**Абстракции операционной системы, интерфейс к программам POSIX**

* 1. **Виртуальные и физические ресурсы в вычислительной системе. Понятие уровней взаимодействия компонент вычислительной системы (пользовательские программы, абстракции операц. системы, драйверы, аппаратура).**

Вычислительная система (ВС) – совокупность аппаратных и программных средств, функционирующих в единой системе и предназначенных для решения задач определенного класса.

Структура вычислительной системы:

|  |
| --- |
| *6. Прикладные системы* |
| *5.Системы программирования* | | |
| *4. Системные программы* | | | | |
| *3. Управление логическими ресурсами* | | | | | | |
| *2. Управление физическими ресурсами (драйвера)* | | | | | | | | |
| *1. Аппаратура (системные шины, память, т. д)* | | | | | | | | | | |

По мере движения от аппаратуры к пользователю нарастает уровень абстрагирования, а сложность описания падает.

Физические ресурсы: процессор, оперативная память, внешнее устройство.

### Управление физическими ресурсами

Драйвер физического устройства – программа, основанная на использовании команд управления физического устройства и для взаимодействия с ним. Предоставляет упрощенный интерфейс для упрощенного доступа к данному физическому ресурсу.

### Управление логическими/виртуальными ресурсами.

Логическое/виртуальное устройство (ресурс) – устройство, некоторые эксплуатационные характеристики которого реализованы программным образом.

Ресурсы вычислительной системы – совокупность всех физических и виртуальных ресурсов. Одной из характеристик ресурсов является их конечность, следовательно, возникает конкуренция за обладание ресурсом между его программными потребителями.

Средства программирования, доступные на уровнях управления ресурсами ВС:

* система команд компьютера
* программные интерфейсы устройств (как физических, так и виртуальных
* )

*Операционная система* – это комплекс программ, обеспечивающий управление ресурсами вычислительной системы. Пользователю же доступна система команд.

* 1. **Понятие дистрибутива операционной системы, системных компонент. Понятие пакета программного обеспечения, репозитории пакетов.**

Дистрибутив содержит программы для начальной инициализации системы, программу-установщик и набор спец файлов, содержащих пакеты.

Типы дистрибутив:

1.Основанные на сборке на машине (Gentoo, Slackware)

2.Множество установленных бинарных пакетов с доступным исходником (Centos, Debian, AltLinux)

3.Без исходников (Ubuntu)

4.С подпиской за деньги (Suse, Fedora)

Пакет – набор взаимосвязанных модулей, предназначенных для решения задач некоторого класса.

Модуль – совокупность небольших независимых блоков, законченный фрагмент программы, файл с исходником.

Unpack

Preconfig

Config

Preinstall

Install

Postinstall

Пакетные менеджер – набор утилит для управления пакетами.

Утилиты – программы (команды) для выполнения задач.

Типы пакетных менеджеров:

Rpm .rpm

Dpkg .deb

Portage .tar.gz

Emerge .tar.gz

Репозиторий – место, где хранятся и поддерживаются данные для дальнейшего распространения по сети. CD/DVD тоже могут быть репозиторием.

* 1. **Обычные (тяжелые) процессы (порождение, завершение, ожидание завершения). Жизненный цикл процесса, состояния процесса. Замена тела процесса (системный вызов execve и семейство функций exec).**

Процесс – нечто, обладающее ресурсами.

Обычные процессы - это процессы, выполняющиеся внутри защищенных участков памяти ОС, то есть имеющие собственные виртуальные адресные пространства для статических и динамических данных. Т.к. управление процессами тесно связано с управлением и защитой памяти, переключение процессора с выполнения одного процесса на выполнение другого является достаточно дорогой операцией.

Порождается процесс системным вызовом fork. При вызове fork:

1.ОС заводит новую запись в таблице процессов;

2.Модифицирует приоритет;

3.Возня с таблицами страниц обоих процессов

1)Отец и сын обращаются к одному куску ОП. Копируем таблицу страниц одного и запрашиваем у ОС кусок ОП и перезаписываем все туда;

2)Ленивое копирование(не целые таблицы, а страничками) Делаем read only(RO) одну из таблиц. При попытке записи другая таблица перезаписывается. Если ОП переполняется:

1)Swap(Из ОП часть уходит в hhd)

2)ООМ Киллер(SIG KILL). Убиваем процесс, который жрет больше всех памяти.

При замене тела все ровно сперва вызывается fork и только потом меняется код, то есть системный вызов execve.

