22.02 Александр Станиславович

igumnov @ mail.ru

Евгений Б/Т/Г/Кукин

Андрей Робачевский UNIX, 1991 перевод POSIX стандарта, unix.org

parallel.ucan.ru

История

1966-68 г переход от mainframe к массовым компьютерам

Ибх унифицировать прогр обеспечение. Ранние заточка под проц

MULTICS (AT&T, DEC)

- Исп иерарх ф.с
- В рамках одного namespace и файлы и драйвери

Но придумали слишком академично. Все распалось. Вернулись обр

AT&T Генри Ричи, Томсон

Захотели запустить Startrack на PDP-4 (12 разр маш. слова)

- Придумали драйвер
- 1 тыс строк ядра ОС

1969 г - оформили UNICS - без защиты доступа к памяти, однопольз

Переименов в UNIX

В AT&T в то время ими разр клас. прогр средств:

- awk
- sed
- grep из go to /re/ and print
- make
- C compiler

Далее яз. C влияет на перенос ядра под разн арх

PDP-8 в PDP-11 (16 разр маш слов защ памяти)

В универ. начали пытаться переносить UNIX на разн устр-ва

(Австралия - Atlas)

Экономико-полит сост

AT&T - вся связь принад ей. → налож запрет на разр прогр обесп

(продавать не могли)

Ричи и Томсон вели курс по ОС. Добились беспл распр исх текстов

UNIX 1970-80 BSDUnix Berkley Software department

В Berkley студенты писали: упр памятью, новую ФС

Выиграли у DEC конкурс на Reference implementation TCP/IP

UNIX - при каждой документ иду версию

В 1980 присвоили версию System III

Доб TCP/IP ⇒ System V

Все коммерч unix осн на вер System V

DJVU Антрепренер

В 1990 AT&T разделилась и исслед подразд выделили в Bell Labs

⇒Bell Labs стала владеть правами на UNIX

Sun, SGI (Silicon graphic), Apollo, IBM - рабочие станции

Появ нужда в единой ОС и переносить нужно быстро

Sun как раз основал студент, разр. BSDUnix

У каждой фирмы свои разработки

Sun - NFS, IBM - Виртуальщ, SGI - OpenGL

ОС: HPux, Irix, AIX - были разные. Нет эф. в работе

UNIX потерял конкур. преим. - Выпуск Windows NT под x386, DEC, PowerPC

- обесп совместимость всех версий

проц.

Bell Labs наехали на Berkley - закройте исх тексты (1985-1990)

→ сом.: универ убирает приставку unix, убир запатент код и пишут свой

→ BSDLite (не комп под любой комп)

1984 - идея мирового стандарта: осн неком сообщ., группа экспертов выдели самое ценное. →1990 POSIX (portable os interface X (Unix))

POSIX зафикс UNIX на уровне 1980 года. Тогда могли комп под Win32/POSIX

Но POSIX слишком урез набор команд

1990 Линус Торвальдс - реализ POSIX с нуля

Ричард Столман - обиделся на закр код и орг GNU - дубл набор unix утилит

POSIX

описан в POSIX ①

stdlibc (fopen, fread, printf, malloc) ② - тоже опис в POSIX отобр в вид ядра

③ Набор утилит: awk, cpio(apx), sed, grep, mkdir

④ Shell (sh)

GNU: набор утилит + компилятор C

Linux оказался впереди

На осн BSD Lite → FreeBSD - оптим под перс. ком (с Intel 386)

↘ OpenBSD - верифр ОС. с пом экспертов

↓ NetBSD - максимально переносимая ОС

Linux в нач отставал от BSD

Отлич в лицензии. BSD лицензия: можно доб свое и не показывать

GNU: любой продукт нужно всем показать

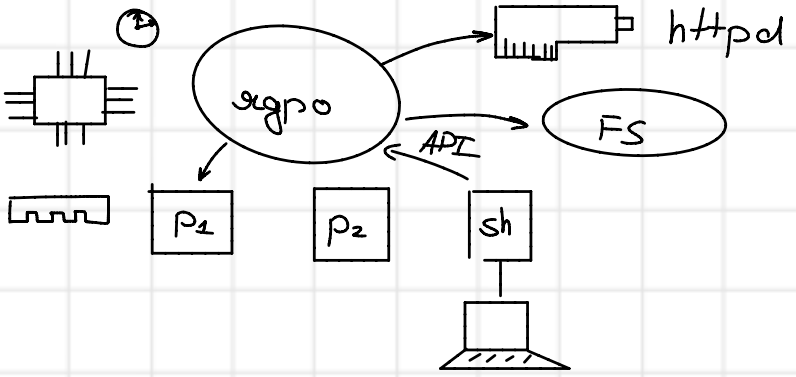
Linux пишется профессионалами

На курсе берем только POSIX и интерфейс ядра

Linux cross reference (tobaxtrem)

