки, выполняющие большую часть функций целевой ОС, средства отладки.

Прикладные среды. Способы реализации

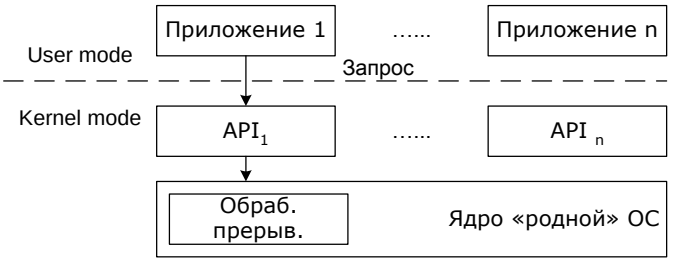
Альтернатива эмуляции – прикладные среды. В состав каждой прикладной среды входит набор функций интерфейса API. Прикладные среды имитируют обращение к библиотечным функциям инородной среды. На самом деле обращение происходит к внутренним библиотекам. Это называется трансляцией библиотек. Чтобы программа, написанная под одной ОС работала под другой, необходимо обеспечить бесконфликтное взаимодействие способов управления процессами в разных ОС.

Способы реализации прикладных сред

1. В данном случае прикладная программная среда реализована как верхний слой ядра родной ОС.



2.



Отдельный слой (API). Прикладная программная среда на основе нескольких равноправных API.

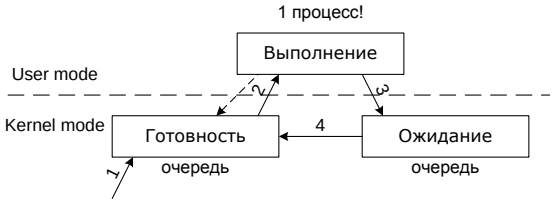
3. Реализуется по микроядерному принципу (Windows NT).

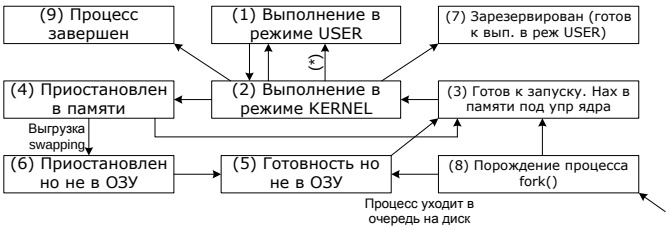
1) Процесс сосстояния жизненный цикл многозадачных ОС.Утилиты статуса,информационной структуры.

Жизненный цикл процесса:

- Выполнение – активное состояние процесса, когда он занимает процессорный ресурс
- Ожидание – пассивное состояние – процесс заблокирован по внутренним причинам: ожидание данных, прерывания и т.д. (связанные с логикой самого процесса)

- Готовность – пассивное состояние – процесс заблокирован по внешним причинам (ожидание события, прерывания и т.д.)





8->3 если есть место в ОЗУ, если нет, то 8->5
4, 6 – ожидание
1, 2 – выполнение
3, 5, 7 - готовность

Вопрос 11 Интерфейс прикладного программирования. Функции API. Способы реализации API

Интерфейсы ОС.

Интерфейс ОС – это прикладная система программирования.

1. Пользовательский интерфейс – реализуется с помощью специальных программных модулей, которые транслируют запросы пользователя на специальном командном языке в запросы к ОС. Совокупность таких модулей называется интерпретатором. Он выполняет лексический и синтаксический анализ и либо сам выполняет команду, либо передает ее API.
2. API – предназначен для предоставления прикладным программам ресурсов ОС и реализации других функций. API описывает совокупность функций, процедур, принадлежащих ядру и надстройкам ОС. API использует системные программы как в составе ОС, так и за ее пределами, используя прикладные программы посредством среды программирования.

Вопрос 12 ОС мобильных устройств. Классификации,примеры.Характеристика с точки зрения основных типов архитектуры ОС.

— мобильный телефон с полноценной операционной системой (Symbian OS,Windows Mobile, Palm OS, GNU/Linux, Android и др
WinMob
Характеристики

Windows Mobile для Pocket PC (сенсорных устройств) в стандартной поставке включает следующие особенности:

- начальный экран «Today» (русск. *Сегодня*) отображает текущую дату, информацию о владельце, предстоящие встречи, новые сообщения и задачи. Начиная с WM 6.5 экран имеет название «Home screen»;
- кнопка «Пуск», находящаяся в верхнем баре, раскрывает меню со списком программ и служебными ссылками как в настольной версии Windows;
- панель задач отображает текущее время, вариант звукового профиля и заряд батареи;
- мобильный браузер Internet Explorer Mobile, основанный на настольной версии IE[12];
- Windows Media Player для Windows Mobile;
- интеграция с сервисами Windows Live;
- клиент для PPTP VPN;
- функция Internet Connection Sharing (ICS), позволяющей делить подключение к Интернет с настольным компьютером через USB или Bluetooth;
- файловая система и структура папок аналогичны таковым в Windows 9x/Windows NT;
- многозадачность.

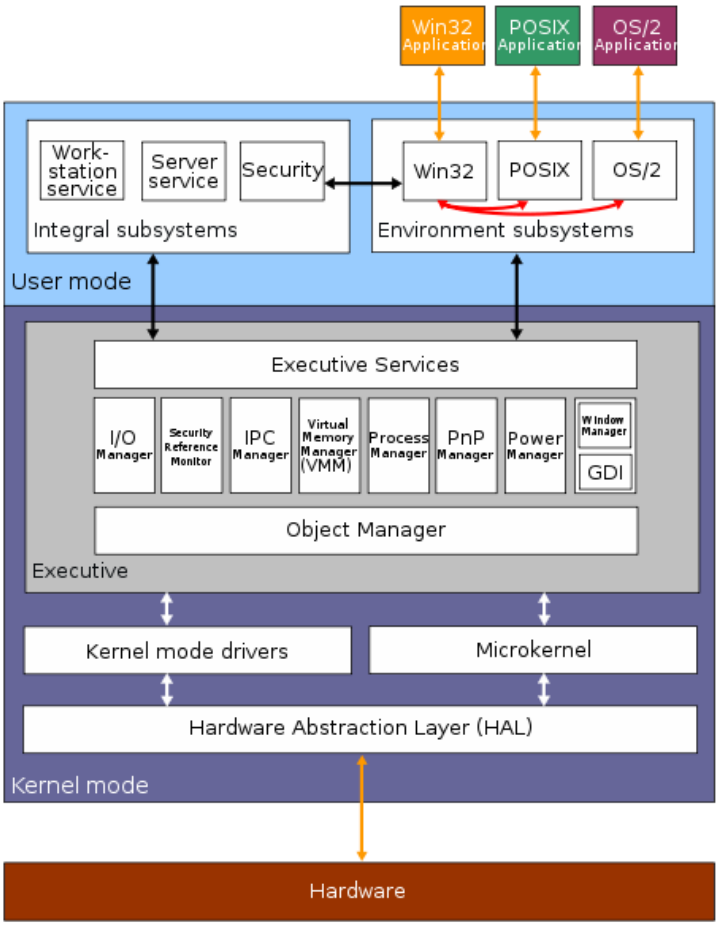
Основные отличия *smartphone-версии* от Pocket PC:

- интерфейс, адаптированный под телефонный форм-фактор, рассчитан на кнопочное управление устройством, из-за этого программы Pocket PC и Smartphone не совместимы друг с другом;
- начальный экран «Today» отображает (в порядке сверху-вниз) ярлыки на недавно запущенные приложения, текущую дату, предстоящие встречи, звуковой профиль и новые сообщения;
- кнопка «Пуск» расположена в нижней панели;
- список программ отображается в отдельном экране;
- отсутствует экранная клавиатура, вследствие наличия штатной (как мобильной, так и QWERTY).

Вопрос 13 Анализ архитектуры и модели функционирования ОС линейки Windows.

Архитектура **Windows NT** имеет модульную структуру и состоит из двух основных уровней — компоненты, работающие в режиме пользователя и компоненты режима ядра. Программы и подсистемы, работающие в режиме пользователя имееют ограничения на доступ к системным ресурсам. Режим ядра имеет неограниченный доступ к системной памяти и внешним устройствам. Ядро системы NT называют гибридным ядром или макроядром. Архитектура включает в себя само ядро, уровень аппаратных абстракций (HAL), драйверы и ряд служб (Executives), которые работают в режиме ядра (Kernel-mode drivers) или в пользовательском режиме (User-mode drivers)[1][2].

Пользовательский режим Windows NT состоит из подсистем, передающих запросы ввода/вывода соответствующему драйверу режима ядра посредством менеджера ввода/вывода. Есть две подсистемы на уровне пользователя: подсистема окружения (запускает приложения, написанные для разных операционных систем) и интегрированная подсистема (управляет особыми системными функциями от имени подсистемы окружения). Режим ядра имеет полный доступ к аппаратной части и системным ресурсам компьютера. И также предотвращает доступ к критическим зонам системы со стороны пользовательских служб и приложений.



Вопрос 14 Модели функционирования ОС

Вопрос 15 Мотивация выбора архитектуры для построения ОС реального времени.

Вопрос 16 Критерии показателей ОС различных классов.(1 лекция)

Вопрос 17 Мультипрограммированная многозадачность. Реализация современных ОС.

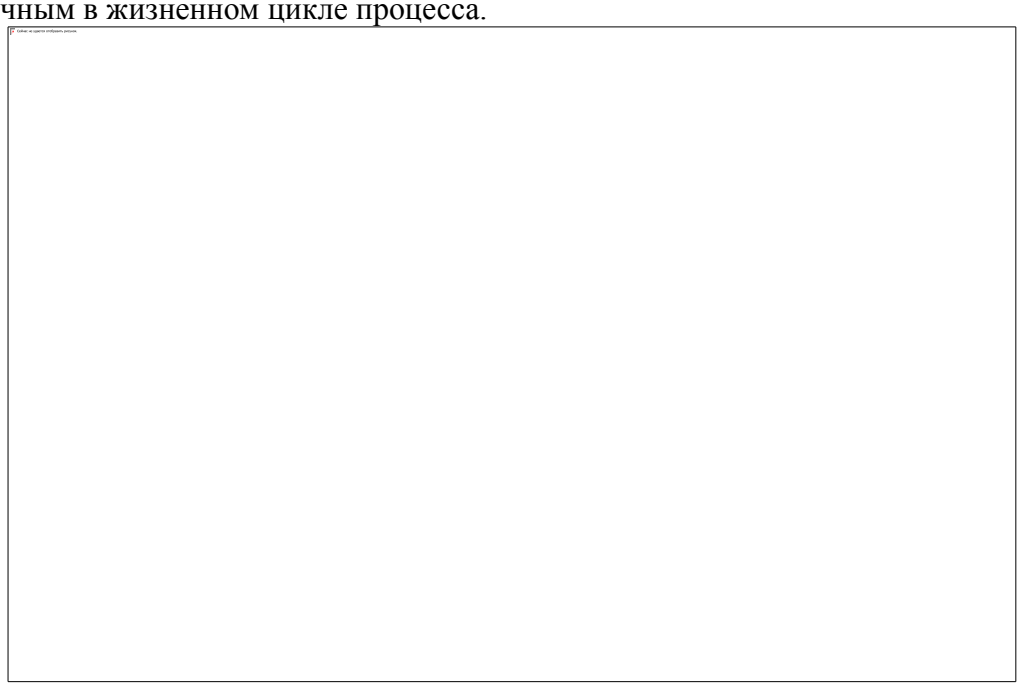
Вопрос 18 Задача планирования и диспетчиризации.Типыпланировщиков.Планирование в системах реального времени.Разделение времени.

Вопрос 19 Процесс сосстояния жизненный цикл многозадачных ОС.Утилитыстатуса,информационной структуры.