Execve выставляет pid, заменяет код, сегмент данных, стек.

Прототип: int execve (const char\* path, char\*\* args, char\*\* env)

В execve args и env должны указывать в никуда, чтобы знать, что массив закончился.

Переменное окружение – строка с полным путем до обрабатываемого файла. Перем окружение

является глобальным массивом.

Семейство exec:

Exec l, v, p

L – list;

V – вектор (переменное количество параметров);

P – искать в переменном окружении path

Примеры исп: execlp, execvpl, т.д. Порядок не влияет.

* 1. **Нити (легкие) процессы (порождение, завершение, ожидание завершения) библиотека pthread.**

Порождаются системным вызовом clone.

Pthread\_create является оберткой над clone. Описан в библиотеке pthread.h

Процесс и нити (потоки) обладают одним и тем же кодом, данными и кучами в ОП, но разными стеками. Используется одна и та же область памяти, не дублируется.

Т.е процесс является основной нитью, потоки – дополнительные нити, которые появляются и завершаются по ходу процесса. Используются для многопоточности.

Прототип функции потока:

Void\* func(void\* arg);

Нити создаются функцией pthread\_create.

Int pthread\_create(pthread\_t,NULL,func,&thread\_int[i]);

Где последний параметр – указатель на то, что получает func на вход.

Функция pthread\_join() получает возвращенное значение функции потока.

Pthread\_exit аналог \_exit.

* 1. **Идентификаторы пользователя, родителя, группы и сессии у процессов, эффективный идентификатор пользователя и группы: повышение и понижение прав процесса, в каких случаях это необходимо. Процессы демоны в UNIX.**

Uid-идентификатор пользователя

Gid-идентификатор группы

Euid-эффективный

Egid-эффективный

Egid и euid используются при просмотре прав

Egid может быть любым, при условии, что euid=0

Euid из 0 может стать любым

Pid-идентификация процесса

Ppid-идентификация отца

Процесс init запускается первым и он является в той или иной степени прародителем всех процессов.

Права на ФС. Спец биты и 3 группы(rwx, x – права на выполнение (может и не быть))

Также есть числа owner и group.

Если совпадают owner и euid и есть х, то относится к первой группе.

Если совпадают group и egid и есть х, то ко второй.

Иначе, к третьей группе.

Suid бит нужен чтобы поднимать себе права до root.

Возможен переход прав от 100 до 0 и с 0 до 200, но нельзя перейти на прямую с 100 на 200.

Демоны – процессы, запускаемые системой, работающие в фоновом режиме и не имеющие управляющего терминала. Задачи демонов: вход пользователя, управление оборудованием, управление выполнением заданий по расписанию и т.д.

Демон – способ жизни. При становлении демоном нужно понизить себе права от root на другого пользователя.

* 1. **Работа с файлами, файловые дескрипторы, таблицы открытых файлов, системные вызовы: open, close, ioctl, fcntl, dup, dup2, read, write.**

Файловый дескриптор – некоторое неотрицательное число, позволяющее идентифицировать соответствующий файл.

Fcntl выполняет управляющие операции над файловым дескриптором.

Ioctl делает все стандартные манипуляции с файлами.

Dup дублирует строку из таблицы открытых файлов процесса, т.е. две строчки указывают на одну и ту же в таблице открытых файлов ОС.

Dup2 копирует из одной строчки в другую.

Open возвращает число, указанное в таблице открытых файлов процесса.

Если появляется процесс, это регистрируется в соответствующей таблице.

Параметры для open:

O\_SYNC – запрет буферизации на операции I\O

O\_CLOEXEC – открытые файлы остаются открытыми, даже если поменялся код

O\_FARGEFILE – возможность открыть большие файлы

O\_DIRECTORY – каталог открывается как файл

O\_TRUNC – очищает файл

О\_CREAT – создает, если не существует

O\_EXCL – если существует, то не создает

O\_APPEND – продолжает

O\_RDONLY

O\_WRONLY

O\_RDWR

**7. Сигналы, обработчики сигналов, доставка сигналов, сигнальная маска; системные вызовы signal, sigprocmask, pause, kill. Работа с таймером функции alarm и sleep.**

Сигнал – высокоуровневая концепция, аналогичная прерыванию. Виды сигналов:

1.”Просто сигналы”. Ячейки, 1, если появился. Сигналы слипаются, если случаются два одинаковые сигналы

2. Сигналы реально времени. Складываются в очередь. Ни с чем не ассоциированы.