(1.03) Unix глазами польз

Проц-не связ с польз сеансами - daemon



Ядро: 3 лог уровня

Привел режим - ядро, возм упр вирт памятью (32 разр - 4 ГБ). Возм изм табл вирт памяти. Процессы не могут вмешиваться

Ядро имеет дост к памяти любого процесса

Половина адр пр-ва исп ядром для отобр ядра в себя

Аппаратные прерывания

Многозадачность: добровольная и принудительная (на ядро не распр)

Идея микроядерной арх: в ядре только обр таймера - ост вынести [не в Linux]

Друг сторона ядра - общение с польз API

- взаим между проц
- выделение памяти по запросу
- взаим с I/O

#include <unistd.h> <fcntl.h> man open (в ядре) Но сами передать упр в ядро не можем

→ open - спец замыкка с ним регистрами - номер функ → прочр прерывание → int 22h → ост нашего проц → syscall → kernel open → верифр парам → file descriptor

Совместное исп ресурсов

uid, gid root==0

При созд польз созд группа, куда он входит

Кажд проц присвоен уник номер pid

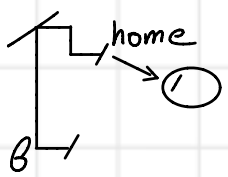
ps -a-все проц ps -e в BSD

x - без привязки к терминалу

Каждый thread имеет свой pid

'stat file.txt'

ФС в Linux - большое дерево inode



На кажд диске есть корневой каталог (inode=2)

Корневой каталог устр монтируется к главному

Права доступа $\underbrace{rwx}_{ч},\underbrace{r-x}_{g},\underbrace{r--}_{all}$ #--право отсутствует

ФС как способ предст имен /etc/passwd - стартует то, что в нем

symlink - созд txt строку, кот исп при разборе имени

/etc/passwd файл

/etc/ каталог

имя

FIFO

Socket, (ip,port) || (имя в фс) Клиент созд сокет в tmp

node
char
block - точки входа в ядро Linux. Через них вып низкоур операции

/dev/sda maj=8 #драйвер SCSI. Сам диск | номер драйвера устр-ва
minor=0 #номер устр на шине | номер устр (для драйвера)

/tmp

/dev/tty/console #текущий терминал

/etc #конфиг

/bin # исп файлы /sbin # сист файлы /lib #для /bin

/usr #польз, менедж пакетов

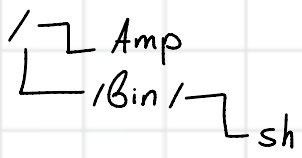
/usr/local, сами скрипты

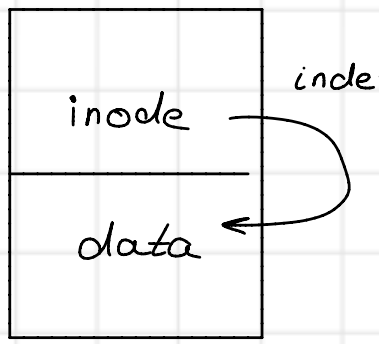
7 задач по 1 банку. Долг 1/4

(15.03) # Счетчик ссылок

Устройство каталогов

1-я Unix система





index node (stat -проси)

inode = 2 корневой каталог

/

Имя	inode
.	2
..	2
Bin	3

Для корневого

/Bin

.	3
..	2
sh	1017

Жесткие ссылки - hard link

Symlink - для монтирования устр (ссылка на любую часть дерева)

Удаление - выставления флага, что inode свободен

#include <unistd.h>

unlink ("path"); // вычерк стр из каталога и уменьш счет в inode на 1

// Если первый симв "/" то разбор с корневого

// ".Bin", "../bin" - отсчет ведется от текущего каталога (св-ва текущ проц.

// > chroot /tmp смена корневого каталога

// > cd - смена текущего каталога

// Есть счетчик сколько раз файл открыт

link ("old", "new") // Созд новое имя для файла

symlink ("old", "new") // Созд новый объект и разм текст old внутрь

Ядро нам возвр целое число

#include <errno.h>

ret = symlink ("string", "new") // Просто число

// Остальное в глоб переем

extern int errno // EACCES, ELOOP, ETOOLONG

↑ хран переем в опер памяти ⇒ пробл с многопот ⇒ храним в local потока

perror ("string") // Выводит на stderror описание ошибки

char *str = serror() // указание на строку с ошибкой

Права доступа к каталогу

r - читать имена w - менять имена

x - доступ к inode

/bin ← нет прав на чтение inode

чwx ч--ч-- ls /bin - должна сработать

ls -l - уже не сработает т.к читаем данные inode

Если нет правим на чтение чwx-wx-wx. Любой может созд файлы, но не читать, что есть в каталоге