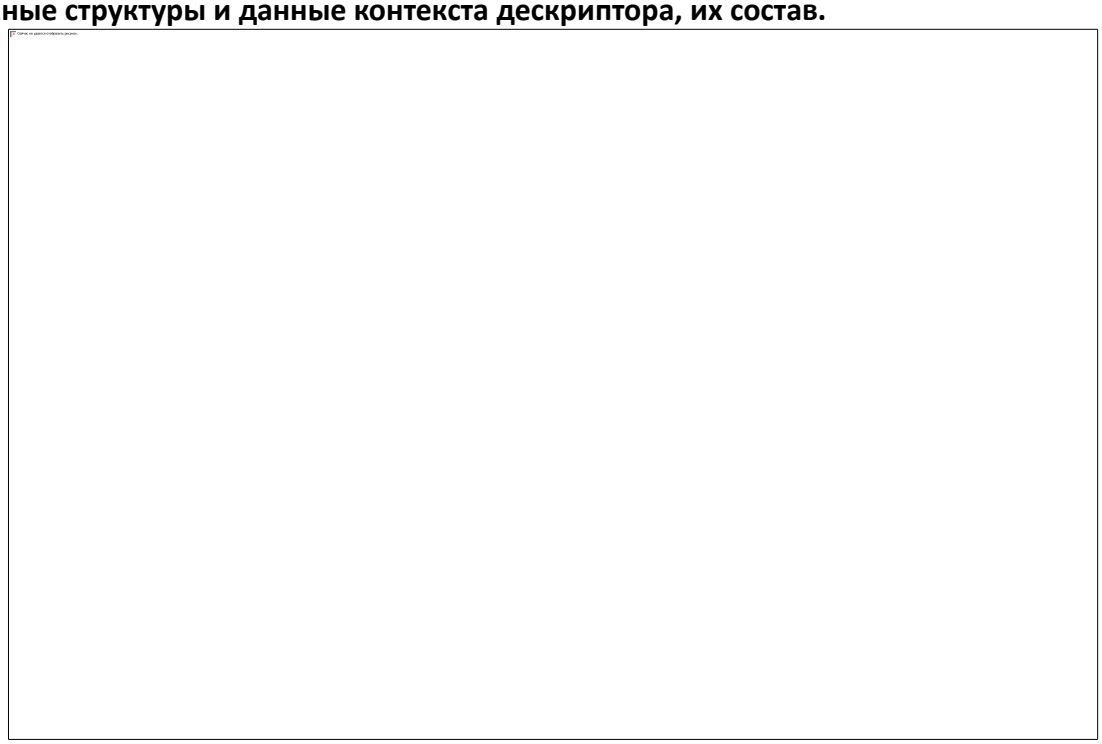
1. Процесс выполняется в режиме задачи. При этом процессором выполняются прикладные инструкции данного процесса.
2. Процесс выполняется в режиме ядра. При этом процессором выполняются системные инструкции ядра операционной системы от имени процесса.
3. Процесс не выполняется, но готов к запуску, как только планировщик выберет его (состояние Процесс находится в очереди на выполнение и обладает всеми необходимыми ему ресурсами, кроме вычислительных. Процесс находится в состоянии сна (asleep), ожидая недоступного в данный момент ресурса, например завершения операции ввода/вывода.Процесс возвращается из режима ядра в режим задачи, но ядро прерывает его и производит переключение контекста для запуска более высокоприоритетного процесса.

Процесс только что создан вызовом fork() и находится в переходном состоянии: он существует, но не готов к запуску и не находится в состоянии сна.

7. Процесс выполнил системный вызов exit() и перешел в состояние зомби (zombie, defunct). Как такового процесса не существует, но остаются записи, содержащие код возврата и временную статистику его выполнения, доступную для родительского процесса. Это состояние является конечным в жизненном цикле процесса.



Впрос 20 Контекст дискрипторапроцесса,структура и иерархия контекста UNIX подобных ОС.Системные структуры и данные контекста дескриптора, их состав.



+ читай вопрос 21

Вопрос 21. Структуры процесса на примере Unix. Основные системные структуры данных процесса.

Структура данных proc – запись таблицы процессов, которая всегда хранится в оперативной памяти.
Char p_stat Состояние процесса.
char p_pri Текущий приоритет процесса.
Unsigned int Флаги, определяющие дополнительную информацию о состояние процесса
unsigned short p_uid UID процесса
unsigned short p_suid EUID процесса
int p_sid идентификатор сеанса
short p_pgrp Идентификатор группы процессов (равен идентификатору группы процессов)
short p_pid Идентификатор процесса
short p_ppid Идентификатор родительского процесса.
sigset_t p_sig Сигналы ожидающие доставки
caddr_t p_ldt Указатель на таблицу управления памятью процесса
unsigned int p_utbl[] Массив записей таблицы страниц для u-area
short p_xstat Код возврата, передаваемый родительскому процессу
структура u-area содержит дополнительные данные о процессе, необходимые ядру только при выполнение процесса (содержит информацию об открытых файловых дискрипторах, диспозицию сигналов, статистику выполнения процесса, сохраненные значения регистров, также содрежсит стек ядра, который используется вместо обычного при выполнение процесса в режиме ядра).

Вопрос 22 Обращение к файлам из процесса. Этапы системной таблицы идентификации файлов и процесса.

Вопрос 23.Идентификаторы файла,файловые дескрипторы,индексные дискрипторы,файловые ссылки укажите назначение и сферу применения соответствующие каждому понятию

индексный дескриптор — inode. Содержит информацию о файле , необходимую для обработки данных, т.е. метаданные файла. Используется файловой системой.

файловый дескриптор — индекс в таблице файловых дескрипторов процесса. Представляет собой неотрицательное число, возвращаемое системными вызовами, такими как open(), pipe(). Файловый дискриптор связан сполями u_ofile и u_pofile структуры user и таким образом обеспечивает доступ к соответсвующему элементу файловой таблицы(структуре данных file).

Файловые ссылки — 2 вида. Жесткие ссылки и символьные. Жесткие ссылаются на vnode. Символьные — это путь к жесткой ссылке. Для связи vnode и содержимого каталогов ФС.Связь имя файла ->inode.

Впрос 24 Трансляция имен. Где и как применяется.Примеры системных функций.

Прикладные процессы, запрашивая услуги файловой системы, обычно имеют дело с именем файла или файловым дескриптором, полученным в результате определенных системных вызовов. Однако ядро системы для работы с файлами использует не имена, а индексные дескрипторы. Таким образом необходима трансляция имени файла, передаваемого в качестве аргумента системному вызову open(), в номер соответствующего vnode.
Имя файла может быть абсолютным или относительным. Абсолютное — путь из корневого каталога. Относительное — из текущего.
Каждый процесс адресует эти каталоги двумя полями структуры u_area:
u_cdir укзатель на vnode текущего каталога
u_rdir указатель на vnode корневого каталога
В зависимости от имени файла трансляция начинается с vnode, адресованного либо полем u_cdir либо полем u_rdir. Трансляция имени осуществляется покомпонентно , при этом для vnode текущего каталога вызывается соответствующая ему опреация vn_lookup(), в качестве аргумента которой передается имя следующего компонента. В результате операции возвращается vnode, соответсвующий искомому компоненту.

Вопрос 25 Структуры каталогов и форматы записи каталогов для разных ФС.(примеры)

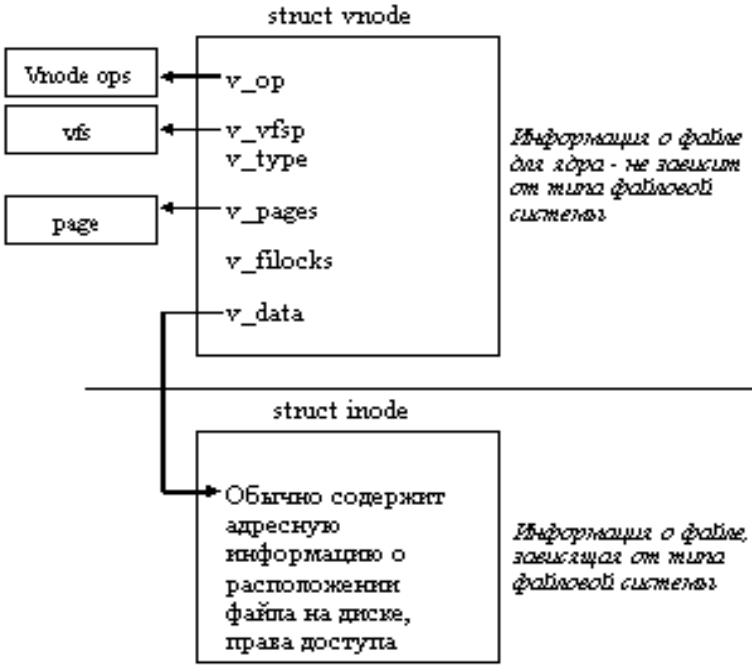
Каталог FFS. Представлен следующей структурой:
d_ino – номер inode
d_reclen – длина записи
d_namlen – длина имени файла
d_name[] - имя файла
Имя файла имеет переменную длину, дополненную нулями до 4-байтной границы. При удалении имени файла принадлежащая ему запись присоединяется к предыдущей.

Вопрос 26. Структуры данных ФС Unix. Назначение, основной состав.

Система S5FS.

Работа ядра с файлами во многом основана на использовании структуры vnode, поля которой представлены на рисунке 5.8. Структура vnode используется ядром для связи файла с определенным типом файловой системы через поле v_vfsp и конкретными реализациями файловых операций через поле v_op. Поле v_pages используется для указания на таблицу физических страниц памяти в случае, когда файл отображается в физическую память (этот механизм описан в разделе, описывающем организацию виртуальной памяти). В vnode также содержится тип файла и указатель на зависимую от типа файловой системы часть описания характеристик файла - структуру inode, обычно содержащую адресную информацию о расположении файла на носителе и о правах доступа к файлу. Кроме этого, vnode используется ядром для хранения информации о блокировках (locks), примененных процессами к отдельным областям файла.

Ядро в своих операциях с файлами оперирует для описания области файла парой vnode, offset, которая однозначно определяет файл и смещение в байтах внутри файла.



s_type – тип
s_fsize – размер файловой системы
s_lsize – размер массива индексных дескрипторов
s_tfree – число свободных блоков
s_tinode – число свободных inode
s_fmode – флаг модификации
s_fnonly – флаг режима монтирования
размер логического блока
список номеров свободных inode
список адресов свободных блоков

inode ->dinode Формат: | тип файла | Snid | sgid | - | rwx | rwx | rwx|
di_mode
di_nlinks – число ссылок на файл
di_nid, di_gid - идентификаторы пользователей и группы

di_size – размер файла в байтах
di_atime – время последнего доступа к файлу
di_ntime — время последней модификации файла
di_ctime — время последней модификации inode
di_addr[n] -

При каждом открытии процессом файла ядро создает в системной области памяти новую структуру типа file, которая, как и в случае традиционной файловой системы s5, описывает как открытый файл, так и операции, которые процесс собирается производить с файлом (например, чтение). Структура file содержит такие поля, как:
flag - определение режима открытия (только для чтения, для чтения и записи и т.п.);
struct vnode * f_vnode - указатель на структуру vnode (заменивший по сравнению с s5 указатель на inode);
offset - смещение в файле при операциях чтения/записи;
struct cred * f_cred - указатель на структуру, содержащую права процесса, открывшего файл (структура находится в дескрипторе процесса);

Вопрос 27. Функции ФС, низкоур, высокоур форматирование, разделы.

Основные функции любой файловой системы нацелены на решение следующих задач:

- именование файлов;
- программный интерфейс работы с файлами для приложений;
- отображения логической модели файловой системы на физическую организацию хранилища данных;
- организация устойчивости файловой системы к сбоям питания, ошибкам аппаратных и программных средств;
- содержание параметров файла, необходимых для правильного его взаимодействия с другими объектами системы (ядро, приложения и пр.).