Способы доставки сигналов:

1.Игнорируются

2.Обработчик по умолчанию от ОС

3.Запуск обработчика пользователя

Обработчик сигнала – поток. Он не может быть прерван приходом того же сигнала, в отличии от прерывании.

SIGTERM - завершает процесс. Приличный способ. Может перехватиться

SIGKILL – убивает процесс. Нельзя игнорировать или перехватывать

Гадкие сигналы, отправляющийся при возникновении программного прерывания:

1)SIGILL – не может быть исполнен, так как нет инструкции

2)SIGBUS – нехороший разговор с ОП

3)SIGSEGV – segmentation fault

SIGSTOP – остановить процесс. Нельзя игнорировать или перехватывать

SIGCONT – возобновить процесс

SIGINT – ctrl + c

SIGQUIT – приходит от init

SIGCHLD – отцу о том, что сын завершен или приостановлен

SIGUSR1, SIGUSR2 – по инициативе программиста

Системные вызовы:

int kill(pid\_t pid, int sig\_no) – отправить сигнал, sig\_no – номер сигнала. Если

* Pid=0, сигнал посылается всем процессам с тем же gid
* Pid= -1, всем процессам, кроме вызвавшего
* Pid< -1, по модули всем с тем же gid
* Pid>1, тому процессу

Sigprocmask() принимает на вход битовый массив и меняет список маскируемых сигналов и 1 маскируются.

Свой обработчик сигнала:

void handler(int sig)

Установить обработчик сигнала:

typedef void (\*handler\_t)(int);

handler\_t signal(int signo, handler\_t hdl), где signo – номер сигнала, hd1 может принимать значения

SIG\_IGN – сигнал должен игнорироваться или SIG\_DFL – стандартная обработка

Sigaction() продвинутый образ signal().

Pause() приостанавливает выполнение программы до получения неигнорируемого сигнала.

Int alarm(unsigned int seconds) посылает процессу сигнал SIGALRM через seconds секунд. Возвращает ненулевое число (оставшееся время), если заказ был уже сделан и ноль, иначе.

Unsigned int sleep(unsigned int seconds) переводит процесс в режим ожидания либо на seconds секунд, либо на время до получения сигнала, который не может быть проигнорирован.

**8. Средства синхронизации процессов: семафоры, двоичные семафоры (mutex); IPC (Inter Process Communication) объекты и время их жизни. Рассказ на примере IPC SystemV.**

Виды семафор: двоичный (mutex) и недвоичный.

Критическая секция – участок кода, в котором одновременно может находиться только один процесс.

Операции над семафорами:

1.Up-поднять семафор. Может только тот, кто вышел

2.Down.

1)Если поднят, то отпускаем

2)Если опущен, процесс блокируем до поднятия семафора

Значение семафора не может быть отрицательным.

Монитор Хоара – функция, которая может быть выполнена только одним процессом.

Двоичные семафоры обладают двумя операциями: lock(down) и unlock(up)

IPC-объект – нечто, что живет внутри ОС. Исчезает, когда ОС перезагружается.

Системные вызовы:

Int semget(key\_t key, int num\_sems, int flags) получаем семафоры. Второй аргумент – кол-во семафор, третий – флажки (IPC\_CREAT, IPC\_EXCL)

Struct sembuf{

Int sem\_num //номер в массиве

Short sem\_op//как менять значение

Int sem\_flags//по факту 0

}

Int semop(int sem\_id, struct sembuf \*ops, int num\_ops) применяет операции над массивом семафор

Первый аргумент – возвращаемое значение semget, третий – количество операции

Если

1)semop = 0, процесс блокируется, пока значение семафора не будет равен нулю

2)semop > 0, выполняется up и значение семафора увеличивается на semop

3)semop < 0, уменьшается на semop если возможно, иначе – блокируется

Int semctl(int sem\_id, int semnum, int cmd, ...) операции, определенные параметром cmd

Значения для cmd:

SETVAL – установить значение

GETVAL – подсмотреть значение в текущий момент

**9. IPC (Inter Process Communication) объекты и время их жизни. Рассказ на примере IPC**

**SystemV. Разделяемая память IPC, присоединение/отсоединение к адресному**

**пространству процесса, системные вызовы для этого.**

Разделяемая память – самый быстрый механизм межпроцессного взаимодействия, в котором несколько процессов имеют доступ к одному и тому же куску ОП. Кусок иденфицириуется ключом.