Обычно права на чтение и исп стоят вместе

В inode пишет либо Владелец файла, либо администратор

link ← везде исп текст имена на файл

unlink Система отдает inode и раскручивает ссылки

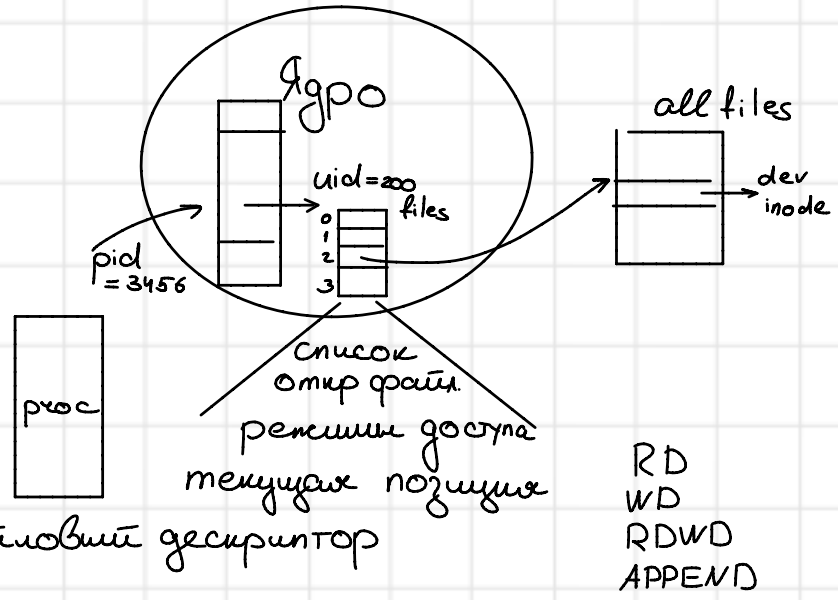
symlink

stat readlink - Возвр текст внутри симв ссылки

open lstat - на вход текст строка - Выдает инф о симв ссылке

chmod lchoun

choun Читать симв ссылку можно всегда



int fd ← файловый дескриптор

Первые 3 fd имеют особ знач в shell

0 - stdin

1 - stdout

2 - stderr

int fd = open ("path", flag, mode) ← доб маска inode (не дает писать никому другому)

ret = creat ("path", mode)

ret = read (fd, buf, size)

ret = close (fd)

2203 303/304 3 этаж след пара

Стандартных вызов link (old, new) нельзя сделать для каталога

find, tar - надеятся на граф без циклов

symlink - разрешены

err = close (fd)

Флаги в правах доступа

sticky - если прилож работает в реальном врем - ставим sticky, чтобы не выгружалось на диск

Еще одно знач - sticky bit

/tmp w x-r-x-rw ≡ удалять файлы можно только если у польз есть права на изм самих файлов, не смотря на изм директории

sgid - /etc/shadow rw------- Только root читает

suid Права на доступ к секр файлу не секр польз

↳ такие же флаги как r, w, x

/bin/passwd - прог, кот исп passwd имеет достип к паролям

Только для владельцев Отлаживать при этом запрещается

vipw - ручное редакт паролей - вырезали строку польз → запускали vi с новыми правами, а там можно !cmd

Так же исп sgid и suid для блок обш доступа к файлу

*В Linux есть добр блокировка, хотим принудительную (rwx r-s r-w) (set)

смена гр есть, а самого права на исполнения нет

fd = open("fname", flags, mode)

#include <fontl.h>

O_RDONLY В табл откр файлов помещ нужная запись (у конц процесса)

O_WRONLY

O_RDWR (сумма 2-ух предыдущих)

O_APPEND #допустим для логов, чтобы не перетирать друг друга

O_CREAT (созд нового файла, если он не существует)

mode & ! umask Установка umask ␣007 - теперь всегда в конце ---

O_EXCL (гарантировано созд новый файл, можно битово сложить с пред. флагом)

O_TRUNC - укорачивание файла.

O_SYNC - вызовы read и write будет возвр, когда данные будут записаны на физ носитель

n=write(fd, buf, size) - весь буфер сразу уйдет в опер память и нам уже что-то вернут, хотя на медленную дискету еще не записали

- O_NONBLOCK - проти возможная ситуация - сейчас ресурс для записи не дост.