В многопользовательских системах появляется ещё одна задача: защита файлов одного пользователя от несанкционированного доступа другого пользователя, а также обеспечение совместной работы с файлами, к примеру, при открытии файла одним из пользователей, для других этот же файл временно будет доступен в режиме «только чтение».

Низкоуровневое форматирование (англ. Low level format) — операция, в процессе которой на магнитную поверхность диска наносятся т. н. сервометки — служебная информация, которая используется для позиционирования головок жёсткого диска. Выполняется в процессе изготовления жёсткого диска, на специальном оборудовании, называемом серворайтером.

Раздел (англ. partition) — часть долговременной памяти жёсткого диска, выделенная для удобства работы, и состоящая из смежных блоков. Всего разделов на физическом диске может быть не более 4-х. Из них не более одного расширенного(дополнительного) раздела

Вопрос 28. Структура тома NTFS. Системные файлы.

Как и любая другая система, NTFS делит все полезное место на кластеры - блоки данных,используемые единовременно. NTFS поддерживает почти любые размеры кластеров- от 512 байт до 64 Кбайт, неким стандартом же считается кластер размером 4 Кбайт. Никаких аномалий кластерной структуры NTFS не имеет,поэтому на эту, в общем-то, довольно банальную тему, сказать особо нечего.

Диск NTFS условно делится на две части.Первые 12% диска отводятся под такназываемую MFT зону - пространство, в котороерастет метафайл MFT (об этом ниже). Записьк аких-либо данных в эту область невозможна.MFT-зона всегда держится пустой - этоделается для того, чтобы самый главный,служебный файл (MFT) не фрагментировался при своем росте. Остальные 88% диска представляют собой обычное пространстводля хранения файлов.

Файловая система NTFS представляет собой выдающееся достижение структуризации: каждыйэлемент системы представляет собой файл -даже служебная информация. Самый главныйфайл на NTFS называется MFT, или Master File Table -общая таблица файлов. Именно он размещается в MFT зоне и представляет собой централизованный каталог всех остальных файлов диска, и, как не парадоксально, себя самого. MFT поделен на записи фиксированного размера (обычно 1 Кбайт), и каждая запись соответствует какому либо файлу (в общемсмысле этого слова). Первые 16 файлов носят служебный характер и недоступны операционной системе - они называются метафайлами, причем самый первый метафайл -сам MFT. Эти первые 16 элементов MFT - единственная часть диска, имеющая фиксированное положение. Интересно, что вторая копия первых трех записей, длян адежности (они очень важны) хранится ровно посередине диска.

Первые 16 файлов NTFS (метафайлы) носятслужебный характер. Каждый из них отвечаетза какой-либо аспект работы системы.Преимущество настолько модульного подходазаключается в поразительной гибкости -например, на FAT-е физическое повреждение всамой области FAT фатально дляфункционирования всего диска, а NTFS можетсместить, даже фрагментировать по диску,все свои служебные области, обойдя любыенеисправности поверхности - кроме первых 16элементов MFT.

Метафайлы находятся корневом каталоге NTFSдиска - они начинаются с символа имени"$", хотя получить какую-либо информацию о них стандартными средствами сложно. Любопытно, что и для этих файлов указан вполне реальный размер — можно узнать, например, сколько операционная система тратит на каталогизацию всего вашего диска, посмотрев размер файла $MFT. В следующей таблице приведены используемые в данный момент метафайлы и их назначение.$MFT сам MFT
$MFTmirr копия первых 16 записей MFT, размещенная посередине диска
$LogFile файл поддержки журналирования (см. ниже)
$Volume служебная информация - метка тома, версия файловой системы, т.д.
$AttrDef список стандартных атрибутов файлов на томе
$. корневой каталог
$Bitmap карта свободного места тома
$Boot загрузочный сектор (если раздел загрузочный)
$Quota файл, в котором записаны права пользователей на использование дискового пространства (начал работать лишь в NT5)
$Upcase файл - таблица соответствия заглавных и прописных букв в имен файлов на текущем томе. Нужен в основном потому, что в NTFS имена файлов записываются в Unicode, что составляет 65 тысяч различных символов, искать большие и малые эквиваленты которых очень нетривиально.
Файлы
Прежде всего, обязательный элемент - запись в MFT, ведь, как было сказано ранее, все файлы диска упоминаются в MFT. В этом месте хранится вся информация о файле, за исключением собственно данных. Имя файла, размер, положение на диске отдельных фрагментов, и т.д. Если для информации

не хватает одной записи MFT, то используются несколько, причем не обязательно подряд.

Каталоги.
Каталог на NTFS представляет собойс пецифический файл, хранящий ссылки над ругие файлы и каталоги, создавая иерархическое строение данных на диске.Файл каталога поделен на блоки, каждый из которых содержит имя файла, базовые атрибуты и ссылку на элемент MFT, который уже предоставляет полную информацию обэлементе каталога. Внутренняя структура каталога представляет собой бинарное дерево. Бинарное же дерево располагает имена файлов таким образом, чтобы поиск файла осуществлялся более быстрым способом- с помощью получения двухзначных ответов на вопросы о положении файла. Вопрос, накоторый бинарное дерево способно дать ответ, таков: в какой группе, относительно данного элемента, находится искомое имя -выше или ниже?
Поддерживает журналирование и сжатие.

Вопрос 29. Каталог, файловые записи HPFS.

HPFS-тома используют размер сектора 512 байтов и имеют максимальный размер 2199Gb. HPFS том имеет очень небольшое количество фиксированных структур. Секторы 015 тома (BootBlock, имя тома, 32 бита ID, дисковая программа начальной загрузки). Начальная загрузка относительно сложна (в стандартах МСДОС) и может использовать HPFS в ограниченном режиме

Сектора 16 и 17 известны как SuperBlock и SpareBlock соответственно. SuperBlock изменяется только при помощи утилит. Он содержит указатели свободного пространства, список плохих блоков, полосу блока каталога, и корневую директорию. Он также содержит дату, соответствующую последней проверке и восстановлению утилитой CHKDSK/F. SpareBlock содержит различные флажки и указатели которые будут обсуждаться позже; Он изменяется, хотя нечасто, при работе системы.

Остаток диска разделен на 8MB полосы. Каждая полоса имеет собственный список свободного пространства, где биты представляют каждый сектор. Бит 0 если сектор использован 1 если сектор доступен. Списки размещаются в начале или хвосте списка.

Одна полоса, размещенная в "центре" диска, называется полосой блока каталога и обрабатывается специально. Обратите внимание, что размер полосы зависит от текущей реализации и может изменяться в более поздних версиях файловой системы.

Файлы и Fnodes

Каждый каталог или файл в HPFS-томе закрепляется за фундаментальным объектом файловой системы, называемым Fnode (произносится "eff node"). Каждый Fnode занимает одиночный сектор и содержит управляющую информацию, хронологию доступа, расширенные атрибуты и списки управления доступом, длину и первые 15 символов имени, и структуру распределения. Fnode всегда находится рядом с каталогом или файлом, который он представляет.

Структура распределения в Fnode может принимать несколько форм, в зависимости от размера каталога или файлов. HPFS просматривает файл как совокупность одного или более секторов. Из прикладной программы это не видно; файл появляется как непрерывный поток байтов.

Каталоги в HPFS, как и в FAT, образуют древовидную структуру. Но при этом внутри каталога HPFS строит сбалансированное дерево (B*-Tree) на основе имён файлов для быстрого поиска файла по имени внутри каталога.

Вопрос 30.Структура тома FAT. Формат записи каталога в FAT таблице.

В файловой системе FAT смежные секторы диска объединяются в единицы, называемые кластерами. Количество секторов в кластере может быть равно 1 или степени двойки (см. далее). Для хранения данных файла отводится целое число кластеров (минимум один), так что, например, если размер файла составляет 40 байт, а размер кластера 4 кбайт, реально занят информацией файла будет лишь 1% отведенного для него места. Для избежания подобных ситуаций целесообразно уменьшать размер кластеров, а для сокращения объема адресной информации и повышения скорости файловых операций – наоборот. На практике выбирают некоторый компромисс. Так как емкость диска вполне может и не выражаться целым числом кластеров, обычно в конце тома присутствуют т.н. surplus sectors – «остаток» размером менее кластера, который не может отводиться ОС для хранения информации.

Пространство тома FAT32 логически разделено на три смежные области:

Зарезервированная область. Содержит служебные структуры, которые принадлежат загрузочной записи раздела (Partition Boot Record – PBR, для отличия от Master Boot Record – главной загрузочной записи диска; также PBR часто некорректно называется загрузочным сектором) и используются при инициализации тома;

Область таблицы FAT, содержащая массив индексных указателей ("ячеек"), соответствующих кластерам области данных. Обычно на диске представлено две копии таблицы FAT в целях надежности;

Область данных, где записано собственно содержимое файлов – т.е. текст текстовых файлов, кодированное изображение для файлов рисунков, оцифрованный звук для аудиофайлов и т.д. – а также т.н. метаданные – информация относительно имен файлов и папок, их атрибутов, времени создания и изменения, размеров и размещения на диске.

В FAT12 и FAT16 также специально выделяется область корневого каталога. Она имеет фиксированное положение (непосредственно после последнего элемента таблицы FAT) и фиксированный размер в секторах.

Каталог FAT (папка, директория) является обычным файлом, помеченным специальным атрибутом. Данными (содержимым) такого файла в любой версии FAT является цепочка 32-байтных файловых записей (записей каталога). Директория не может штатно содержать два файла с одинаковым именем. Если программа проверки диска обнаруживает искусственно созданную пару файлов с идентичным именем в одном каталоге, один из них переименовывается.

При создании каталога для него «пожизненно» выставляется DIR_FileSize = 0. Размер содержимого каталога определяется простым следованием по цепочкам кластеров до метки End Of Chain. Размер самого каталога лимитируется файловой системой в 65 535 32-байтных записей (т.е. записи каталога в таблице FAT не могут занимать более 2Мб). Это ограничение призвано ускорить операции с файлами и позволить различным служебным программам использовать 16 битное целое (WORD) для подсчета количества записей в директории. (Как следствие, возникает теоретическое ограничение на количество файлов в каталоге – 65 535 при условии, что все имена файлов следуют стандарту 8.3). Каталогу отводится один кластер области данных (за исключением случая, если это корневой каталог FAT12/FAT16) и полям DIR_FstClusHI / DIR_FstClusLO присваивается значение номера этого кластера. В таблицу FAT для записи, соответствующей этому кластеру, помещается метка EOC, а сам кластер забивается нулями. Далее создаются два специальных файла, без которых директория FAT считается поврежденной (первые две 32-байтных записи в области данных кластера) – файлы нулевого размера “dot” (идентификатор каталога) и “dotdot” (указатель на родительский каталог) с именами “.” (точка) и “..” (две точки) соотв. Штампы даты-времени этих файлов приравниваются значениям для самого каталога на момент создания и не обновляются при изменениях каталога. Поля DIR_FstClusHI / DIR_FstClusLO файла «.» содержат значение номера содержащего его кластера, а файла «..» – номера первого кластера каталога, содержащего данный. Таким образом, файл «.» отсылает к самому каталогу, а файл «..» – к начальному кластеру родительского каталога; если

родительский каталог – корневой, начальным кластером считается нулевой.

Вопрос 31.Создание процесса в Unix. Фазы, наследование атрибутов, различия между подственными процессами.

Наследуется: id пользовавтеля, группы, диспозиция сигналов и их обработчики, ограничения накладываемые на процесс, текущий и корневой каталог, маска создания файлов, все файловые дескрипторы, включая файлы-указатели, …

После возврата из fork() процессы ввыполняют один и тот же код.

Отличия:id, pid, потомок свободен от сигналов, возвращаемое значение вызовом fork().

Процесс потомок получает копию u-area, различное время статич выполнения процесса в режиме ядра и в режиме задачи, блокировки памяти записей, уст родителем потомком не наследуются

Вопрос 32. Создание потоков/процессов в Win/QNX. Фазы наследования атрибутов при порождении, флаги.