Системные вызовы:

Int shmget(key\_t key, int size, int shmflag) получение разделяемой памяти.

Void\* shmat(int shmid, void\* addr, int mode) присоединение куска виртуальной памяти. Возвращает указатель на начало региона виртуальной памяти. Второй параметр желательно должен быть NULL, т.к. иначе указывается конкретный адрес.

Int shmdt(void\* shmaddr) отсоединяет.

Int shmctl(int shmid, int cmd, struct shmid \*buf) делает различные операции. Если в cmd задан IPC\_RMID, то сегмент разделяемой памяти будет удален.

**10. IPC (Inter Process Communication) объекты и время их жизни. Рассказ на примере IPC**

**SystemV. Очереди сообщений. Синхронизация процессов при помощи сообщений.**

В очереди хранятся тексты разной длины. Каждое сообщение содержит type (не равный нулю) и body (текст, набор байтов).

Системные вызовы:

Int msgget(key\_t key, int msgflag) создает очередь и возвращает идентификатор

Struct msgbuf(

Unsigned long type;

Char msgtext[MSGSIZE]; //данные

}

Int msgsnd(int msgid, struct msgbuf \*msgp, size\_t msgsz, int msgflg) добавляет в очередь сообщение

Int msgrcv(int msgid, struct msgbuf \*msgp, size\_t msgsz, long msgtyp, int msgflg) достать сообщение из очереди. Если msgtyp

1. = 0, выковыривать из очереди точно так, как было положено
2. > 0, выковыривать те сообщения, msgtyp которых совпадает
3. < 0, выковыривать те, модуль которых больше модули числа

Возвращается длина сообщения.

Int msgctl(int msgid, int cmd, struct msgbuf \*msgp) зависит от параметра cmd. Если cmd является

IPC\_STAT – получить информацию о очереди сообщения

IPC\_RMID – удалить очередь сообщения

**11. Именованные, неименованные каналы: (pipe, mkfifo, PF\_UNIX (для socket)).**

Канал – объект ядра, предназначенный для передачи данных. Каналы имеют два конца: для чтения и для записи.

Неименованный канал.

int pipe(int fd[2]) создает канал. Fd[0] для чтения и fd[1] для записи.

Канал не имеет и имени и к нему нельзя подключиться из другого процесса. Соответственно, могут подключиться только родственные процессы, чей общий предок создал канал.

Если канал пуст, то при операции чтения зависнет

Запись в канал с переполненным буфером так же зависнет

Если закрыты все дескрипторы записи, благополучно закончится с EOF

Если закрыты все дескрипторы чтения, то при первой попытке записи процесс получит SIGPIPE, который завершает процесс.

Конвейер – запуск нескольких программ, при котором информация, выдаваемая первой программой на вывод, поступает второй на ввод, вывод второй – на ввод третьей и т.д. Важно закрывать все дескрипторы, связанные с каналом во всех процессах.

Именованный канал.

int mkfifo(const char \*pathname, int perm) создает файл FIFO. Второй параметр – права доступа, аналогичные open().

В отличии от неименованного канала, закрытие всех дескрипторов чтения или записи не означает непригодность канала, т.к. в любой момент дескрипторы могут появиться.

Создание файла FIFO еще не означает создание объекта канала. Это произойдет только в том случае, когда его открывают с помощью open() и будет существовать до тех пор, пока существует хотя бы один связанный с ним дескриптор.

Если файл открыт только на запись, процесс повиснет и будет ждать, пока кто-то не откроет на чтение.

PF\_UNIX

Подгрузить модуль в ядро, если не подгружен

Int socket(int protocol\_family; int type; int protocol) второй параметр может принимать значения

SOCK\_STREAM - поток байтов

SOCK\_DGRAM – сообщение

Третий параметр равен нулю.

PF\_UNIX сетевое взаимодействие на локальном уровне

**Файловые системы и устройства хранения данных**

**12(1). Виды устройств хранения данных (жесткие диски, твердотельные накопители, оптические диски, магнитные ленты), проблемы возникающие при хранении данных на них (проблема доступа, надёжности, фрагментации, износа); механизм SMART. (Алгоритмы**

**минимизации перемещений по магнитным и оптическим дискам).**

Hdd диски

Шпиндель. Вращает диски. От скорости вращения шпинделя зависит скорость доступа к данным

Пластина диска состоит из поверхностей, поверхности делятся на дорожки – кольцевые области. Совокупность дорожек – цилиндр. Дорожки делятся на ровные отрезки – секторы.

