Семинар №2

Вопрос адресации → ключевым при работе компьютера
доступ к данным по адресам
процессор Intel → две схемы управления памятью
→ сегм. по запросу } выполнение
→ стран. по запр. } по управл. памятью

Сегментный адрес
↳ DOS – когда шина адреса была 20-разрядная
⇒ ввели сегментные регистры → сдвигаются на 4 +4 разряда
Начальный адрес сегмента → в дескрипторе глоб. таблицы дескрипт
+ смещение (из команды)
(любая программ. считает что она начин. с нулевого адреса
⇒ смещение из команды (счетчик команд / инд. регистры /
регистр. указ / операнд из ком)
+ начало сегмента ⇒ линейный физ. адрес
Процессор Intel — лок. таблицы дескрипторов (16-разр)
⇒ не может содерж. лин. физ. адрес, но
по аналогии в LDTR может находится селектор
⇒ описывает сегмент памяти где нах.

виртуальное адресное простр. процесса
⇒ столько сколько процессов
Мы процессы не создаем перевод реальный → защищенный
работа с польз.
и клав.
опред. объем
доп. адр. пространства

Системные таблицы
(убрать сброс процессора
limit FFFF сегмент. данных / стека / видеобуфера / кода
(разрядность операндов — 16-разрядные сегменты)
защищ. режим 32-разр. регистры 16-разр.

Задание:
расписать дескр. сегментов по фрагментам
расписывать по битово
Смещение в реальном режиме не может превышать 16
$2^{20} = FFFFF_{16}$ – 1мв – 1024кв
Перед возвр. в реал. режим ffff – запись в теневые регистры
забыли запись в атрибуты

С каждым сегмент. регистр. → сопоставлен теневой в котором при обр. к сегм. р. записыв. инф. из дескриптора

=> чтобы исключить обращ. к табл. GDT которая находится в операт. памятью (ибо ОП - отдает от процессора - затратное действие

цикл обр. к памяти требует опред. кол-ва тактов.

Как часто процессор обращается к опер. пам.?
=> на каждой команды + форм. адреса по котор. обр к пам.

получ. физ адреса
инф. с дескр. записыв в теневой регистр (в процессоре)
=> исклю. обращ к GDT

зачем это делать? (лаба 2)

зачем ffff -? - максим. размер сегмента 1мв

Провести эксперимент закомментить

→ поднимается вопрос с теневыми регистрами

Мы определить объем доступного физ. пространства

Что делать?:

- какая память доступна (4Гб) в защищенном
  => объявить дескриптор (начиная со второго мв
  сокращ. байт или слово =>
  записываем сегментную разрыв. со своб
  если совпали => память
  микр. счетчик.

- какой сегмент?
  - DS - limit FFFFF именно запись

16р CS 16р DS => 32р код 32р CS
объем дост. пам. 32р DS

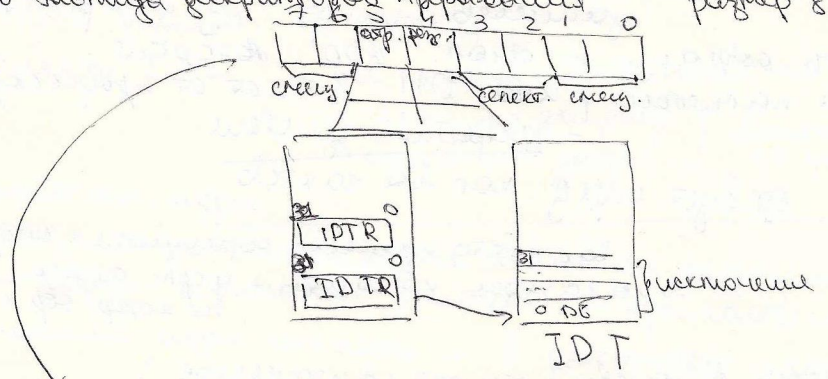
32р SS - прерыв. только в защищ. режиме

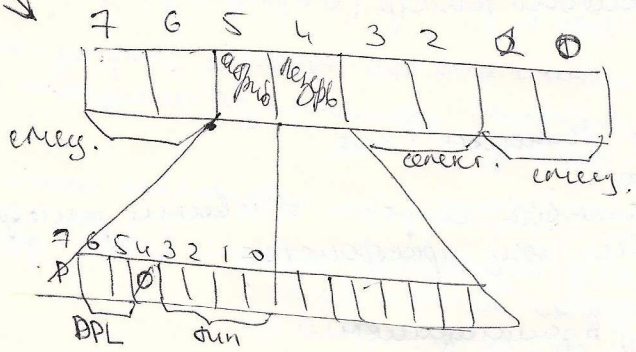
Обеспечить возм. работы с байт и слов.

=> написать обр. прерыв.

Прерывание в защищенном режиме

есть таблица дескрипторов прерывания размер 8 байт





Исключения:

0 – деление divide error

8 – double fault (искл. или прер. не прер. „полная система" => завершение работы компа)

13 – general protection (общая защита – спец. заплатка

11 – segment not present (сегмент отсутствует

14 – Page fault (страничное прерывание
когда процессор обращ. к ком. или данным отсутс. в
=> система должна загруз. страницу

Всего 18 – искл.