EAGAIN - будет в Errorn - говорит попытаться еще раз

- O_NOCTTY # no control tty

/dev/console - устр упр

/dev/tty - устр ввода / вывода

ctl C

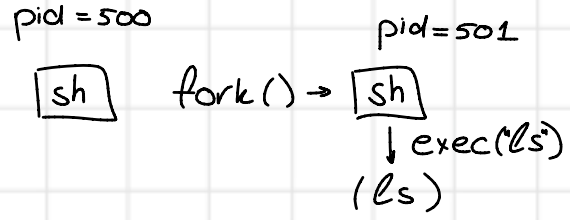
ctl z - в фон режим (SIGSTOP)

ctl D - заверш чтения

Все отн к проц, кот запущены из терминала

Но если он откр файл, то ему могут назначить упр терминал (без этого флага)

- O_CLCEXEC



Вызов open к каталогам не применим

Есть спец набор ф-ций в библ C

n=read(fd, buf, size)

write(fd, buf, size)

Возвр число байт, кот были прочитаны / записаны

либо -1 и смотрим в ERRNO - код ошибки

Если 0 - достигли конец файла

Если есть структура в разн частях памяти

writev(fd, iovec, num)

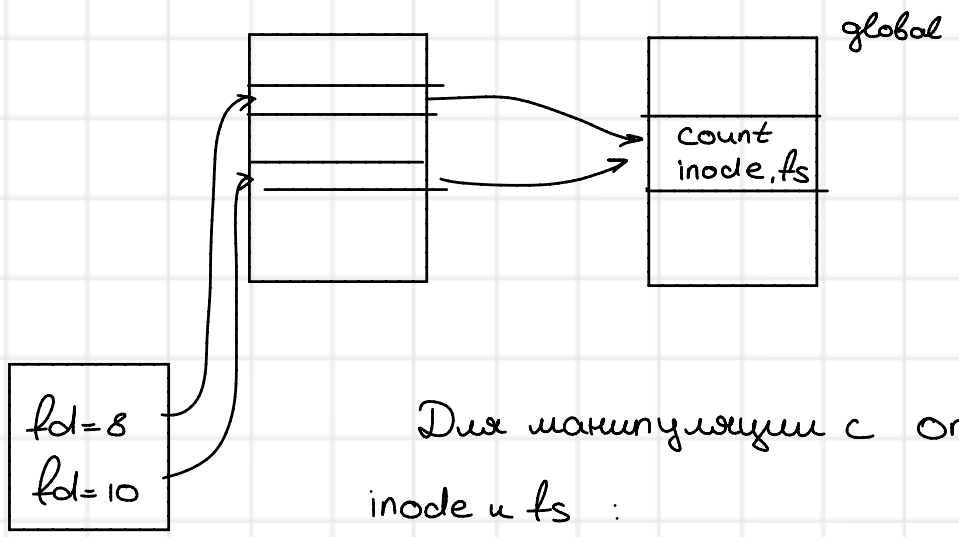
readv(fd, iovec, num)

↳ массив из структ iovec

```
struct iovec {
    iov_base
    iov_len
}
```

← делаем такую структ для кажд участка

<sis/uio.h> — тут опр iovec



Для манипуляции с откр файлами на оси только inode и fs :

newfd = dup(fd) # фц ссылаются на один файл

newfd = dup2(old, new)

cat <f1 >f2

sh --fork()--> sh exec("cat")

cat >f2

close(1);

fd = open("f2", O_WRONLY | O_CREAT, 0755)

exec("cat")

fd = open("f2", O_WRONLY | O_CREAT | O_CLOSEXEC, 0755)

newfd = dup(fd, 1)

exec("cat")

savedout = dup(1) # дуб станд ввывод, возвр обр

fd = open("logfile") # Advanced Bash Progr

dup2(fd, 1)

libfunc()

dup2(savedout, 1)

close(savedout)

FILE *fr = fopen("file", "rw")

fp = fdopen(fd, "rw")

int fd = fileno(fp)

Чтение каталогов

#include< dirent.h>

DIR *d = opendir("/")

struct * dirent = read dir(d) # d_ino, d_name[256], d_type

close dir(d)

<unistd.h> # Перемещение по файлу

offset_t pos = fseek(fd, offset, from)

SEEK_CUR
SEEK_END
SEEK_SET

Можно делать дырявые ф-ии

Блокировки ф-лов

BSD → flock(fd, op) POSIX lockf

LOCK_SH - много проц они могут указать

LOCK_EX - эксклюзивное польз. Проц, кот запросил - ждет в ядре, пока не освобод. все остальные

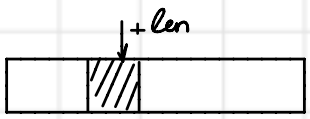
LOCK_UN

+

LOCK_NB - проверка, нет ли блокировки

lockf - более гибкий

lockf (fd, cwd, len)