Головка. Считывает/записывает данные.

Магнит. Притягивает/оттягивает головку.

Моторчик вращает шпиндель, обеспечивая равномерность.

Координаты состоят из:

1)Номер поверхности

2)Номер цилиндра

3)Номер сектора

Твердотельные накопители – SSD диски с быстрым доступом данных.

Оптические диски – носитель, чтение с которого ведется с помощью лазерного излучения.

Магнитные ленты – носитель в виде гибкой ленты, покрытой тонким магнитным слоем.

LBA – механизм адресации к блоку данных, где каждый блок имеет свой номер LBA

SMART – технология оценки состояния жесткого диска аппаратурой самодиагностики. SMART производит наблюдение за основными характеристиками накопителя и каждая получает оценку. Результатом определяется общее состояние диска на данный момент и примерный срок продолжительности исправной работы.

Минимизация перемещений головки является целью, т.к. на этот процесс требуется много ресурсов.

Step scan. Упорядочиваем все операции. До конца вниз и до конца верх

Nstep scan. Упорядочиваем только определенный кусок. Дальше подставляем в нужные места оставшийся операции

**13(2). Организация единого виртуального дискового пространства. Механизм LVM; Обеспечение надёжности и быстродействия при хранения данных на нескольких устройствах: RAID 0,1,2,3,5.**

Механизм LVM(Logical Volume Manager) позволяет несколько дисков или их области использовать как один логический том. Разделы дисков разбиваются на блоки и объединяются в виртуальный том, группу томов, которая разбивается на логические тома. Логический том можно перераспределить на другие определенные части и, освободив ЗУ, попросту оставить пустым (или выкинуть).

Единицей LVM механизма можно представить RAID. RAID – виртуализация данных, объединяющая диски в логический элемент. Назначения: увеличение скорости доступа и надежности хранения.

RAID 0. Отображение из виртуальной памяти блоков данных в разные жесткие диски. Т.к. доступ к разным дискам идет независимо, значительно увеличивается скорость, но теряется надежность

RAID 1. Копирование информации с одного диска на все, т.е. все диски будут содержать одно и то же. Надежно, но скорость может потенциально падать и не увеличивается вовсе.

RAID 2. Чем больше хотим восстановить, тем больше нужно хранить доп. информации. Работает на основе кода Хемминга, но эта операция является дорогостоящей.

RAID 3. Диск контрольной суммы. Каждый блок в этом диске определяется: сумма = блок1 xor блок2. Если 2 диска, то это RAID 1. Не оптимален тем, что нужно каждый раз обращаться к диску контрольной суммы.

RAID 5. Циклическое распространение контрольной суммы.

**14(3). Таблица разделов на блочном устройстве. MBR, GPT.**

Таблица разделов обеспечивает возможность хранить на жестком диске несколько ОС.

При загрузке компьютера BIOS считывает MBR в ОП и передает управление коду находящегося там начальной загрузки.

В hdd выделяется под MBR 512 байтов для загрузчика первой стадии, таблица разделов и копия таблицы разделов. В таблице разделов каждая запись описывает один из разделов hdd. Может быть максимум 4 раздела. Также, в таблице разделов хранится boot flag. Boot flag указывает откуда грузить ОС.

Проблемы MBR:

1. Только 4 раздела
2. Нельзя сделать размер больше терабайта в связи с ограничением из-за размера поля под хранение размеров.

EFI – интерфейс между ОС и программами. EFI предназначен для замены BIOS. GPT используется EFI там, где BIOS использует MBR. В GPT допускается большее количество разделов. У каждого раздела есть подпись для защиты, который является неким ключом. Также, в GPT есть дублирование в конце диска.

**16(5). Типы файлов, права доступа на файлы и каталоги. Текущий каталог, системный вызов chroot. Индексный дескриптор, системные вызовы stat, fstat, lstat. Структура каталогов файловой системы UNIX, назначение каталогов и некоторых особенно важных файлов.**

Средство, отвечающее за хранение файлов, называется файловой системой. При создании на диске ФС создается один каталог, называемый корневым. В Unix ФС представляет собой единое дерево каталогов

chroot — операция изменения корневого каталога. Программа, запущенная с изменённым корневым каталогом, будет иметь доступ только к файлам, содержащимся в данном каталоге. В Unix каталоги хранят только имя файла и файловый дескриптор. Вся прочая информация связывается с файловым дескриптором.