Создание Win32 процесса осуществляется вызовом одной из таких функций, как CreateProcess, CreateProcessAsUser (для Win NT/2000) и CreateProcessWithLogonW (начиная с Win2000) и происходит в несколько этапов:

- Открывается файл образа (EXE), который будет выполняться в процессе. Если исполняемый файл не является Win32 приложением, то ищется образ поддержки (support image) для запуска этой программы. Например, если исполняется файл с расширением .bat, запускается cmd.exe и т.п.

- Создается объект Win32 "процесс".

- Создается первичный поток (стек, контекст и объект "поток").

- Подсистема Win32 уведомляется о создании нового процесса и потока.

- Начинается выполнение первичного потока.

- В контексте нового процесса и потока инициализируется адресное пространство (например, загружаются требуемые DLL) и начинается выполнение программы.

Вопрос 33. Алгоритм системного вызова fork(). Пример порождения.

1)резервируется место свопинга для сегм стека, данных
2)размещается новая запись proc в таблице процессов, присваивается PID
3)иниц стр proc, размещ карты отображения для.....
4)размещ u-area проц, копируется с родителя
5)созд соотв область памяти, частично совпадающая с родительской
6)устанавл в 0 значение возвращ для потомка
7)родителя возвращается PID
8)процесс помечается как готовый к записи и помещается в очередь готовых процессов

Пример:

```
if(fork()==0) {
        execl(“son.out,”son”,NULL);
}
```

Вопрос 34.Взаимосвязь функции exec() и алгоритма загрузки исполняемого файла.
execve() не возвращает управление при успешном выполнении, а код, данные, bss и стек вызвавшего процесса перезаписываются кодом, данными и стеком загруженной программы. Новая программа также наследует от вызвавшего процесса его идентификатор и открытые файловые дескрипторы, на которых не было флага закрыть-при-exec (close-on-exec, COE). Сигналы, ожидающие обработки, удаляются. Переопределённые обработчики сигналов возвращаются в значение по умолчанию. Обработчик сигнала SIGCHLD (когда установлен в SIG_IGN) может быть сброшен или не сброшен в SIG_DFL.

Вопрос 35.Образ процесса в иртуальной памяти для Unix и Windows. Форматы виртуальной памяти процесса в режиме задач для различных ОС.

Вопрос 36. Системные функции, идентификация различных IPC, пространства имен.

Вопрос 37. Процессы в NT. Состав процесса, адресное пространство, виртуальный образ, ресурсы. Атрибуты, сервисы и т. д.

В разных ОС процессы реализуются по-разному. Эти различия заключаются в том, какими структурами данных представлены процессы, как они именуются, какими способами защищены друг от друга и какие отношения существуют между ними. Процессы Windows NT имеют следующие характерные свойства:

Процессы Windows NT реализованы в форме объектов, и доступ к ним осуществляется посредством службы объектов.

Процесс Windows NT имеет многонитевую организацию.

Как объекты-процессы, так и объекты-нити имеют встроенные средства синхронизации.

Менеджер процессов Windows NT не поддерживает между процессами отношений типа "родитель-потомок".

В любой системе понятие "процесс" включает следующее:

исполняемый код, собственное адресное пространство, которое представляет собой совокупность виртуальных адресов, которые может использовать процесс, ресурсы системы, такие как файлы, семафоры и т.п., которые назначены процессу операционной системой, хотя бы одну выполняемую нить.

Адресное пространство каждого процесса защищено от вмешательства в него любого другого процесса. Это обеспечивается механизмами виртуальной памяти. Операционная система, конечно, тоже защищена от прикладных процессов. Чтобы выполнить какую-либо процедуру ОС или прочитать что-либо из ее области памяти, нить должна выполняться в режиме ядра.

Пользовательские процессы получают доступ к функциям ядра посредством системных вызовов. В пользовательском режиме выполняются не только прикладные программы, но и защищенные подсистемы Windows NT.

В Windows NT процесс - это просто объект, создаваемый и уничтожаемый менеджером объектов. Объект-процесс, как и другие объекты, содержит заголовок, который создает и инициализирует менеджер объектов. Менеджер процессов определяет атрибуты, хранимые в теле объекта-процесса, а также обеспечивает системный сервис, который восстанавливает и изменяет эти атрибуты.

В число атрибутов тела объекта-процесса входят:

Идентификатор процесса - уникальное значение, которое идентифицирует процесс в рамках операционной системы.

Токен доступа - исполняемый объект, содержащий информацию о безопасности.

Базовый приоритет - основа для исполнительного приоритета нитей процесса.

Процессорная совместимость - набор процессоров, на которых могут выполняться нити процесса.

Предельные значения квот - максимальное количество страничной и нестраничной системной памяти, дискового пространства, предназначенного для выгрузки страниц, процессорного времени - которые могут быть использованы процессами пользователя.

Время исполнения - общее количество времени, в течение которого выполняются все нити процесса.

Напомним, что нить является выполняемой единицей, которая располагается в адресном пространстве процесса и использует ресурсы, выделенные процессу. Подобно процессу нить в Windows NT реализована в форме объекта и управляется менеджером объектов.

Объект-нить имеет следующие атрибуты тела:

Идентификатор клиента - уникальное значение, которое идентифицирует нить при ее обращении к серверу.

Контекст нити - информация, которая необходима ОС для того, чтобы продолжить выполнение прерванной нити. Контекст нити содержит текущее состояние регистров, стеков и индивидуальной области памяти, которая используется подсистемами и библиотеками.

Динамический приоритет - значение приоритета нити в данный момент.

Базовый приоритет - нижний предел динамического приоритета нити.

Процессорная совместимость нитей - перечень типов процессоров, на которых может выполняться нить.

Время выполнения нити - суммарное время выполнения нити в пользовательском режиме и в режиме ядра, накопленное за период существования нити.
Состояние предупреждения - флаг, который показывает, что нить должна выполнять вызов асинхронной процедуры.
Счетчик приостановок - текущее количество приостановок выполнения нити.
Кроме перечисленных, имеются и некоторые другие атрибуты.
Как видно из перечня, многие атрибуты объекта-нити аналогичны атрибутам объекта-процесса.
Весьма сходны и сервисные функции, которые могут быть выполнены над объектами-процессами и объектами-нитями: создание, открытие, завершение, приостановка, запрос и установка информации, запрос и установка контекста и другие функции.

Вопрос 38.Многопоточность, многонитевость в NT. Состав, атрибуты и тд

Процессы Windows NT имеют следующие характерные свойства:

Процессы Windows NT реализованы в форме объектов, и доступ к ним осуществляется посредством службы объектов.

Процесс Windows NT имеет многонитевую организацию.

Как объекты-процессы, так и объекты-нити имеют встроенные средства синхронизации.

Менеджер процессов Windows NT не поддерживает между процессами отношений типа "родитель-потомок".

В Windows NT процесс - это просто объект, создаваемый и уничтожаемый менеджером объектов. Объект-процесс, как и другие объекты, содержит заголовок, который создает и инициализирует менеджер объектов. Менеджер процессов определяет атрибуты, хранимые в теле объекта-процесса, а также обеспечивает системный сервис, который восстанавливает и изменяет эти атрибуты.

В число атрибутов тела объекта-процесса входят:

Идентификатор процесса - уникальное значение, которое идентифицирует процесс в рамках операционной системы.

Токен доступа - исполняемый объект, содержащий информацию о безопасности.

Базовый приоритет - основа для исполнительного приоритета нитей процесса.

Процессорная совместимость - набор процессоров, на которых могут выполняться нити процесса.

Предельные значения квот - максимальное количество страничной и нестраничной системной памяти, дискового пространства, предназначенного для выгрузки страниц, процессорного времени - которые могут быть использованы процессами пользователя.

Время исполнения - общее количество времени, в течение которого выполняются все нити процесса.

Объект-нить имеет следующие атрибуты тела:

Идентификатор клиента - уникальное значение, которое идентифицирует нить при ее обращении к серверу.

Контекст нити - информация, которая необходима ОС для того, чтобы продолжить выполнение прерванной нити. Контекст нити содержит текущее состояние регистров, стеков и индивидуальной области памяти, которая используется подсистемами и библиотеками.

Динамический приоритет - значение приоритета нити в данный момент.

Базовый приоритет - нижний предел динамического приоритета нити.

Процессорная совместимость нитей - перечень типов процессоров, на которых может выполняться нить.

Время выполнения нити - суммарное время выполнения нити в пользовательском режиме и в режиме ядра, накопленное за период существования нити.

Состояние предупреждения - флаг, который показывает, что нить должна выполнять вызов асинхронной процедуры.

Счетчик приостановок - текущее количество приостановок выполнения нити.

Вопрос 39.Эмуляция сред на примере NT. Пример создания процессаиз прикладной программы, написанной для инородной среды.

Состояния:Готовность, Первоочередная готовность, Выполнение, Ожидание, Переходное состояние, Завершение.

Вопрос 40. Базовая структура процесса в многосредовых ОС.

Процесс (или по-другому, задача) - абстракция, описывающая выполняющуюся программу. На протяжении существования процесса его выполнение может быть многократно прервано и продолжено. Для того, чтобы возобновить выполнение процесса, необходимо восстановить состояние его операционной среды. Состояние операционной среды отображается состоянием регистров и программного счетчика, режимом работы процессора, указателями на открытые файлы, информацией о незавершенных операциях ввода-вывода, кодами ошибок выполняемых данным процессом системных вызовов и т.д. Эта информация называется контекстом процесса.

Кроме этого, операционной системе для реализации планирования процессов требуется дополнительная информация: идентификатор процесса, состояние процесса, данные о степени привилегированности процесса, место нахождения кодового сегмента и другая информация. В некоторых ОС (например, в ОС UNIX) информацию такого рода, используемую ОС для планирования процессов, называют дескриптором процесса.

Дескриптор процесса по сравнению с контекстом содержит более оперативную информацию, которая должна быть легко доступна подсистеме планирования процессов. Контекст процесса содержит менее актуальную информацию и используется операционной системой только после того, как принято решение о возобновлении прерванного процесса.

Очереди процессов представляют собой дескрипторы отдельных процессов, объединенные в списки. Таким образом, каждый дескриптор, кроме всего прочего, содержит по крайней мере один указатель на другой дескриптор, соседствующий с ним в очереди. Такая организация очередей позволяет легко их переупорядочивать, включать и исключать процессы, переводить процессы из одного состояния в другое.

Программный код только тогда начнет выполняться, когда для него операционной системой будет создан процесс. Создать процесс - это значит:
создать информационные структуры, описывающие данный процесс, то есть его дескриптор и контекст;
включить дескриптор нового процесса в очередь готовых процессов;
загрузить кодовый сегмент процесса в оперативную память или в область свопинга.

В многозадачной (многопроцессной) системе процесс может находиться в одном из трех основных состояний:
ВЫПОЛНЕНИЕ - активное состояние процесса, во время которого процесс обладает всеми необходимыми ресурсами и непосредственно выполняется процессором;
ОЖИДАНИЕ - пассивное состояние процесса, процесс заблокирован, он не может выполняться по своим внутренним причинам, он ждет осуществления некоторого события, например, завершения операции ввода-вывода, получения сообщения от другого процесса, освобождения какого-либо необходимого ему ресурса;
ГОТОВНОСТЬ - также пассивное состояние процесса, но в этом случае процесс заблокирован в связи с внешними по отношению к нему обстоятельствами: процесс имеет все требуемые для него ресурсы, он готов выполняться, однако процессор занят выполнением другого процесса.

Вопрос 41. Реализация сервисов по запросу приложений в ОС с различными архитектурами. 2 подхода к реализации клиент-серверной модели.

совместное использование сегментов библиотек dll

однократное или многократное тиражирование. Каждое из приложений может модифицировать данные, используемые всеми остальными. Обращение к dll из прикладной программы осуществляется через обычный вызов процедуры.