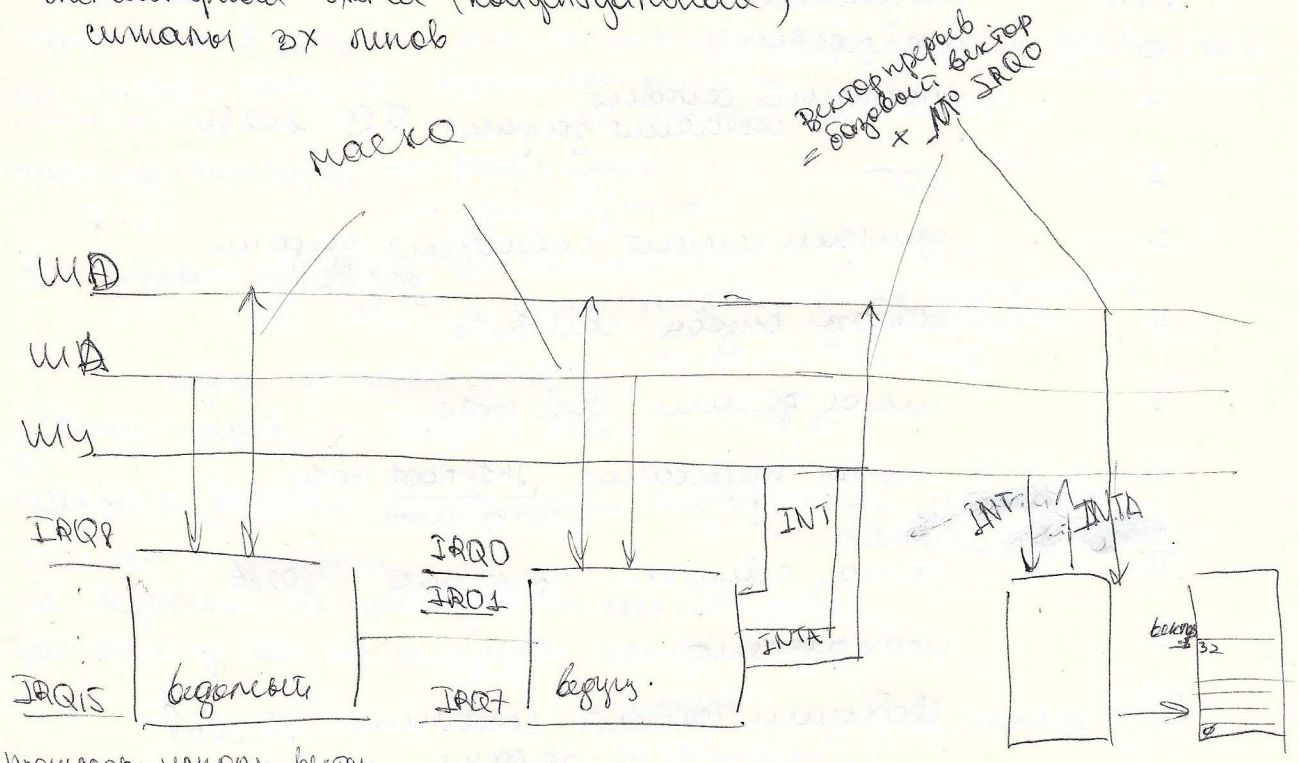
20 – 31 – зарезервированы

32 – 255 – определяется пользователем user define

Смещение = номер искл × 8

Адресовать прерывания (аппаратные)
через программ. контроллер прерываний

Элементарная схема (концептуальная)
сигналы 3х линий



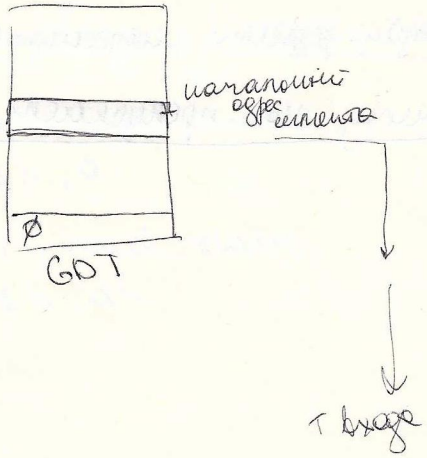
Процессор использ. вектор.

Базов. вектор ведущ. контр. - 8
=> INT8h

Взаимн. разделение
=> таблица дескрипт прерыв.

для получения смещения адреса вект. прерыв.

Для того чтобы адресовать прерывания надо перепрограммировать контроллер прерываний 32



Адресация аппаратных прерываний

Тип	назначение
0	неопределен.
1	свободный сегмент состояния задачи TSS 80286
2	LDT
3	занятый сегмент состояния задачи для 80286
4	шлюз вызова call Gate
5	шлюз задачи task Gate
6	шлюз прерывания Interrupt Gate аппар. прерыв.
7	шлюз ловушки Trapt Gate 80286
8	неопределено
9	свободный сегмент состояния задачи 80386↑
Ah	неопределено
Bh	занятый сегмент состояния задачи 80386↑
Ch	Call Gate
Dh	неопределено
Eh	шлюз прерыв. Interrupt Gate 80386↑
Fh	Trapt Gate 80386↑

В защищ. режиме адресация

Почему мы пропускаем первый мегабайт?