Хранимая структура данных, содержащая всю информацию о файле (размер, владелец, группа, права, ссылка разблокировки, на файл устройства), называется индексным дескриптором. Функция, предназначенная для удаления файла, называется unlink(). Для изменения прав доступа к файлам используется команда chmod. Чтобы начать работу с файлом, его необходимо открыть. Это делается системным вызовом int open(const char \*name, int mode); Параметр mode задает режим. Основные режимы O\_RDONLY, O\_WRONLY,O\_RDWR. Чтение файла int read(int fd, void \*buf, int len); fd задает файловый дескриптор; buf указывает на буфер; len сообщает вызову размер буфера.

Типы объектов ФС:

1.Регулярный файл

2.Каталог

3.Файл устройства

4.Unix сокет. Сокеты – набор функции для обеспечения обмена данными между процессами

5.FIFO

6.Символическая ссылка. Файл, который содержит путь в другой файл. В отличии от ярлыка, ОС сама интерпретирует запись (в ярлыках это делает внешняя программа)

Системный вызов stat возвращает информацию о файле

Lstat идентичен stat, но в случае символьных ссылок возвращает информацию о ссылке

Fstat возвращает информацию об открытом файле, возвращаемым open

**Аппаратура и работа с ней**

**17(1). Иерархия памяти, Устройство памяти, Основные характеристики памяти. КЭШ L1/L2/L3, расслоение памяти.**

Иерархия памяти:

1)РОН, КЭШ L1;

2)КЭШ L2;

3)ОП;

4)ВУ прямого доступа с внутренней КЭШ буферизацией;

5)ВУ прямого доступа без внутренней КЭШ буферизации;

6)ВУ долговременного хранения данных.

КЭШ - промежуточный буфер с быстрым доступом, содержащий информацию, которая может быть запрошена с наибольшей вероятностью

1)L1. Самый быстрый. Он является неотъемлемой частью процессора, поскольку расположен на одном с ним кристалле и входит в состав функциональных блоков.

2)L2 находится между ЦП и ОЗУ, это устройство быстрее ОП, но медленнее L1.

3)ОП - часть системы компьютерной памяти, в которой во время работы компьютера хранится выполняемый машинный код (программы), а также входные, выходные и промежуточные данные, обрабатываемые процессором.

4)Устройства, аппаратное управление которых имеет кэш-буфер, менее эффективно по срав. с ОП, но за счет внутр. кеширования сокращается реальное кол-во обращений к устройству, тем самым получается существ. повышение производительности работы устройства.

5)Менее эффективны, но предназначены для оперативного доступа к данным, т.е. это устройства которые используются в программе для организации внешнего хранения и доступа за данными.

6)Устройства для долгого хранения информации, к ним можно отнести устройства прямого доступа и последовательного доступа.

Основные характеристика памяти

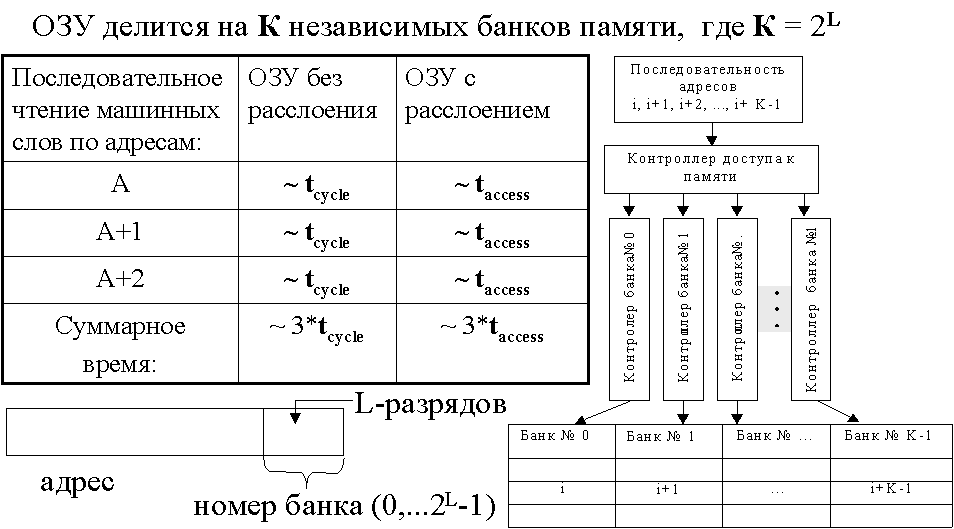
1) объем ОП

2) tread, twrite, tcycle, trefresh

3) частота шины памяти

Расслоение памяти

Расслоение ОЗУ - один из путей аппаратного решения проблемы дисбаланса между скоростью доступа к данным, находящимся в ОП, и производительностью процесса. Так же используются контроллеры.