Два подхода к реализации клиент-серверной модели:

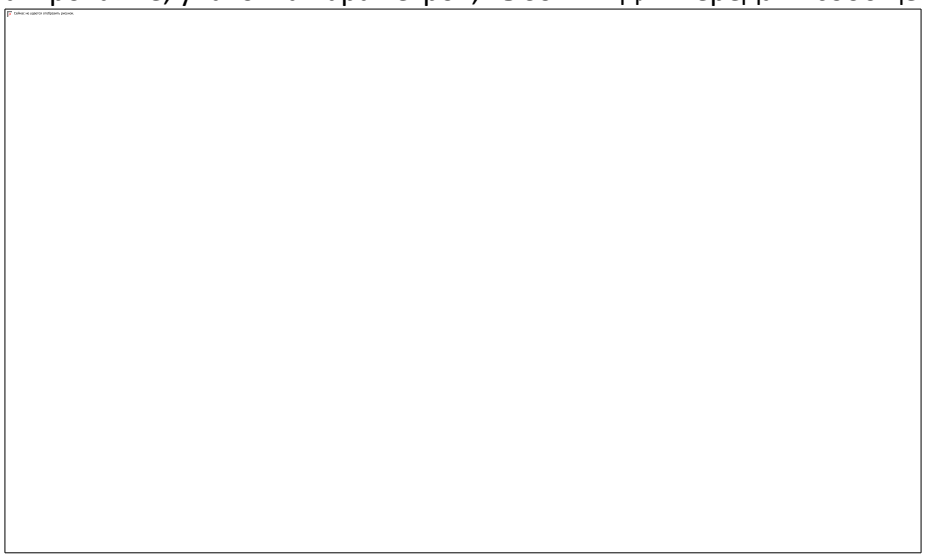
1)модель с защитой памяти и подсистем

Взаимодействие между приложением и подсистемой реализуется за счет реализуется за счет передачи сообщений

2)модель API с dll защитой

Связь приложения с API происходит через точки входа(stub,звглушки)

В заглушке: форматирование, упаковка параметров, вызов NT для передачи сообщения серверу.



Вопрос 42. Преимущества и недостатки клиент-серверной модели. Способы повышения производительности

Преимущества:
1)Защита глобальных структур данных серверов и других приложений
2)автономность серверов
3)сервер и приложения пользователя не могут менять данные ядра
4)единообразный доступ к ядру
5)разделение между ОС и клиентом
6)любое количество серверов поддерживает множество сред
Недостатки:
1)падение производительности

Способы повышения производительности:
1)пакетизация
2)сокращения количества обращений к серверу за счет реализации на стороне клиента функций API, которые не используют и не модифицируют глобальные данные и в результате все API функции выполняются в dll на стороне клиента.
3)кеширование данных

Вопрос 43.Физическая организация устройств ввода/вывода. Организация ПО ввода/вывода.
Физическая организация устройств ввода-вывода

Устройства ввода-вывода делятся на два типа: блок-ориентированные устройства и байт-ориентированные устройства. Блок-ориентированные устройства хранят информацию в блоках фиксированного размера, каждый из которых имеет свой собственный адрес. Самое распространенное блок-ориентированное устройство - диск. Байт-ориентированные устройства не адресуемы и не позволяют производить операцию поиска, они генерируют или потребляют последовательность байтов. Примерами являются терминалы, строчные принтеры, сетевые адаптеры.

Внешнее устройство обычно состоит из механического и электронного компонента. Электронный компонент называется контроллером устройства или адаптером. Механический компонент представляет собственно устройство. Некоторые контроллеры могут управлять несколькими устройствами.

Операционная система обычно имеет дело не с устройством, а с контроллером. Контроллер, как правило, выполняет простые функции, например, преобразует поток бит в блоки, состоящие из байт, и осуществляют контроль и исправление ошибок. Каждый контроллер имеет несколько регистров, которые используются для взаимодействия с центральным процессором. В некоторых компьютерах эти регистры являются частью физического адресного пространства. В таких компьютерах нет специальных операций ввода-вывода. В других компьютерах адреса регистров ввода-вывода, называемых часто портами, образуют собственное адресное пространство за счет введения специальных операций ввода-вывода (например, команд IN и OUT в процессорах i86).

Организация программного обеспечения ввода-вывода

Основная идея организации программного обеспечения ввода-вывода состоит в разбиении его на несколько уровней, причем нижние уровни обеспечивают экранирование особенностей аппаратуры от верхних, а те, в свою очередь, обеспечивают удобный интерфейс для пользователей.

Ключевым принципом является независимость от устройств. Вид программы не должен зависеть от того, читает ли она данные с гибкого диска или с жесткого диска.

Очень близкой к идее независимости от устройств является идея единообразного именования, то есть для именования устройств должны быть приняты единые правила.

Другим важным вопросом для программного обеспечения ввода-вывода является обработка ошибок. Вообще говоря, ошибки следует обрабатывать как можно ближе к аппаратуре. Если контроллер обнаруживает ошибку чтения, то он должен попытаться ее скорректировать. Если же это ему не удается, то исправлением ошибок должен заняться драйвер устройства. Многие ошибки могут исчезать при повторных попытках выполнения операций ввода-вывода, например, ошибки, вызванные наличием пылинок на головках чтения или на диске. И только если нижний уровень не может справиться с ошибкой, он сообщает об ошибке верхнему уровню.

Еще один ключевой вопрос - это использование блокирующих (синхронных) и неблокирующих (асинхронных) передач. Большинство операций физического ввода-вывода выполняется асинхронно - процессор начинает передачу и переходит на другую работу, пока не наступает прерывание. Пользовательские программы намного легче писать, если операции ввода-вывода блокирующие - после команды READ программа автоматически приостанавливается до тех пор, пока данные не попадут в буфер программы. ОС выполняет операции ввода-вывода асинхронно, но представляет их

для пользовательских программ в синхронной форме.

Последняя проблема состоит в том, что одни устройства являются разделяемыми, а другие - выделенными. Диски - это разделяемые устройства, так как одновременный доступ нескольких пользователей к диску не представляет собой проблему. Принтеры - это выделенные устройства, потому что нельзя смешивать строчки, печатаемые различными пользователями. Наличие выделенных устройств создает для операционной системы некоторые проблемы.

Для решения поставленных проблем целесообразно разделить программное обеспечение ввода-вывода на четыре слоя :
Обработка прерываний,
Драйверы устройств,
Независимый от устройств слой операционной системы,
Пользовательский слой программного обеспечения.

Вопрос 44.Система ввода/вывода в NT. Компоненты, процедуры диспетчера функций драйвера

Часть исполнительной системы, получающая запросы от процессов пользователя либо режима пользователя, либо ядра, и передающая эти запросы устройствам ввода-вывода в преобразованном виде.

Система управляется пакетами. Каждый запрос представляет IRP-пакет (Input Request Package).
IRP позволяет использовать иерарх. настр. системы В/В и драйверов.
IRP – структура данных, управляющая обработкой выполнения операций на каждой стадии их выполнения.
Диспетчер В/В – входит в состав системы В/В, управляет передачей запросов В/В в файловую систему, реализует процедуру общего назначения для разных драйверов.

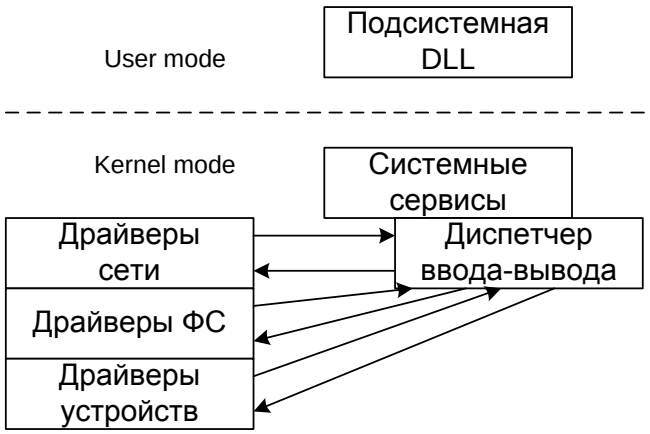
Функции и этапы работы диспетчера:

- Создание IRP определенного формата
- Передача пакета соответствующему драйверу
- Удаление IRP

Функции драйвера:

- Получение IRP
- Выполнение операции
- Возвращение управления В/В, либо реализация завершения

В NT существуют драйверы файловой системы и драйверы устройств В/В. При любой операции В/В задействованы и драйверы ФС и драйверы устройств. Работают через IRP, контролируются диспетчером. Реализуется общая процедура для различных драйверов. Упрощаются отдельные драйвера и вносится универсальность.

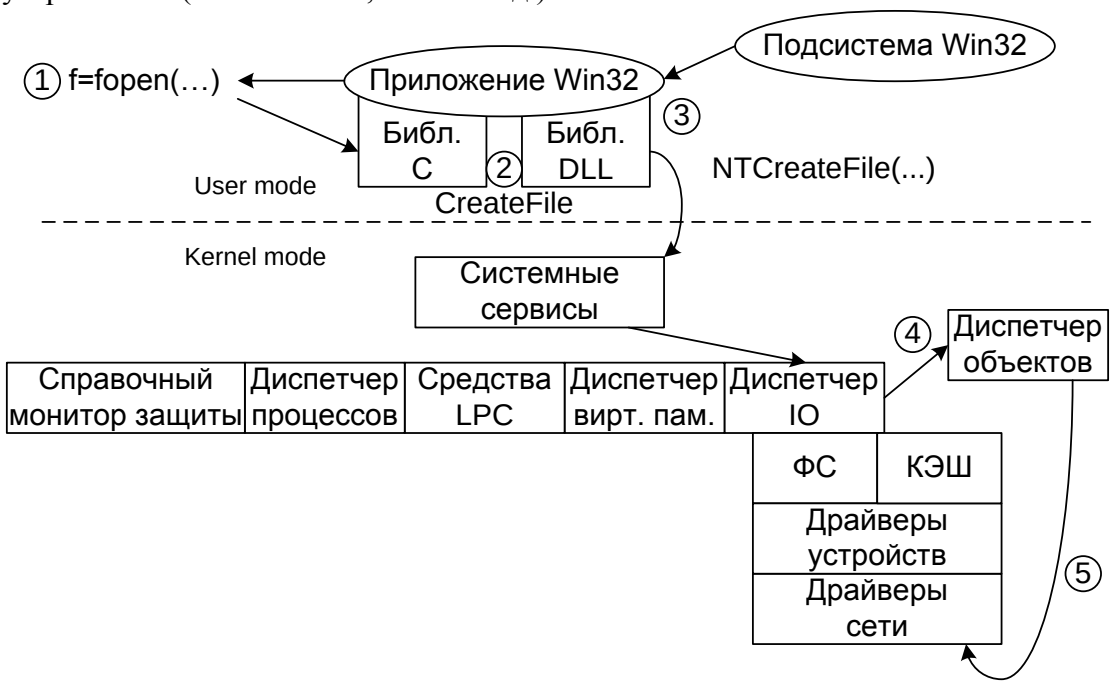


Принцип: виртуальные файлы устройств (как в UNIX) – объектная модель. Работа осуществляется посредством файловых описателей. Описатель ссылается на объект – файл.
Потоки пользовательского режима вызывают базовые сервисы файла объекта (чтение, запись, открытие). Диспетчер динамически преобразует эти запросы в запросы к реальным физическим устройствам (накопителям, сетям и т.д.)

Вопрос 45.Объектная модель системы ввода-вывода в NT.Пример открытия файлового объекта.

Принцип: виртуальные файлы устройств (как в UNIX) – объектная модель. Работа осуществляется посредством файловых описателей. Описатель ссылается на объект – файл.

Потоки пользовательского режима вызывают базовые сервисы файла объекта (чтение, запись, открытие). Диспетчер динамически преобразует эти запросы в запросы к реальным физическим устройствам (накопителям, сетям и т.д.)