Семинар 3

Программа в защищ. режиме → нулевой уровень привилегий
Нет никакой защиты в реальном режиме
нет уровней привилегий

таблица векторов ← реальный
IDT ← защищенный

Если первые 32 дескр. ← исключения
32..255 ← дескр. которые может определить пользов.
отрис. от клавиатуры и систем таймер.

Вектор прерыв ← базы + IRQ..
⇓
перепрогр. базовый вектор ведущ. контроллера на новый равный 32
сигнал → импульс
на ведомый не приходит не каких прерыв.
замаскир. все на ведущем контр.
(1 - запрет
0 - разрешение) → маска ff
на ведущем разр. RQ0 RQ1 → маска fC
контроллер обращ. через порты к контроллеру
порт ведущего 21h
ведомого A1h

Сохранение масок ведущего и ведомого
8 входов → байт

mask_master db 0
mask_slave db 0

сохранение масок

```
in al, 21h
mov mask_master, al
in al, A1h
mov mask_slave, al
# установка новых масок
mov al, fCh
out A1h, al
mov al, ffh
out 21h, al
```

восстановление старых масок

```
mov al, mask_master
out 21h, al

mov al, mask_slave
out A1h, al
```

Запрет ~~зап~~ маскируемых и не маскируемых прерываний
(Запретить все прерывания)

cli - запрет маскируемых

70h - порт посылать команду 80h

Запрет маскируемых и немаскир.

mov al, 80h
out 70h, al # запрет не маскируемых
cli # запрет маскируемых

Разрешение прерываний

~~mov al, 30h # ла~~
~~out 30h, al~~

sti

~~xor~~
xor al, al
out 70h, al

Перепрограммирование ведущего контроллера

- ведомый не трогается
- ведомый такой же как ведущий

Слово команды исполнения

mov al, 11h ; СКИ1
out 20h, al
mov al, base_vec ; СКИ2
out 21h, al
mov al, 4 ; СКИ3 – контрол. соединения по линии IRQ2
out 21h, al
mov al, 1 ; СКИ4 – требуют EOI
out 21h, al

Линия A20

Спецификации:

XMS

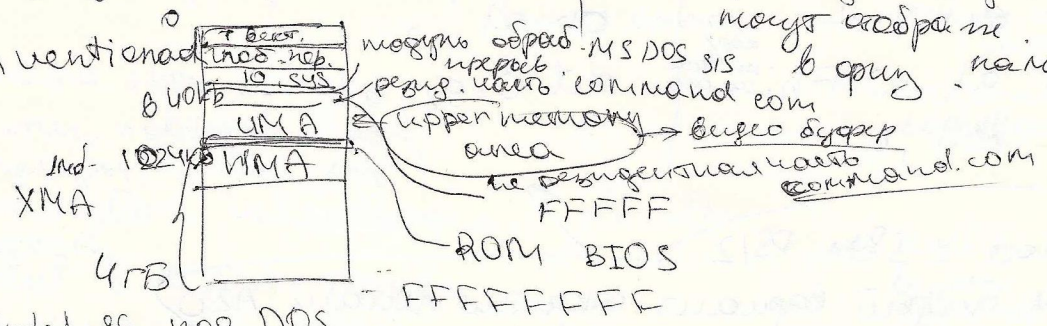
EMS { extended memory specification
expanded memory specificat

дополнительная доп память

расширенная доп память

оперирует областями памяти
в защищенном режиме

LIM... - версии
страничное преобр.
страницы на диске могут отображ. в физ. память



в Intel 86 под DOS

32р. шина данных
смещение: $2^{32} \to 4ГБ$

карты первого мегабайта

реальный

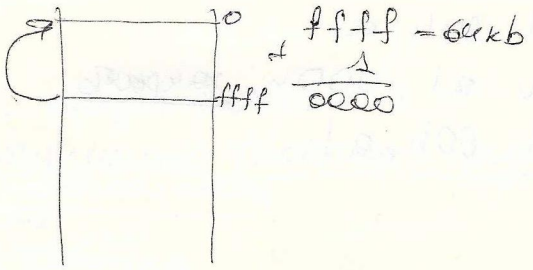
20 линий

0
1

0
1
20
31

заворачивание/завораживание

ffff + 1 = 00000 на практике адреса



для обратной совместимости аппаратно
компьютер стартует в реальном режиме
линия A20 закрыта

Из реального → защищен.
нужно снять с принудит. обслуживания
если нет → битая память
адреса с 1 будут недоступны

команды открыть/закрыть A20

если в реал. режиме откроем A20
→ станет доступно еще 64кб
High memory area

в защищ. доступно 4ГБ

Рудаков/Финогенов

и никакого критиката
исп. для обращ. к таблиц. GDT
начальный адрес сегмента
+ команда
поддержив. аппаратно

Два кода открыть / закрыть

using this

```
mov al, 0D1h; ком. управл
out 64h, al
mov al, 0DFh; код/команда на откр.
out 60h, al
```

```
in al, 0x92
or al, 2
out 0x92, al
```

Начиная с IBM PS/2 имеется быстрый вариант открытия линии A20

исключает опрос

но не надежно

поддерживается не всеми ипатф.