ждем участок pos... pos+len-1

F_LOCK F_TEST

F_TLOCK

F_ULOCK

fcntl(fd, op, ...) - можно на ходу менять параш

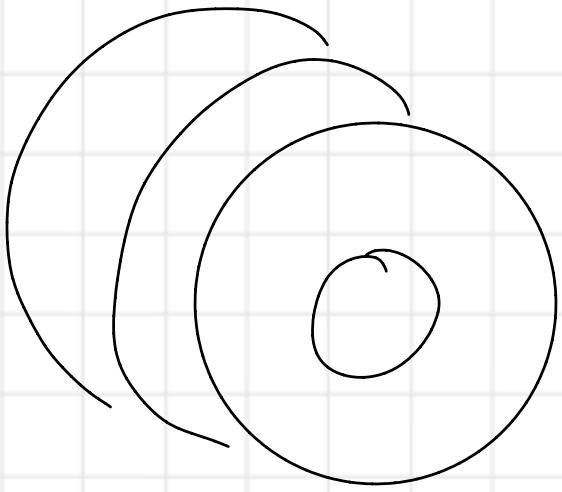
29.03 Файловые сист

fs5

ufs ext

ffs

4кб собр разм сектора диска (читаем 4кб даже для чт 1 байта)



CPU перех по адр загрузчика

BIOS пров все шины

В BIOS есть драйвер для работы с диском

Читаем MBR и передаем упр в нужное место

с смещением (0 сектор 0 дорожки)

Сейчас исп загрузчик GRUB (догов. 3 операц.)

↙ маленькая ОС

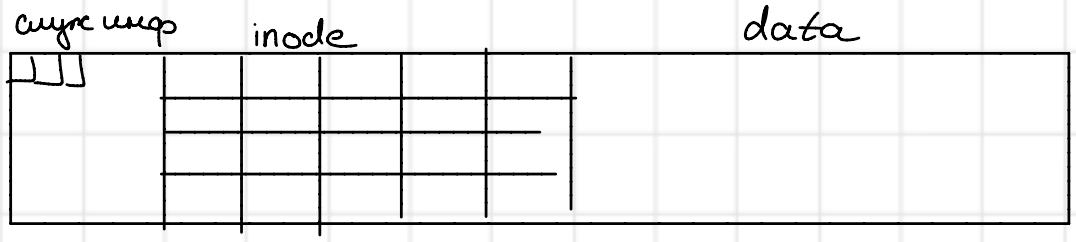
Нам ост 300 байт – он обр к BIOS и проч в опер память сам загр GRUB

⇒ нужно записать GRUB в такое место, кот не перезатрется

Сам загр уже просм все диски, ищет /boot/grub.conf. Там уже напис все про ядро и доп парам

Нам часть в GRUB - stage1

При форм диска с ext сист - ост свобвсю 0-дорожку

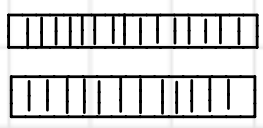


Разм inode меньше сектора (inode ≈ 64 байта)

Удобно выравн разм inode по разм сектора

Служ инф: • SB (super block) – опр. разб и размеры – геометрия

• Битовые карты



free inodes (ищем первый своб inode)

free blocks

↙ созд структуру

- type
- nlinks
- uid, gid
- флаги доступа flags
- size (в байтах) → • врем метки
  - atime – врем посл доступа
  - mtime – врем модиф
  - ctime – change time (inode)