Вопрос 46. Унифицированная модель драйвера в NT. Многослойный драйвер.

Требования, которые предъявляются для разработчиков драйверов под NT:

- Драйверы пишутся на языках высокого уровня для обеспечения переносимости.
- Операция В/В управляется IRP-пакетами. IRP используются многократно по слоям. Система динамически назначает драйверы для управления дополнительными новыми.
- Драйверы должны синхронизировать доступ к глобальным данным
- Должны позволять вытеснять потоками более высокого приоритета
- Должны быть прерываемы другими потоками
- Код драйвера должен быть способен выполняться на нескольких процессорах.
- Драйверы должны восстанавливаться после сбоя и выполнять нереализованные операции (в NT 4.0 не реализовано)

Драйверы имеют унифицированную модель интерфейса => диспетчер В/В не видит их структуру и детали реализации. При взаимодействии различных драйверов диспетчер играет роль посредника.

Драйверы могут быть:

- Однослойные (для последовательного порта и др.)
- Многослойные (для ЗУ большой емкости)

Пример многослойного драйвера – при работе через SCSI-интерфейс драйвер дисков передает запрос на драйвер SCSI, который реализует обработку. Запрос к дискам реализует драйвер SCSI.

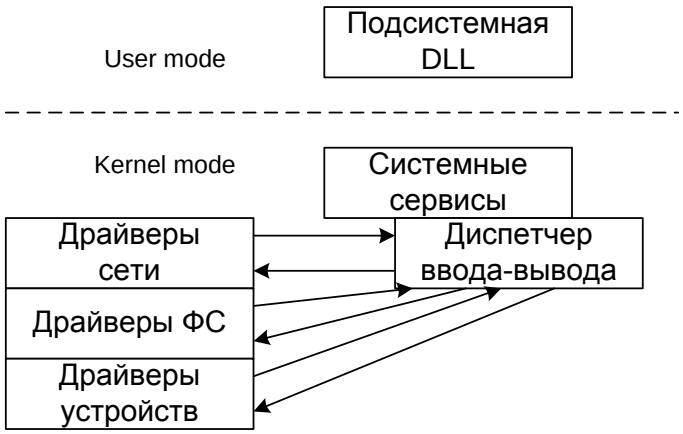
Вопрос 47.Синхронный и асинхронный ввод/вывод в NT, формат IRP, проекционный ввод/вывод.
Часть исполнительной системы, получающая запросы от процессов пользователя либо режима пользователя, либо ядра, и передающая эти запросы устройствам ввода-вывода в преобразованном виде.

Система управляется пакетами. Каждый запрос представляет IRP-пакет (Input Request Package).
IRP позволяет использовать иерарх. настр. системы В/В и драйверов.
IRP – структура данных, управляющая обработкой выполнения операций на каждой стадии их выполнения.
Диспетчер В/В – входит в состав системы В/В, управляет передачей запросов В/В в файловую систему, реализует процедуру общего назначения для разных драйверов.

Функции и этапы работы диспетчера:
- Создание IRP определенного формата
- Передача пакета соответствующему драйверу
- Удаление IRP

Функции драйвера:
- Получение IRP
- Выполнение операции
- Возвращение управления В/В, либо реализация завершения

В NT существуют драйверы файловой системы и драйверы устройств В/В. При любой операции В/В задействованы и драйверы ФС и драйверы устройств. Работают через IRP, контролируются диспетчером. Реализуется общая процедура для различных драйверов. Упрощаются отдельные драйвера и вносится универсальность.



Вопрос 48.Структура драйвера, минимальные наборы процедур различных драйверов: простейший, с прерываниями, многослойный

Всякий драйвер – независимый элемент ОС, который может быть динамически загружен или удален без изменения конфигурации ОС.

Драйверы имеют унифицированную модель интерфейса => диспетчер В/В не видит их структуру и детали реализации. При взаимодействии различных драйверов диспетчер играет роль посредника.

Драйверы могут быть:

- Однослойные (для последовательного порта и др.)
- Многослойные (для ЗУ большой емкости)
- с прерываниями

Передача может быть:

- Асинхронной – более быстродействующий вариант и экономичный в плане ресурсов
- Синхронный – проще в плане реализации

Структура драйвера:

Составная часть	**Назначение процедуры.**
Процедура инициализации (1)	Выполняется диспетчером В/В при загрузке ОС. Создаются системные объекты для распознавания устройства и доступа к нему.
Набор процедур распределения (2)	Главные функции, предоставляемые драйверами устройствам. Запрос В/В заставляет диспетчер В/В генерировать IRP и обращается к драйверу через процедуру распределения.
Процедура запуска В/В (3)	Драйвер использует эту процедуру для начала передачи данных. Необязательная процедура характерна для физических устройств при передаче больших объемов.
ISR Interrupt (4)	Прерывание от устройства. На момент выполнения приоритет RPL повышается. Сообразно ISR выполняет часть кода обработчиков. В соответствии с прерыванием эта часть кода может быть с повышенным приоритетом, а часть кода обработки является вторичной и выполняется посредством отлож. вызова DPC (ISR генерир. DPC).
Процедура DPC (5)	Процедура обработки прерывания, которая выполняется при более низком, чем у IRQ приоритете. Выполняет большую часть обработки по обслуживанию прерывания. Это инициирует завершение В/В и запуск следующего запроса из очереди на обработку устройствами В/В.
Процедура завершения (6)	Является посредником между драйверами нижнего и верхнего уровня. В ней хранится информация о возможных значениях, сбое или необходимости очистки при инициализации.
Процедура отмены В/В (7)	Может быть не одной. Информация об активизации процедуры отмены хранится в IRP.
Процедура выгрузки (8)	Освобождение системных ресурсов, которые используются драйвером. Затем диспетчер В/В удаляет драйвер из памяти. Драйвер может быть загружен/выгружен во время работы.
Процедура протоколирования ошибок (9)	Информирование диспетчера В/В об ошибке, который помещает эту информацию в журнал ошибок.

Минимальный драйвер должен содержать 3 процедуры:

Драйвер устройства:
(1), (2), (3), (4), (5), (6), (8) для многослойного драйвера

Вопрос 49.Семафоры, мьютексы, алгоритмы разделения памяти с минимальным количеством средств синхронизации.

Семафоры.

Семафоры – средство синхронизации доступа нескольких процессов к разделяемым ресурсам (функция разрешения/запрещения использования ресурса).

Требования к семафорам:

- должны быть размещены в адресном пространстве ядра;
- операции над семафорами должны быть неделимыми (атомарными), что возможно только при выполнении операций в режиме ядра.

Семафоры – системный ресурс. Операции над семафорами производятся посредством системных вызовов. В UNIX системах под семафором понимается группа семафоров.

Структура для описания семафоров:

sem_id – идентификатор семафора;
sem_perm – права доступа;
sem_first – указатель на первый элемент;
sem_num – число семафоров в группе;
sem_oper - последняя операция;
sem_time – время последнего изменения;
И др.

Также в структуре sem хранится имя семафора, значение семафора, идентификаторы процессов, выполнивших последние операции изменения семафора, число процессов, ожидающих увеличения семафора, число процессов, ожидающих обнуления семафора.

Мьютексы.

Мью́текс (англ. mutex, от mutual exclusion — взаимное исключение) — одноместный семафор, служащий в программировании для синхронизации одновременно выполняющихся потоков.

Мьютексы (mutex) — это один из вариантов семафорных механизмов для организации взаимного исключения. Они реализованы во многих ОС, их основное назначение — организация взаимного исключения для потоков из одного и того же или из разных процессов.

Мьютексы — это простейшие двоичные семафоры, которые могут находиться в одном из двух состояний — отмеченном или неотмеченном (открыт и закрыт соответственно). Когда какой-либо поток, принадлежащий любому процессу, становится владельцем объекта mutex, последний переводится в неотмеченное состояние. Если задача освобождает мьютекс, его состояние становится отмеченным.

Задача мьютекса — защита объекта от доступа к нему других потоков, отличных от того, который завладел мьютексом. В каждый конкретный момент только один поток может владеть объектом, защищённым мьютексом. Если другому потоку будет нужен доступ к переменной, защищённой мьютексом, то этот поток засыпает до тех пор, пока мьютекс не будет освобождён.

Цель использования мьютексов — защита данных от повреждения; однако им порождаются другие проблемы, такие как взаимная блокировка (клинч) и «соревнование за ресурс».

Вопрос 50.IPC UNIX, применение, мотивация использования, системные вызовы, функции, примеры кода.

Основная идея в Unix – каждый процесс изолирован от другого => защита процессов и ОС, взаимодействия осуществляется с помощью механизмов , реализованных на уровне ядра.
Задачи IPC: передача данных между процессами, совместное использование данных(без копирования), синхронизация(извещение).
Средства IPC: сигналы, каналы, именованные каналы, сообщения, разделяемая память, семафоры, сокеты.

Сигналы.

Сигналы – способ передачи уведомления о событии, произошедшем либо между процессами, либо между процессом и ядром. Сигналы очень ресурсоемки. Ограничены с точки зрения системных средств. Они малоинформативны. Являются простейшим способом IPC. Используются для генерации простейших команд, уведомлений об ошибке. Обработка сигнала похожа на обработку прерывания. Сигнал имеет собственное имя и уникальный номер. Максимальное количество сигналов – 32. SIGINT - генерируется при нажатии клавиши del, Ctrl+C и т.д. Сигнал может быть отправлен процессу ядром или другим процессом. Это можно сделать посредством системного вызова.

Причины генерации сигналов:

1) Ядро отправляет сигнал процессу при нажатии клавиши;
2) Деление на 0;
3) Обращение к недоступной области памяти;
4) Особые аппаратные ситуации;
5) Особые программные ситуации.

Основные действии я при приеме сигналов:

- игнорирование;
- действия на данный сигнал, согласно умолчанию;
- обработка собственным обработчиком.

Две фазы существования сигнала:

1) генерация и отправка;
2)доставка и обработка.

Каналы.

Обмен данными между процессами порождает программный канал, обеспечивающий симплексную передачу между двумя процессами (задачами).

Системный вызов: int pipe(int *filedes);

filedes[0] – указатель на запись;
filedes[1] – указатель на чтение;

По сути системный вызов вызывает 2 файловых дескриптора, которые позволяют открыть файл на запись и файл на чтение. Эти каналы могут использоваться только родственными процессами.

В отличие от неименованных каналов (pipe) – возможен обмен данными не только между родственными процессами, так как буферизация происходит в рамках файловой системы с именованием.

Системный вызов: mknode(char *pathname, int mode, int dev);

Правила открытия канала на запись и чтение для независимых процессов:

1) при чтении n байт из N (n<N) системный вызов возвращает n байт, а остальное сохраняет до следующего чтения;

2) при чтении n байт из N (n<N) (n>N) возвращается N и происходит обработка прерывания;

3) если канал пуст, то возвращается 0 байт, если канал открыт на запись, то процесс будет заблокирован до того момента, пока не появится хотя бы 1 байт;

4) если в канал пишут несколько процессов, то информация не смешивается (это достигается путем использования tag’ов);

5) при записи, когда FIFO полон, процесс блокируется до тех пор, пока не появится место для записи.

При использовании FIFO не гарантируется атомарность операций.

При записи FIFO работает как клиент-серверное приложение.

Функции сервера:

- создание FIFO (mknode());
- открытие FIFO на чтение;
- чтение сообщений;
- закрытие FIFO.

Функции клиента:

- открытие FIFO на запись;
- запись сообщений для сервера в FIFO;
- удаление FIFO.

Сообщения.

Сообщения обслуживаются ОС, являются разделяемым системным ресурсом, являются частью ядра ОС, образуют очереди, которые образуют список. Сама очередь имеет уникальное имя – идентификатор.

Процессы могут читать сообщения из различных очередей.