убедится заранее не возможно!

Закрыть линию A20

```
mov al, 0D1h;
out 64h, al
mov al, 0DDh; закрыть
out 60h, al
```

Семинар 4

Linux - UNIX подобная ОС. Реализ. все парадигмы UNIX

основная абстракция - процесс

(часть времени → режим пользов. "задача", выполняет собственный код

В режиме ядра процесса выполн. повторн. вход. процедуры (чистый код) использовать несколько процессов может в разных точках (не модифиц. сами себя) → режим ядра выполн. привилег. код опер. С

из этих процедур вынесены все данные (переменных быть не может)

Существ. системные таблицы - не находятся в коде структуры раздел. процедурами ядра

Много потомков ядро может разделять структуры | ядро и функции плохо описаны

В UNIX любой процесс создает сист. вызовом fork() clone()

при этом создается новый процесс - процесс потомок

который находится в отношении (потомок - предку) (не математическое)

поддерживается соотв. ук. в системе

предок получает указатель на потомка

потомок - указатель на предка

Процесс описывается структурой

```
struct proc // UNIX   <- см. методу
{
}

struct task_struct // Linux   <- найти в интернете
{                               (искать)
}
```

Дискрипторы процесса

поля - указатели на другие структуры

это необход. чтобы система могла управлять процессом и выделять ресурсы

Потомок ← копия процесса предка в том смысле что
- наследует код предка,
- дескрипторы открытых файлов, ~~сто~~
- сигнальную маску окружения

В старых системах
код предка копировался в адресное простр. потомка
← потому создавалось собственное адресное пространство
(создать соотв. страницы) ← неэффект.
↓
В современном — таблицы страниц

В системе могут одновременно существовать одинаковые копии

⇒ В совр. системах сущ. два способа оптимизации задачи создания нового процесса
(оптимизация fork())

1. System five
копирование при записи (copy on write)
когда вызов. fork() и созд. потомок для него создаются собств. карты (таблицы страниц) ссылаются на страницы адресного пр. предка
Для стр. данных и стека потока
права — ONLY READ

если пытаются изм. страницу ← возникает искл. по правам доступа

обр. это искл. система обнаур. флаг [Copy on write] и создает копию в адресном простр. того процесса который пытался ее изменить
(дескриптор должен добавл. в табл. страниц)

⇒ созданы только копии изменных страниц.
→ решено проблему кол-го использов. страниц.
управл. памятью стр. по запросу
Изм. права и флаг (см. выше)
сущ. до тех пор пока процесс потомок не вызовет
- exit()
- exec()

exit() – завершение процесса

В рез. fork() – создается иерархия процессов
поддерж. указат. (дерево)

сущ. двусвязные списки блок. процессов

В системе таблицы используют редко
(массив структур)
↳ дополнительные накладн. расходы
(фиксать допто)
– вставка
– удаление

Используются двусвязные списки

ускорить поиск

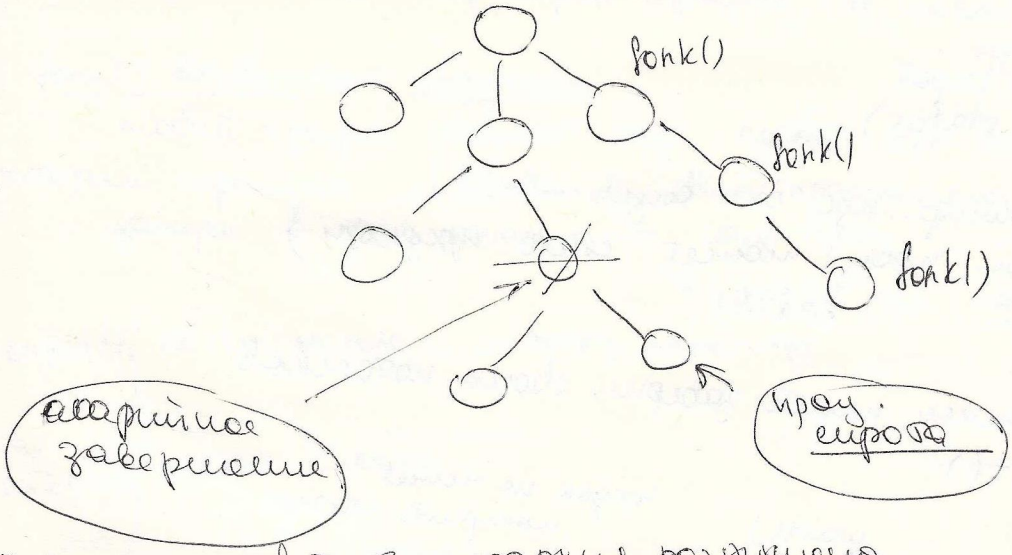
Процессы в списке наход. в соотв. с приоритетами
в системе всегда есть процесс с идентиф. – „0"
идентиф – „1"