Использование расслоения памяти. Физически ОЗУ представимо в виде объединения k устройств, способных хранить одинаковое количество информации и способных взаимодействовать с процессором независимо друг от друга. При этом адресное пространство ВС организовано таким образом, что подряд идущие адреса, или ячейки памяти, находятся в соседних устройствах (блоках) оперативной памяти. Программа состоит (в большей степени) из линейных участков. Если использовать этот параллелизм, то можно организовать в процессоре еще один буфер, который организован так же, но в котором размещаются машинные команды. За счет того, что есть параллельно работающие устройства, то этот буфер автоматически заполняется вперед. Т.е. заодно обращение можно прочесть k машинных слов и разместить их в этом буфере. Далее, когда нужна очередная команда (ее адрес находится в счетчике команд), происходит ее поиск (по адресу) в буфере, и если такая команда есть, то она считывается. Если такой команды нет, то опять-таки работает внутренний алгоритм выталкивания строки, новая строка считывается из памяти и копируется в буфер команд. Расслоение памяти в идеале увеличивает скорость доступа в k раз, плюс буфер команд позволяет сократить обращения к ОЗУ.

**18(2). Процессор. Принципы работы процессора и основные компоненты окружения процессора: такт, системы команд, шины (процессор –- память, процессор — внешние устройства ), регистры (назначение регистров), память.**

Регистры:

1) регистры общего назначения - для арифметических операции

2) регистр флагов — регистр процессора, отражающий текущее состояние процессора.

3) счетчик команд — адрес след. команды

4) служебные регистры

Служебные регистры:

1) адресные регистры : вершина стека , селекторы сегментов(где начинается код) — ESS,EDS

2) работы с прерываниями: регистр прерывания (прерывание — событие в компе. при котором в процессоре происходит предопределенная послед. действий ), регистр маски, указатель на вектор прерывания, регистр обработки прерывания

3) регистр для пошаговой отладки — trap

4) управление страницами памяти ОП.

Процессор — устройство, обеспечивающее последовательное выполнение машинных команд, составляющих программу

- Блок управления процессором берет из оперативной памяти, в которую загружена программа, определенные значения (данные) и команды которые необходимо выполнить (инструкции). Эти данные загружаются в кэш-память процессора.

- Из буферной памяти процессора (кэша) инструкции и полученные данные записываются в регистры. Инструкции помещаются в регистры команд, а значения в регистры данных.

- Арифметико-логическое устройство считывает инструкции и данные из соответствующих регистров процессора и выполняет эти команды над полученными числами.

- Результаты снова записываются в регистры и если вычисления закончены в буферную память процессора. Регистров у процессора совсем немного, поэтому он вынужден хранить промежуточные результаты в кэш-памяти различного уровня.

- Новые данные и команды, необходимые для расчетов, загружаются в кэш верхнего уровня (из третьего во второй, из второго в первый), а неиспользуемые данные наоборот в кэш нижнего уровня.

- Если цикл вычислений закончен, результат записывается в оперативную память компьютера для высвобождения места в буферной памяти процессора для новых вычислений. То же самой происходит при переполнении данными кэш-памяти: неиспользуемые данные перемещаются в кэш нижнего уровня или в оперативную память.

Шины

Шина памяти предназначена для передачи информации между процессором и основное памятью системы.

Тут должен быть рисунок: шина памяти справа ОП, от нее вправо верх флэшка BIOS. От шины памяти влево процессор. От процессора влево часы. Процессор вниз контроллер. Ниже контроллера PSI-express. От нее 4 линии : видеокарта, контроллер SATA, контроллер USB, сетевая карта.

**19(3). Таймеры в процессоре, принцип организации watchdog устройств. Использование операционной системой различных таймеров.**

Часы подключаются к процессору через шину вместе с ВУ.

Существуют астрологические часы cmos. Большое число, измеряемое количеством секунд. Тикают за счет аккумулятора. В Linux насчитывается начиная с 1971 года.

Часы используются в генераторах тактовой частоты, использующихся для синхронизации различных процессов.

Для счетчиков, привязанных к генераторам тактовой частоты. В регистр (счетчик) пишется число и по истечению происходит прерывание.

Wathcdog. Внешний и внутренний.