При арх можно восст atime, mtime

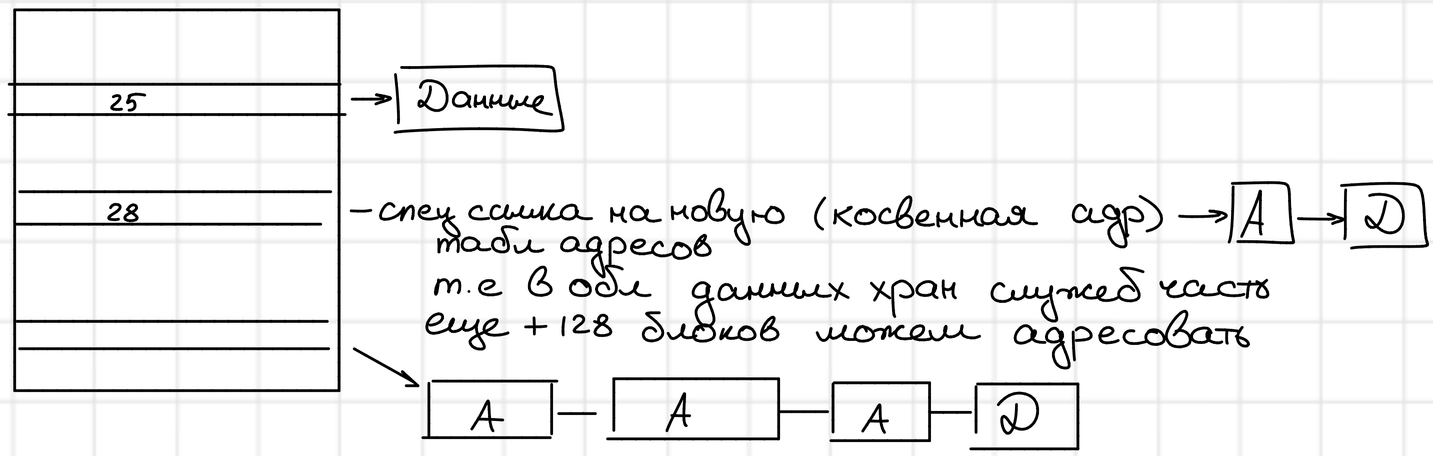
ctime может изм только ядро

*make исп mtime для пересборке

Исп утилиту touch - mtime:= now

В UNIX нет понятия времени созд файла (т.е невозможно найти все файлы созд в нужн день)

• addr [15] массив адресации



Сколько раз перейти знает драйвер

В старых UNIX не было ACL

Пробл классической фс:

- Потеря SB (он был один)

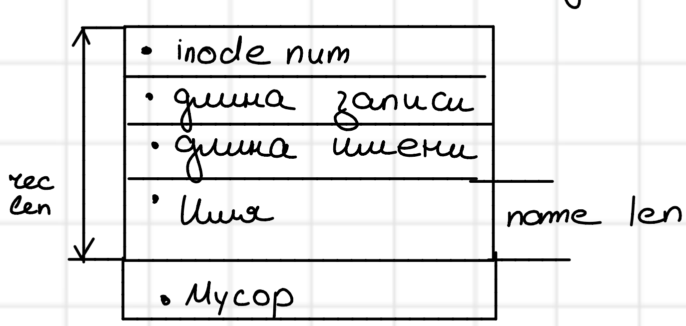
  Можно делать перебор, пока не найд. данные

  ⇒ В Berkley придумали slice – каждое из колец имеет структ ФС

  Сейчас: дисковые группы / группы секторов

- Перескоки с data ⇔ inode

*Каталог – самост структ



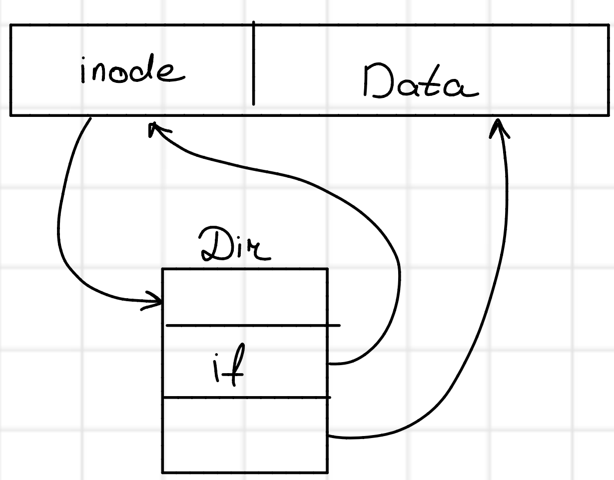
← записыв последовательно

Если удалим, то все просто передвигаем

⇒ к пред просто допис длину удал записи (как мусор)

"Проблема удаления из середины"

Созд нового файла в каталоге



1) помет inode зан. но данные не записали

2) записали данные, но inode не заняли

⇓

fsck – проверка ссылок

Ищет inode на кот никто не ссылается

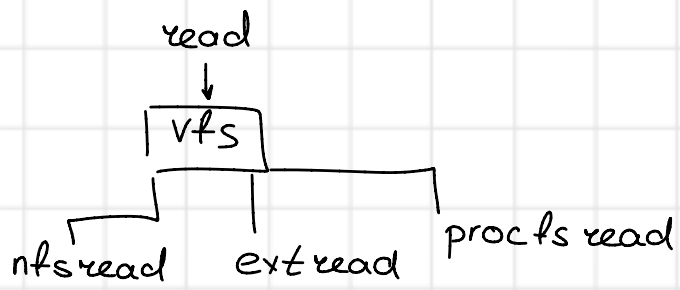
/Lost+Found

Но все перебирать долго ⇒ журналирование

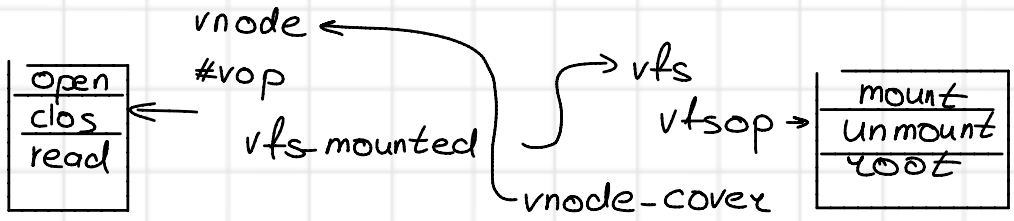
В журнал пишем атомарно что соб. делать

Виртуальная ФС vfs

fd → vnode (Все те же поля без привязки к диску)