Все идентифик. в ИПУХ натур. числа

„0" – процесс запустив. сист.
„1" – процесс открыв. терминал

Sharing Time

предок всех ~~пото~~ проц. запущ. на данном терминале



потомки живут ≤ иерархия разрушена
при заверш. процесса → проверка потомков
если потомок не завершился
→ усыновление

Потомок → ук. на нового предка
предок → ук на нового потомка

fork() – "вилка" (для контроль типов)

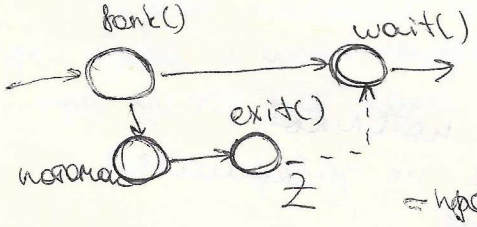
```
#include <stdio.h>
#include <unistd.h>
int main (void)
{
  int childpid;
  if ((childpid = fork()) == -1) // по стан. UNIX fork() = -1
   { pessan ("can't fork() ");
     exit (1);
   }
  else if (childpid = fork()) == 0)
   {
     printf ("execl);
     return 0;      // возврат управл
   }
  else      wait(&status);
   { printf ("parent;
             childpid = %d,        // parent
             pid = %d \n"            получает
             childpid, getpid()      идент. потомка
             }
     return 0;
   }
```

Чтобы не было сирот
← wait(&status)

Система не контр. где был вызов wait()
(любой проц. может стать предком)
–//– exit()

Логика
предок должен ждать заверш. своих потомков.

Проблема wait()



предок не может получить статус
→ бесконечно

Решение
состояние „зомби"
→ процесс отдает все ресурсы
кроме строки в табл. процессов

ресурс → дескриптер

В unix fork()

- в этом случае предок предоставь потомку свои

предок - блокируется до exit() exec()

Системный вызов exec() - метода запустить

- создает так наз. низкоур. процесс

(проц. не созд., но exec() для программы создает адресное простр. (табл. страницу)) ← только у проц.

Проверка:

- прав доступа child
- сущ. данный файл (путь)
- явл. исполняемым файлом

файлы: исход. объект. испол.

. убрать адр. простр. после fork()

⇒ в дескрипт. процесса указатель на адр. простр. проц. поменять, указать новый адрес

⇒ загрузить адрес точки входа в IP | EIP | RIP

⇒ загрузить адрес начала табл. стр. СВЗ | счетчик команд

Сист. вызов exec() переводит процесс на новое адресное пространство

fork() бомба ≈ подумать

любой процесс может созд. любое кол-во процессов

потомок переходит на выполн. новой программы

запуск в два этапа → созд. проц.

→ переход на новое адр. простр.

execl() execlp, execle ← форм. параметры

execv() execvp, execve

execl("/bin/ls", "ls", "-l", 0);

execl (char *name, char arg[0], ..., char arg[n], 0);

void *shmat(int shmid, const void *shmaddr,
int shmflg)

0 = SHM_RND

Проблемы параллельных процессов

Средства взаимодействия процессов IPC
(меж. процессное взаимодействие)

System V(5)

using:
- сигналы
- семафоры
- прогр. каналы имен/неименов.
- сегменты разделяемой памяти
- очереди сообщений

В Unix BSD
- сокеть

Семафоры (не имеют хозяина)
может освободить любой процесс
который знает его идентифр.

- передача информации

позволило устр. активное ож. на процессе
но потребовало системных вызовов

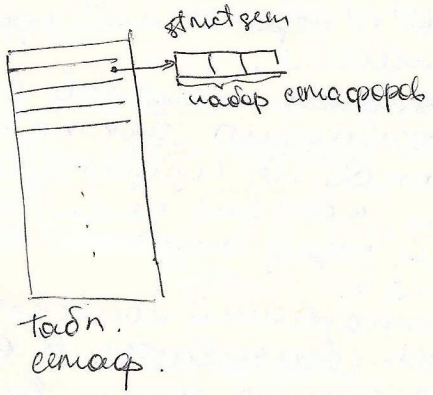
Если много семафоров – сложно следить.

Использов. набора семафоров // Unix / Windows поддержив.

↓ массивы семафоров

В ядре системы имеется таблица семафоров
<sys/sem.h>

struct semids



набор:

- имя (уник. число)
  присв. процессом который создал набор.
  др. процесс. по имени могут открыть набор
  и получить доступ

Наборы
одной неделимой операц.
можно изменить все
или часть набора
семафоров

- UID создателя набора
  и его группы

  процесс UID которого совпадает с UID создат.
  может удалять набор и изменять параметры

- права доступа
  read | write | execute

- кол-во семафоров в наборе

- время изм. одного или нескольких значений
  последним процессом
  (контроль послед. действий)
  до | после

- время послед. измен. управл. процессом
  параметров

- указатель на набор или масс. в семафоров.
  (0 . . . .)