Внешний по истечению времени, если таймер не сбрасывался, делает перезагрузку данного внешнего устройства в целях не зависания.

Внутренний используется для отладки ОС. Ставится определенное число и до того как дойдет до этого числа, ОС должен сбрасывать. Если не сбросило, значит зависло и происходит в частности синий экран смерти.

Таймер также используется при планировании переключения процесса ОС-ом.

При ОС разделения времени кванты времени меняются по прерыванию по таймеру.

**20(4). Виртуальная память и защита памяти, сегментностраничная организация памяти, кольца защиты. Таблицы сегментов, таблицы страниц; назначение принципы работы операционной системы со страницами и сегментами. TLB-KЭШ. Системный вызов: поддержка в аппаратуре системного вызова.**

Виртуальная память делится:

1.Страничка, никуда не отображающаяся (для того, чтобы при выходе из границы была ошибка)

2.Не отображается никуда(на всякий случай)

3.Отображаемые адреса

4.Область DMA обмена (Асинхронный режим работы с ВУ)

5.Код ядра

6.Данные ядра

7.Куча ядра

8.Свободная память (сжимается с двух сторон)

9.Стек. На этом заканчивается память ядра

10.Код

11.Данные

12.Свободно

13.Стек

Программа – набор инструкции, размещенных в памяти.

Программа пользуется виртуальной памятью (те адреса, которые используются в процессе)

1.Сегментная организация

Сегмент – диапазон адресов и набор допустимых команд

Таблица сегментов находится в ОП. Адрес записан в регистре gdtr(global) или ldtr(local)

Бывают 4 типа сегментов: D-данные, I-инструкции (команды), S-стек, system

Колонны таблиц: Исп\не исп, начальный адрес, размер, единица измерения, уровень привилегии, тип сегмента.

2.Страничная организация. Отображается из виртуальной в ОП.

Если происходит обращение в адрес виртуальной памяти, который не отображен, то происходит прерывание.

Колонны таблиц: Отобр\не отобр, адрес в ОП, бит грязности (ставится если обращено с записью и отправляется в swap), бит доступа (ставится, если обращено с чтением), уровень привилегии, права(RO,RW), размер страниц в попугаях.

Чтобы не хранить такие большие объемы можно хранить таблицу в виде дерева, где хранятся таблицы или каталоги соответствующего уровня. Может быть пустым, если данный диапазон не используется.

TLB-КЭШ используется для ускорения трансляции виртуального адреса в физический адрес.

**21(5). Механизм прерываний, работа с переферией. Базовые принципы работы с внешними устройствами (синхронный/асинхронный режимы). Контролер прерываний. Маска прерываний, уход в прерывание и возврат из прерывания. Регистровые окна.**

Основные механизмы прерывания: APIC и MSI. Второй считает, что прерывания приходят в каком-то порядке, а первый делает перераспределение, в результате, второй работает быстрее.

Могут быть два вида прерывании:

Внешние. Приходящие из механизма

Внутренние. Внутренние ошибки процессора

Работа с ВУ бывает 3 типов:

1)Порты I/O. В основном только управляющие команды (вкл, выкл);

2)Диапазон отображаемых адресов. При отображении из виртуальной в ОП, некоторая часть виртуальной памяти может отображаться напрямую в устройства;

3)Механизм прерывания.

Работа ведется в двух режимах: синхронный и асинхронный.

В синхронном режиме работают только первые 2 пункта и взаимодействуют с медленными ВУ.

Асинхронный режим. DMA. Direct Memory Access.

Т.к. ОС еще в начале резервирует память в ОП для ВУ, плохо подключать их после начала работ.

Подключение ВУ:

1.Определение диапазона адресов в виртуальной и ОП

2.Определение “окна” в ОП

3.Сообщение устройству об “окне” через синхронный протокол сведения

4.Отправка данных в ОП устройством

5.Сигнал прерывания в APIC

6.Освобождение диапазона

Другие процессы не имеют доступа на данные куски памяти при записи.

Регистры прерывания и маски прерывания.

Если случилось прерывание и оно не замаскировано:

1.Основные регистры и обработчик прерывания отправляются в стек. В этот момент другие прерывания не обрабатываются. На выходе все прерывания будут замаскированы, кроме данного.

2.Обработчик размаскировывает те прерывания, которые не мешают. Обрабатывается ситуация и размаскировывается замаскированное с тем, что восстанавливается из стека регистр маски. Происходит iret(берет из стека все обратно)