Атрибуты сообщений:

- тип сообщения (мультиплексный/немультиплексный);
- длина сообщений (байт);
- сами данные (м.б. структурированы).

Хранится очередь в виде однонаправленного односвязного списка. Для каждой очереди ядро создает свою структуру.

Разделяемая память.

Разделяемая память может использоваться, если есть определенные объекты, управляющие семафорами. Применение разделяемой памяти, в соответствии со специальным механизмом позволяет повысить быстродействие.

При этом для сервера необходимо:

- получение доступа к разделяемой памяти посредством семафора;
- запись данных в область разделяемой памяти;
- освобождение памяти в соответствие со значением семафора.

Для клиента:

- получение доступа к разделяемой памяти посредством семафора;
- чтение данных из области разделяемой памяти;
- освобождение памяти в соответствие со значением семафора.

Для организации клиент-серверного приложения с доступом к разделяемой памяти необходимо 2 семафора.

Вопрос52.Организация сообщений в различных ОС. Локальное и сетевое применение сообщений

Сообщения.

Сообщения обслуживаются ОС, являются разделяемым системным ресурсом, являются частью ядра ОС, образуют очереди, которые образуют список. Сама очередь имеет уникальное имя – идентификатор.

Процессы могут читать сообщения из различных очередей.

Атрибуты сообщений:

- тип сообщения (мультиплексный/немультиплексный);
- длина сообщений (байт);
- сами данные (м.б. структурированы).

Хранится очередь в виде однонаправленного односвязного списка. Для каждой очереди ядро создает свою структуру.

msgid_ds ----------□ ipc_perm
msg_perm (permission – права доступа)
msg_cbytes (число байт)
msg_num (число сообщений в очереди)
msg_first (первый объект в списке)
msg_last (последний в списке)

Функции:
msg_snd() – отправка сообщения;
msg_rcv() – прием сообщения;
msg_get()– прием сообщения;
msg_ctl()– прием сообщения.

LPC – механизм передачи. Приложение вызывает функции API-dll, после этого производится обращение по точке входа – stub, (осуществляется упаковка сообщений), дальше передается в защищенный локальный сервер LPC или удаленный сервер посредством RPC, тогда сообщение попадает перед сервером в его точку входа stub для преобразования его к виду, понятному для сервера (люблю хорошего русского языка, Botsman).

LPC предлагает 3 способа передачи сообщения:

1. Посылка сообщения в объект – порт, связанный с серверным процессом.
2. Посылка в порт сервера указателя на сообщение, и передача сообщения через общую разделяемую память.
3. Передача сообщения через особую область памяти в рамках разделяемой, причем определенному потоку.

Для любого из 3-х способов необходимо установить соединение (канал связи для передачи сообщений).

Локальное применение собщений — одно из средств IPC, позволяющее процессам обмениваться информацией.

Сетевое применение — передача данных между одтельными компьютерами. Главное отличие заключается в том, что при удалённой передачи, сообщения необходимо переводить в формат понятный приёмнику

Вопрос 53.Функциональные компоненты сетевых ОС, взаимодействие, способы реализации Сетевые и распределенные ОС

Сетевая операционная система составляет основу любой вычислительной сети. Каждый компьютер в сети в значительной степени автономен, поэтому под сетевой операционной системой в широком смысле понимается совокупность операционных систем отдельных компьютеров, взаимодействующих с целью обмена сообщениями и разделения ресурсов по единым правилам - протоколам. В узком смысле сетевая ОС - это операционная система отдельного компьютера, обеспечивающая ему возможность работать в сети.

В зависимости от того, какой виртуальный образ создает ОС для того, чтобы подменить им реальную аппаратуру, различаются сетевые и распределенные ОС.

Сетевая ОС не полностью скрывает распределенную природу сети, то есть является виртуальной сетью. Работая в сетевой ОС, пользователь хотя и может запустить задание на любой машине, всегда знает, на какой машине выполняется его задание. По умолчанию задание выполняется на той машине, на которой пользователь сделал логический вход. Часто под сетевой ОС понимается набор ОС отдельных компьютеров сети.

Магистральным направлением развития сетевых ОС является прозрачности сетевых ресурсов. В идеальном случае сетевая ОС должна предоставить пользователю сетевые ресурсы в виде ресурсов единой централизованной виртуальной машины. Для такой ОС используют специальное название - распределенная ОС. Пользователь распределенной ОС, вообще говоря, не имеет сведений о том, на какой машине выполняется его работа. В настоящее время практически все сетевые ОС еще очень далеки от идеала истинной распределенности.

Функциональные компоненты сетевой ОС

1) Средства управления локальными ресурсами - реализуют все функции автономного компьютера

a. Распределение памятью;

b. Планирование и диспетчеризацию процессов;

c. Управление внешней памятью;

d. Интерфейс с пользователем и пр.

2) Сетевые средства можно разделить на три компонента:

a. Серверная часть ОС - предоставляет локальные ресурсы в общее пользование;

b. Клиентская часть ОС - средства доступа к удаленным ресурсам и услугам;

c. Транспортные средства ОС, которые совместно с коммуникационной системой обеспечивает передачу сообщений в сети.

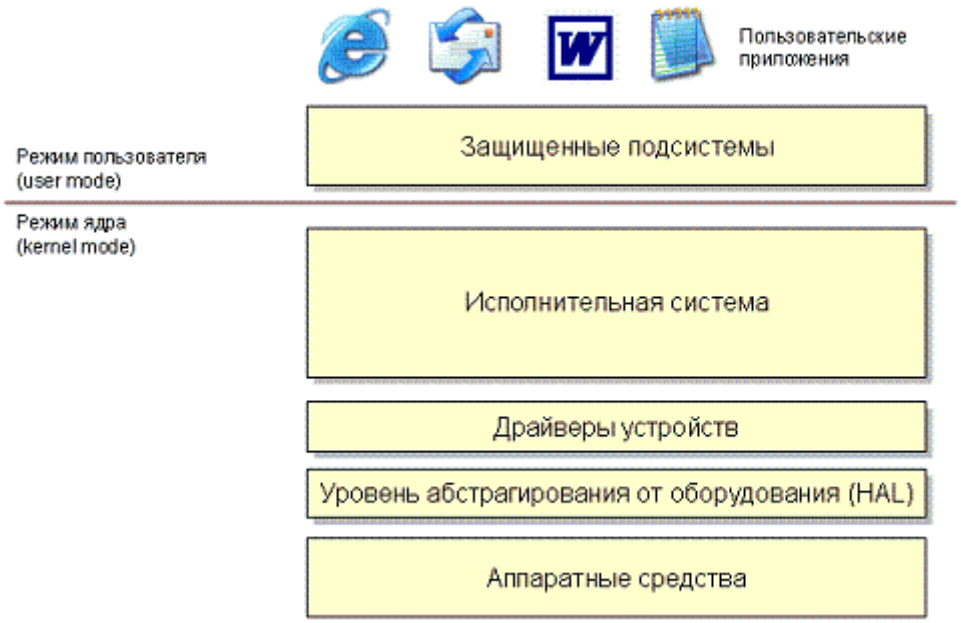
Вопрос 54.Структура ядра ОС с иерархической архитектурой. Алгоритмы выполнения запросов прикладного уровня

Структурно ее можно разделить на две части.
Первая часть работает в режиме ядра (kernel mode) и называется исполнительной системой .
Компоненты режима ядра обладают следующими функциональными возможностями:
1)имеют доступ к оборудованию;
2)имеют прямой доступ ко всем видам памяти компьютера;
3)не выгружаются на жесткий диск в файл подкачки;
4)имеют более высокий приоритет, нежели процессы режима пользователя.

Вторая часть работает в так называемом режиме пользователя (user mode) Эту часть составляют защищенные подсистемы ОС. Особенности процессов пользовательского режима:

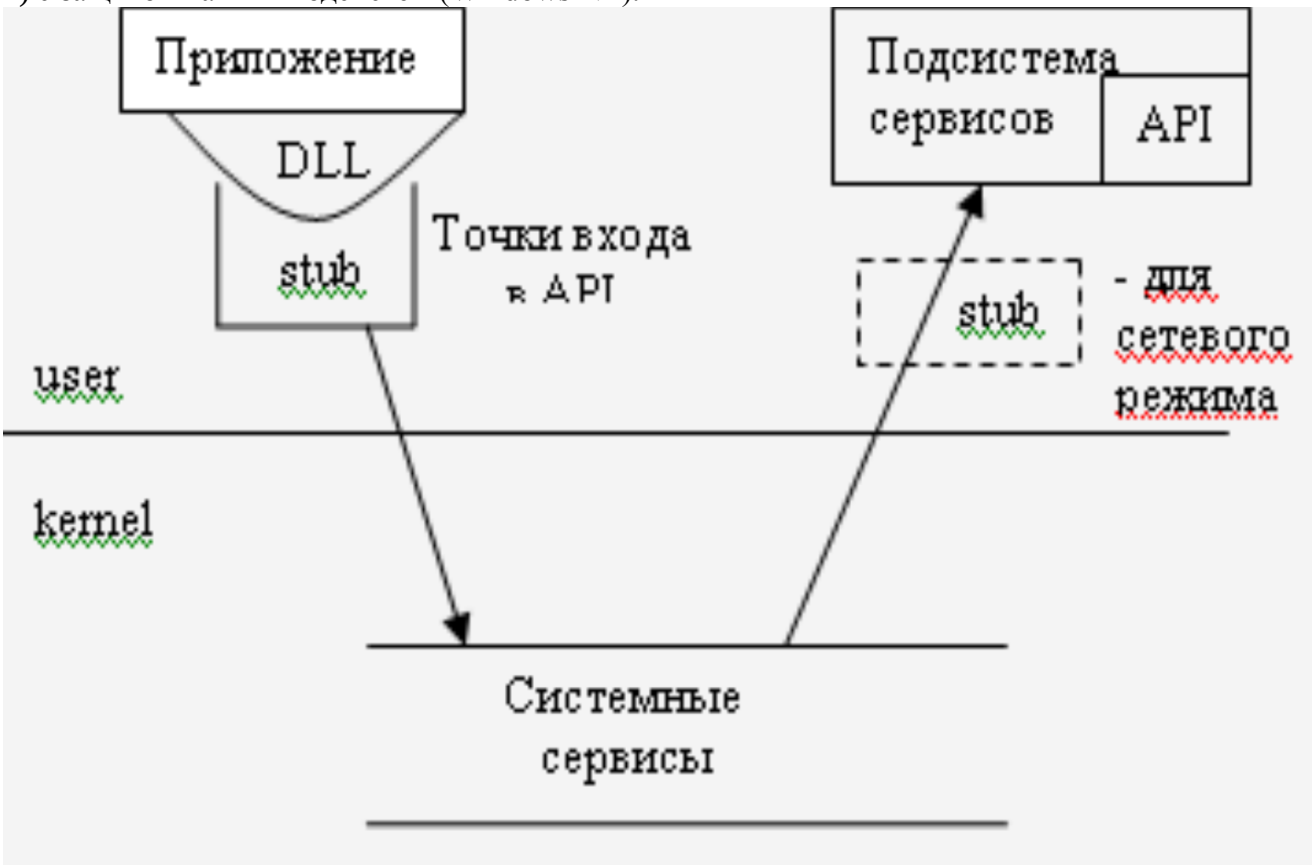
1)не имеют прямого доступа к оборудованию, все запросы на использование аппаратных ресурсов должны быть разрешены компонентом режима ядра;
2)ограничены размерами выделенного адресного пространства, это ограничение устанавливается выделением процессу фиксированных адресов;
3)могут быть выгружены из физической памяти в виртуальную на жестком диске;
4)приоритет процессов данного типа ниже приоритета процессов режима ядра, это предохраняет ОС от снижения производительности или задержек, происходящих по вине приложений.