  о каждом семафоре набора имеются след. данные

  - значение семафора
  - идентификатор процесса который операц. семаф.
    посл. раз
  - число процессов заблок. в данн. момент врем.
    на семафоре

На семафорах сист. вызовы

```
semget(),
semctl()
semOp()
```

формальные параметры
фактич. параметр.

semget();

int semget(key_t key, int num_sem, int flg) // вернет файл. дескрипт

файл дескрипт. — кол-во семафор. — флаги

int semctl(int semfd, int num, int cmd, union semun arg);

int semop(int semfd, struct sembuf *opsptr, int len);

Все провер. на „-1" / если fault = -1

На семор. Unix определено три операции:

```
struct sembuf
{
  ushort sem_num;
  short  sem_op;   // операция на семафор.
  short  sem_fl;   // флаги
};
```

1) sem_op > 0
   инкремент (освобожд. семафора)
   ≈ VS

2) sem_op = 0
   процесс вынужден.
   переводится в состояние
   ожидания до освобожд. ресурса.

3) sem_op < 0
   декремент (захват семафора)
   ≈ PS

Флаги:

IPC_NOWAIT
   информирует ядро о невозможности
   процесса переходить в „ожидание"

объясняется
   желан. избеж. блокир. всех процессов
   которые находятся в очереди к семаф.

можно перех. → процесс не станет ожид. освоб. семаф.

В случ. если процесс захват семаф - аварийно / kill

⇒ блок. на время

SEM_UNDO
- указывает ядру, что необходимо отслеживать изменения значен. семафора в результат. sem_op()

для того чтобы при завершен. процесса ядро могло перевед. сдел. процесс. изменения
⇒ не будет блокир.
т.к. ядро отменит изменения

Пример:

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
struct sembuf sbuf[2]={{0,-1,SEM_UNDO|IPC_NOWAIT},{1,0,1}};
int main()
{
  int perms=S_IRWXU|S_IRWXG|S_IRWX_O;
  int fd=semget(100,2,IPC_CREATE|perms);
  if (fd==-1)
    {perror("semop"); exit(1);}
  if (semop(fd,sbuf,2)==-1) perror("semop");
  return 0;
}
```

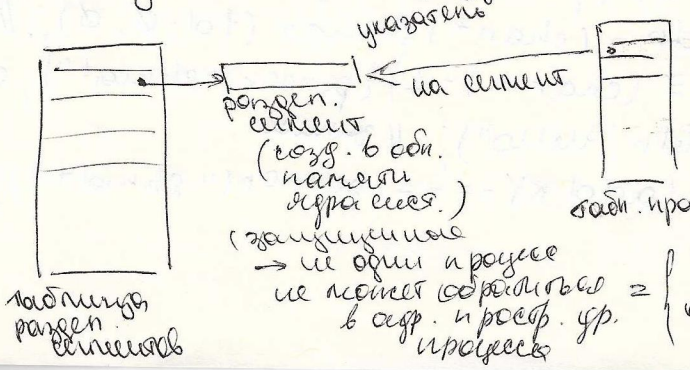
массив структур
захват
// попытка права доступа
// возвр. дескрип. дескрип.
идент.
кол-во сем.
// не удалось создать набор
над набор. выполн. опер.

// семафор был захвачен
→ процесс завершился (без освобождения)

Но система отменит
т.к. установлены флаги

Сегменты разделяемой памяти
коммуникация → передача сообщений информации <sys/shm.h>

В ядре имеется таблица сегментов разделяемой памяти
struct shmid_ds



(защищенные → не один процесс не может обратиться в адр. простр. др. процесса = { взаимод. через адр. простр. ядра }

Разделяемые сегменты
⇒ не выполн. лишнее копиров. данных
а подключ. к адр. простр.
(ссылка на раздел. сегмент)

Системные вызовы:

shmget()
shmctl()
shmat() // attach подкл.
shmdt() // detach отключить

После созд. разд. сегм.
любой процесс может присоед. его к
своему вирт. адр. простр.
using. shmat()
и работать как с сегм. собств. адр. простр.
по заверш. проц. разд. сегм. сохран
т.е. разд. сегм не удал. далее после заверш.
процессов котор. их созд.

У каждого РП есть владелец
и удаля область из ядра или корр. упр. парам.
могут процессы — приличир. права,
создатель или назн. владельца

Пример.