Осн структури vnode, vfs



Двунапр список для переходов

* procfs - интерф ядра Linux

* fuse - драйвер фс. - можем писать свою ФС на чем хотим

Разбор имен /abc/def/gh

root_vnode или cur_vnode

1) Объявл блока занятым
Ни один dir не ссыл на inode, но в счетчике что-то есть

2) Два + число подкаталогов

(12.04) Пайпы. Неименованные каналы

> ls | grep 256 > 0

int fd[2];

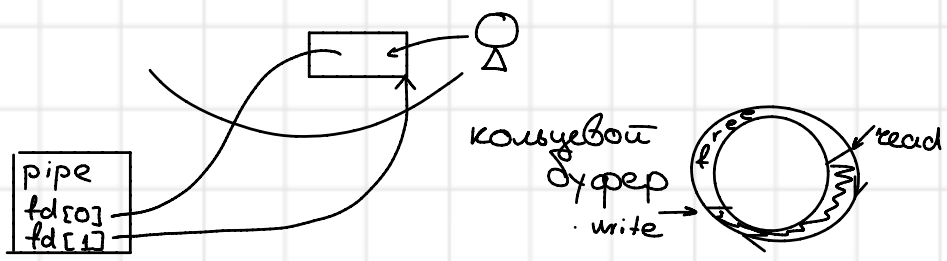
stdin == 0
stdout == 1

err = pipe(fd)

read(fd[0], buf, size)

read(fd[1], buf, size)

dup2(fd[0], 1)
close(fd[0])



n = read(fd[0], buf, 1)

1. read < buf
2. read > buf → Заказали n, получили меньше → EOF
3. read с buf пуст → блокировка

0 - признак конца файла при чтении

write (fd[1] ...)

4 n < free

5. n > free блок + неатомарн

6. n ! нет читателей SIQ PIPE

*Больш записи могут перемешаться

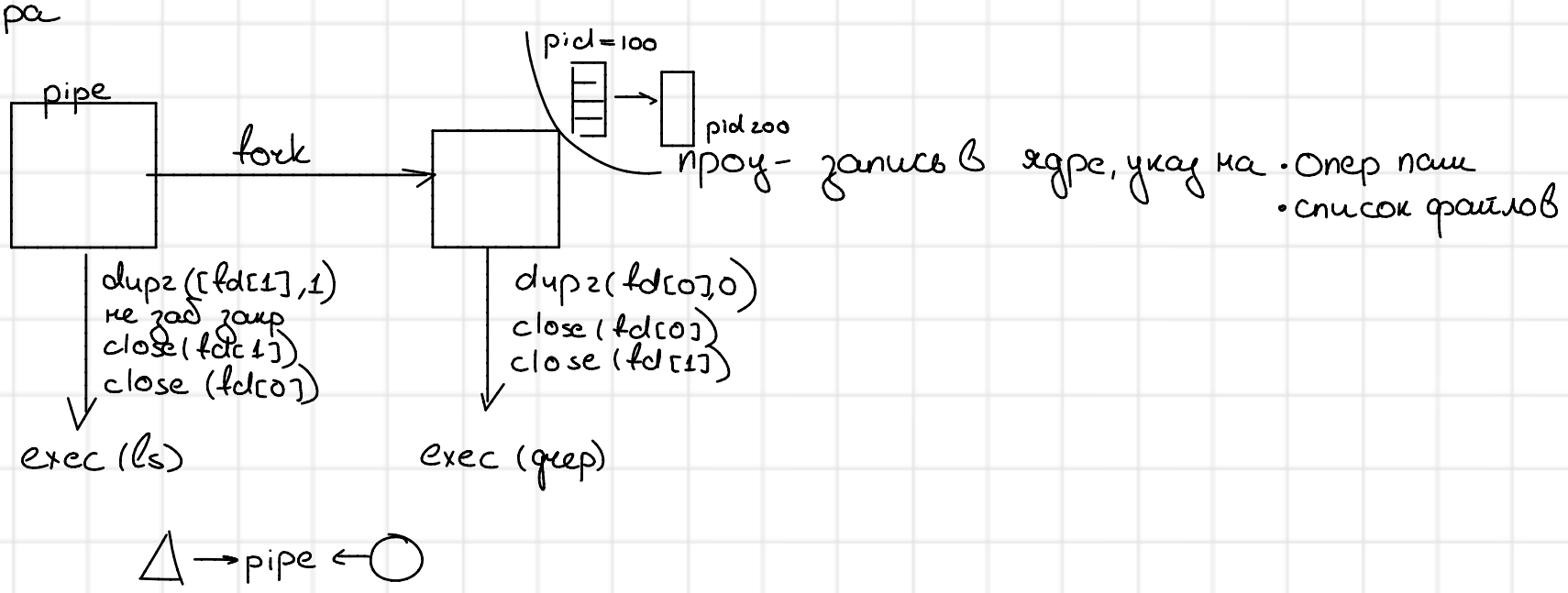
cat /dev/random | фгер 235 либо наоборот "быстро" | "медленно"