Интерфейс сист вызовов
--Межслойный интерфейса
Функциональный слой
--Межслойный интерфейс
ФС
HAL



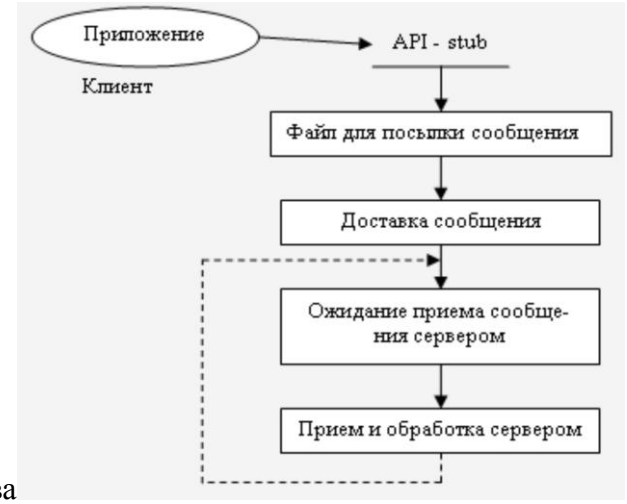
В монолитных и иерархических системах механизм реализации сервиса: запрос системного сервиса приводит к переходу в режим ядра, выполняется в режиме ядра, а затем выход обратно.
В случае клиент-серверной модели:

1) Совместное использование сегментов dll (Windows X, OS/2).
Различные приложения разделяют АП различных динамических библиотек, при этом в Windows делается однократное копирование библиотек в разделяемую память (доступ от всех), а в OS/2 – многократное копирование библиотек в каждый процесс отдельно (в АП процесса). Каждый процесс может модифицировать данные, используемые всеми приложениями. Обращение производится за счет обычных вызовов подпрограмм.
2) с защитой памяти подсистем (Windows NT).



stub – приводит запрос к формату для передачи в подсистему.
Модель защиты (клиент-серверная):
«+»
1. Защита глобальных структур данных
2. Автономность сервисов
3. Сервера для приложения работают в пользовательском режиме, следовательно, не могут влиять на ядро и не могут вызывать внутренние функции ОС. Единственный доступ к ядру – вызов системных сервисов.
4. Обеспечивает разделение сервера – ОС.
5. Возможность одновременной работы любого количества серверов поддерживающих средства API. Повышение степени параллелизма.

«-»
1. Потеря производительности.
Для получения и обработки сообщения сервером необходимо: 1)переключить контекст 2)сохранить предыдущий
3)выбрать для исполнения поток сервера
4)загрузить его контекст
5)выполнить функции API (используя поток сервера)
6)сохранить контекст потока-сервера
7)загрузить обратно поток клиентского потока

8)обработка результата API вызова 

Вопрос 55. Организация прерываний, значение механизма прерываний в организации ОС
Прерывания в ОС реализованы в виде таблицы векторов прерываний, индекс в которой является номером вектора прерывания. В векторе прерывания содержится адрес начала участка кода с обработчиком прерывания.
Также прерывания представлены аппаратно в виде контроллера прерываний, включенным между процессором и внешними источниками прерываний.
Запросы от внешних источников подключены к входам IRQ0-IRQ7 контроллера ; с выхода INT контроллера на вход INTR процессора подается сигнал запроса от наиболее приоритетного источника. При получении ответного сигнала INTA контроллер выдает номер прерывания на шину данных; номер образуется суммой IRQ номера с некоторым базовым значением.
При поступлении заявок от различных источников выполняется заявка с наименьшим номером IRQ. Имеется возможность выборочно заблокировать заявки от отдельных IRQ входов. Блокировку, или маскирование заявок, а также выбор заявкки с наибольшим приоритетом обеспечивают три байтовых регистра iMR, iRr,iSr.
Каждая из линий IRQx в контроллере подключена к разряду x регистра iRr через входной вентиль. Вентиль открыт, если разряд x в регистре маски сброшен. Регистр маски подключен к порту 021.
Виды прерываний: Внешние и внутренние. Внутренние не могут быть замаскированы. Внешние бывают Маскируемые и несакируемые. Внутренние : Сбой(Fault), Ловушка(Trap), Фатальная ошибка(Abort).

Значение механизма прерываний в ОС заключается в получении сигналов от внешних устройств о возникновении какого-либо события. Внутренние прерывания позволяют ОС получить информацию о произошедших ошибках в ходе выполнения программы(возникновение ошибки деления на 0, fault) и о невосстановимых системных сбоях (abort), также они позволяют воспользоваться внутренними системными ресурсами(trap исключения). Для реализации системных вызовов.

Вопрос 56. Обработка нажатия клавиш и других событий в системе.

При нажатии клавиши и отпускании клавиши подается сигнал запроса на вход IRQ1 ведущего контроллера прерываний. Номер прерывания от клавиатуры равен 9.

Регистры устройства подключены к микросхеме интерфейса с периферией, в частности код клавиши доступен через порт 060. Чтение принятых данных не влияет на состояние устройства. Причина прерываний при этом не сбрасывается, а данные при повторном чтении остаются теми же.

Для начала необходимо заменить стандартный обработчик, с сохранением его адреса. Записать в таблицу векторов прерываний по этому индексу адрес нового обработчика. После завершения работы программы восстановить старый обработчик.

Процедура обработки прерывания должна содержать:1) определения кода нажатой клавиши 2) выполнение определенного действия в зависимости от кода 3) сброс текущей заявки в контроллере прерываний 4)снятие причины прерывания — посылкой импульса подтверждения в бите 7 порта 061.

Пример кода:

```
        jmp start:
old_vect        dd      ?
n_vect                  equ     9
lea     ax,new
mov     dx,cs
mov     ds,0
xchg    w [n_vect*4], ax
xchng   w [n_vect*4], dx//замена вектора прерывания
mov     ds, cs

mov     w old_v, ax
mov     w old_v, dx

mov     ax,  w old_v
mov     dx, w old_v+2

mov es, 0
es mov w [n_vect *4], ax//воостановление вектора прерывания
es mov w [n_vect*4 +2], dx

mov     ax,04c00
int     021

new:
push ax,bx,dx
mov dx, 060
in ax,dx
//обработка в зависимости от кода клавиши

mov dx, 020h//снятие заявки в контроллере прерываний
mov ax,020h
out dx,ax

mov dx, 061h//снятие причины прерывания
mov ax,08h
out dx, ax
pop dx,bx,ax
```

iret

Вопрос 57 Резидентные программы

Резидентнтная программа — это программа, которая завершает выполнение, не освобождая занятую память полностью. Этот термин используется применительно к ОС DOS. Обычно в памяти оставляют фрагмент программы, содержащей процедурю обработки прерываний; в нерезидентной части программы выполняется настройка векторов на процедуры обработки из резидентной части программы.

После установки векторов выполнение программы как самостоятельного процесса заканчиваетяся — но таким образом, что код и данные остаются в памяти. Отсюда — принятое в англоязычной технической литературе название Terminate but stay Resident. Или, сокращенно TSR. Оставленая в памяти процкдура обработки прерывания в момент ее вызова считается частью *текущего* , т.е. прерванного процесса. В результате, обращение к функции DOS 04c из резидентной процедуры приводит к завершению *прерванной* программы.

Завершение программы с сохранением ее фрагмента в памяти выполняется вызовом int 027h , вместо обычного int 020h. В регистре dx при вызове int 027 должен быть задан адрес нижней границы области освобождаемой памяти. Фрагмент от смещения 0 до смещения, указанного в dx , сохраняется.

Код программы для обработки исключения 4:

```
jmp start:

msg db 13,10,'?-Trap (into)$'

new4:
        push ax,dx,ds
        mov ax,cs
        mov ds,ax
        lea dx,msg
        mov ah,9
        int 021
#if stop
        mov ax,04c01
        int 021
#else
        pop ds,dx,ax
        iret
#endif

start:
        mov    ax, 02504
        lea    dx, new4
        int    021
        mov    dx, start
        int 027
```

Вопрос 59 ISR,примеры передачи управления стандартному обработчику

Контроллер прерываний

Запросы от внешних источников подключены к входам IRQ0-IRQ7 контроллера ; с выхода INT контроллера на вход INTR процессора подается сигнал запроса от наиболее приоритетного источника. При получении ответного сигнала INTA контроллер выдает номер прерывания на шину данных; номер образуется суммой IRQ номера с некоторым базовым значением.

При поступлении заявок от различных источников выполняется заявка с наименьшим номером IRQ. Имеется возможность выборочно заблокировать заявки от отдельных IRQ входов. Блокировку, или маскирование заявок, а также выбор заявкки с наибольшим приоритетом обеспечивают три байтовых регистра iMR, iRr,iSr.

Каждая из линий IRQx в контроллере подключена к разряду x регистра iRr через входной вентиль. Вентиль открыт, если разряд x в регистре маски сброшен. Регистр маски подключен к порту 021.

Виды прерываний: Внешние и внутренние. Внутренние не могут быть замаскированы. Внешние бывают Маскируемые и несакируемые. Внутренние : Сбой(Fault), Ловушка(Trap), Фатальная ошибка(Abort).

Примером передачи управления обработчику прерыванй может служит инструкция int x. Информация об адресе обработчика прерывания хранится в таблице векторов прерывний. Инструкция int x позволяет напрямую обратиться к процедуре, адрес которой задан в таблице векторов по адресу x.

Пример:

Вопрос 60. Взаимосвязь обработчиков событий в ОС при наличии аппаратных событий, внутренних прерываний, исключений

Вопрос 61. Реализация ISR при необходимости программного отключения отдельного устройства. При необходимости полной замены обработчика. Управление контроллером прерываний. Примеры кода.

Вот тут я не уверен.
Для отключения отдельного устройства в регистр масок контроллера прерываний необходимо записать значения, позволяющее замаскировать порт к которуму будет подключено соответствующее устройство. Для этого нужно обратиться к порту PORTy контроллера прерываний и записать в него в разряд iMrx единицу для маскирования соответствующего прерывания от устройства.

```
Mov dx, porty//port
mov ax ,08h//маска
out dx, ax//посылаем
```

Для полной замены обработчика нужно написать свой обработчик, который помимо всего прочего должен сбрасывать заявку в регистре контроллера прерываний. В таблице прерываний заменить адрес старого обработчика на адрес нового. Для воостановления старого обработчика необходимо записать его адрес в таблицу векторов прерываний.

Замена обработчика:

```
lea     ax,new// new – метка нового обработчика
mov     dx,cs
mov     ds,0//устанавливаем нулевой сегмент
xchg    w [n_vect*4], ax
xchng   w [n_vect*4], dx//замена вектора прерывания
mov     ds, cs//восстанавливаем сегмент

mov     w old_v, ax//сохраняем старый обработчик
mov     w old_v, dx
```

Вопрос 62. Способы отключения прерываний на критических участках выполнения кода ОС или приложения.

Для отключения прерываний на критических участках кода можно использовать несколько возможностей: 1) замаскировать необходимые прерывания в регистре iMrx контроллера прерываний

2)снять флаг разрешения прерываний процессора(флаг I).

1.Для отключения отдельного устройства в регистр масок контроллера прерываний необходимо записать значения, позволяющее замаскировать порт к которуму будет подключено соответствующее устройство. Для этого нужно обратиться к порту PORTy контроллера прерываний и записать в него в разряд iMrx единицу для маскирования соответствующего прерывания от устройства.

```
Mov dx, porty//port
mov ax ,08h//маска
out dx, ax//посылаем
```

2.Для снятия/установки флага I в ассемблере существуют команды cli/sti. Они позволяют заблокировать все внешние прерывания от устройств. Но запрос через дополнительный вход процессора NMI(Non-Maskable Interrupt) не блокируется и обслуживается сразу независимо от флага I. Ко входу NMI обычно подключены сигналы фатальных системных ошибок, например сгнал ошибки четности или обращения к памяти.

Вопрос 63. Табличный способ реализации системных вызовов, примеры в соответствующих ОС