```
#библиотеки
int main()
{ int perms = S_IRWXU | S_IRWXG | S_IRWXO;
  int fd = shmget(100, 1024, IPC_CREATE | perms); // полные права для всех
                  // размер
  if (fd == -1) {perror("shmget"); exit(1); }
  char *addr = (char*) shmat (fd, 0, 0); // подкл. к своему адр. простр.
  if (addr == (char *) -1) {perror("shmat"); exit(1); }
  strcpy(addr, "Hello"); // запись
  if (shmdt(addr) == -1) perror("shmdt"); // отсоединяется от сегмента
  return 0;
}
```

В примере

созд. разд. сегм.

↓ Ø → первый возможн. адрес

void* shmat (int shmid, const void * shmaddr,
int shmflg);

↑ ссылку не возвращ.

В сист. на PC опред. сист. ограничения:

сист. огран.	значения
SHMMNI	max возм. кол-во PC которые могут сущ. в сист. одновр.
SHMMIN	min возм. размер PC в байтах
SHMMAX	max возм. разм. PC в байтах

если больше max ⇒ процесс блокируется пока не освободит какой-либо

если больше MAX — ошибка

Семинар

Сигналы

Механизм сигн. позв. проц. реаг. на события которые могут произ. внутри процес или вне его

Как правило

получение сигнала - завери раб.

Реакция проц. на сигн. завис. от того как сам сигн. опред. свое повед. в случ. приема какого-то опр. сигнала.

Процесс может реагировать ст. обр. т.е. в соотв. сущ. в сист. обр. сигнала

может игнор. или возраст на выпол. свой обр. сигн.

Начиная с класс. Unix

- числовой идентиф.
- мнемоническую форму записи идент.

которые в <signal.h>

(мнемоника - запоминаемая) удобная для запоминания

20 - сигналов (определено в класс. Unix)

```
#define NSIG 20
—//— SIGHUP 1 // разрыв связи с терминалом
—//— SIGINT 2 // завершение прогр. ^C
—//— SIGQUIT 3 // ^\
     SIGKILL 9 //
     SIGSEGV 11 // нарушение сегм. выход за пред. сегм.
     SIGPIPE 13 // запись в канал есть
                   чтение нет
                (не допуст. опер. с ним)
```

SIGALARM 14 // прерыв. от таймера
SIGTERM 15 // kill в ком. стр.
SIGUSER1 16
SIGUSER2 17
SIGCHLD 18 // получ. предок при зав. проц. потомка

Макросы

#define SIG_DFL(int(*)()) 0
SIG_IGN(int (*)()); 1

Средством посылки и приема в ОС unix два сист. вызова

signal();
kill();

~~Dwintkill~~

int kill(int pid, int sig);

// вызов
kill(pid, sig);
~~signal~~ sig будет. послан. сигн. с ид. pid и всем проц. родст.

Например.

pid <= 1
- sig. будет послан группе пр.

pid = 0
- // всем проц. с идент. группы совп. с ид. группы проц. который вызвал kill()

группы процессов

Примеры:

kill (37, SIGKILL);

без условия завершит.

kill (getpid(), SIGALARM);

сам процесс получит сигнал побудки

сист. вызов signal

не явл. стан. в POSIX 1

но определен в

⇒ имеется во всех UNIX и UNIX подоб. сист.

Т.к. он не портабелен

POSIX 1 (1988) FIPS – federal information processing standard (федер. стандарт)

portable Operating system Interface

(мост. переносим. ОС)

POSIX2 – включ. POSIX1

разр. нат. техник ЕИА

с гос. организ.

Группа европей ком.

X\OPEN – основан на

POSIX 1

POSIX 2

X PG3

X PG4 + доп. констр. в рамках X\OPEN

Т.к.

не реком. using в перенос. ПО

т.к. его поведен. System V и р. версиях

отлич. от пов. в base

signal()

возвр. ук. на предыд обр. для данного сигнала

можно using для данного сигнала возвращ.

```
#include <signal.h>
int main()
{ void (*old_handler)(int)=
        = signal(SIGINT, SIG_IGN);
    /* действие */
  signal(SIGINT, old_handler);
  return 0;
}
```

// см. метода (Unix2)

т.к. signal - не входит в POSIX, есть который вход.

есть sigaction (int sig_num, struct sigaction *action, struct sigaction *old_action);

<signal.h>

возм. опред. собств. реакц. на сигнал

В POSIX

sigsetjmp() // отл. одну или нес. поз. в прогр

siglongjmp() // осущ. переход на одну из вед позиций

PIP — „труба"

исп. созд. неименованные прогр. каналы

заст. имена

В. отлич. от разд. памяти

прогр. каналы под. файл. подсистемами

у пр. канал. есть дискр. файла

неименов.

— имя

— inode

сост.

— inode

— дескрипт.

симплекс связь → одностороннее

созд. вызв. pip()

кот. возвр. файл. дескрипт.

т.к. нет имени => то использ. только процессы родств. потом. проц. потомки после fork(), наслед. дескр. открытых файлов

поточная модель передачи данных

для дуплексной
нужно две трубы

прогр. каналы имеют встр. средства взаимодейст.
т.е. нельзя писать — если читают
нельзя читать — если пишут.
fd[2]
Именов. прогр. каналы созд. ком. mknod(1)
mknod <pipe> p
можно вызывать из программы
mknod (<имя>, IFIFO|ACCESS, ∅)
↑ фактич. пар.

Для чего?
для взаимодействия
могут созд. только в адресн. простр. ядра сист.

Прогр. канал или pip не превыш. ноль байт
буферизуется на трех уровнях → 1 страница
на I — в сист. памяти или
область данных ядра системы

II — диск

III — во времени

При переполнении сист. памяти
прогр. каналы (буфера) имеют наиб. время
сущ. перезапис. во вторичную память
эта процедура using стд. функц. упр. или работы с
файлом. если проц. пыт. записать больше (4096)
то труба буфер. во времени, приостановл. проц.

до тех пор пока все данные не будут прочитаны

Огран. разм. канала

→ стремление повысить эфф. обмена

т.к. при этом не испол. обращ. к внешней памяти (медленно)

Прогр. канал потоковая передача данных (при синхрон. перест. сущ)

встр. средства взаимного исключ.

разд. память подкл. к адресному простр.

Разд. сегм. не имеют средст взаимно искл.

→ using семафор (ответственность на программисте)

Очереди сообщений

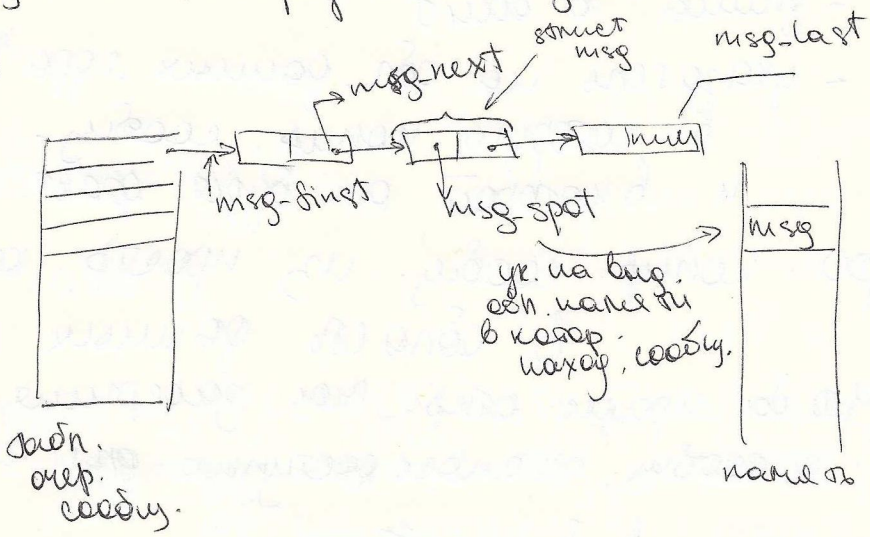
- средство взаим. процессов

msg

В ядре сист. сущ. табл. очередей сообщений

<sys/msg.h>

struct msgid_ds



сист. вызов.

msgget()

msgctl()

msgsnd()

msgrcv()

```
struct msgbuf
{
    long mtype; // тип сообщ.
    char mtext[MSGMAX]
}
```

после того как создана

msgsnd – отпр.

msgrcv – читать

Особенности Важные

когда процесс передает сообщ. в очередь
ядро созд. для него новую запись
и помещ. её в конец связного списка записей
соотв. сообщ. указ. очередей
в каждой записи
- тип сообщ.
- длина в байт
- указатель на обл. данных ядра системы
  в которую копир. сообщ.
  и в которой он будет факт. нах-ся.

ядро копир. сообщ. из простр. отправ
в область данных ядра
что бы процесс отпр. мог завершить.
а сообщ. осталось доступно для чтения др. проц.

Когда проц. выбир. сообщ. из очер
ядро копир. сообщ. в адр. простр. этого процесса
=> сообщ. удаляется (перестает существовать)

Проц. может выбр. сообщ. из очереди:

- взять самое старое не завис. от его типа
- взять сообщ. если его идентиф. совп. с идент. который указан процессе
  (если несколько → самое старое)
- процес может взять сообщ. имеп. знач. типа котор. явл. наим. из меньших или равным значению типа ук. проц.
  (если несколько – самое старое)

При использ. очер.
процессы могут не блокир. в отпр сообщ. при отправке и получ.
(вспомнить диагр. сост. блок. процесса)

Пример:

```
#ifndef MSGMAX
#define MSGMAX 1024
#endif
struct mbuf
{
  long mtype;
  char mtext[MSGMAX];
}
.mobj = {15, "Hello"}

int main()
{ int fd = msgget(100, IPC_CREATE|IPC_EXCL|0642);
  if (fd == -1) // msgsnd(fd, &mobj, strlen(mobj.mtext)
                                                  */
                IPC_NOWAIT))
    perror("message");
  return 0;
}
```

Создается новая очередь сообщ. с идент. 100
права дост.
чтение / запись 6 для влад
чтение 4 для группы
запись 2 для остальных

Если удалось созд. очер.
пишет
в очередь отпр. сообщ. „Hello" с типом 15
— указ. что данный вызов не блок.

В отлич. от разд. сегмен.
передача
выполн. копиров
при перес. коп из обл. данных → ядро
при из обл. данных → в буф получов.

Копир. не выполн.
но при этом если процессу прочитать сообщ.
имеет любк возм.
не будет блок.

⇒ очередь позв. избеж. пним. блокировок
На очереди х определено много влад.
IPC_EXCL — „прин. самост."