↖ медленно

На запись конца нет ⇒ присылаем сигнал

Есть 2 вида сигналов. kill - KILL pid - доб в табл сигналов

NFS - без серв ост в ядре часами. Ошибки домо не будет - не возвр из ядра



Как кешменов. канал, но с именем FIFO

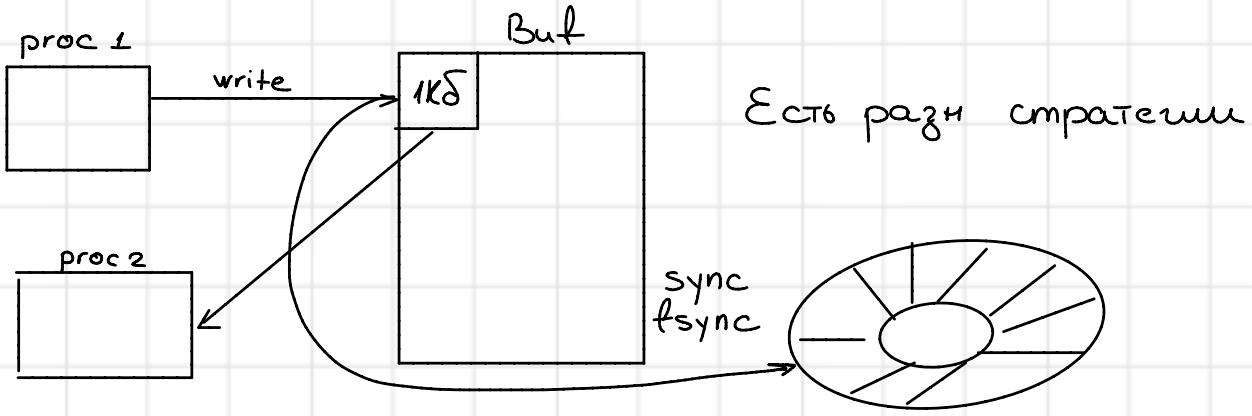
mknod name p

mknod name B 1 100
C 2 0

fd = open("name", flag, 0, O_RDONLY)

Кеширов данных из ФС

Если дост ОП, то вид буфер для промеж обмена с диском

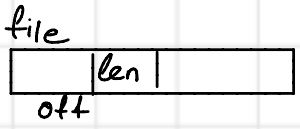


Есть разн стратегии

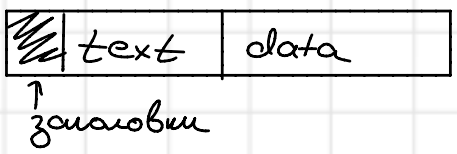
Swap – расширение опер памяти на диск

```
# include <sys/unimap>

caddr_t addr = mmap (addr, len, prot, flag, fd, offset)
```



Отобр исп кода, но не данные. Данные → опер



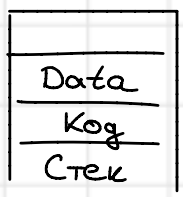
Протоколы отобр

PROT_WRITE
READ
EXEC
NONE

MAP_SHARED
MAP_PRIVATE - имен копия
MAP_ANONY ? при записи ост не заметят

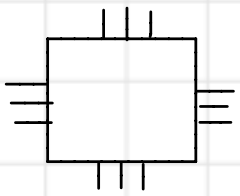
Но если сбрасываем?

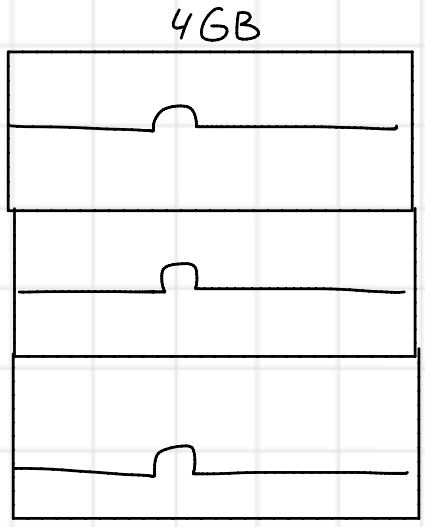
19.04 процесс



8086 – 16 – 16 мб
386 – 39 4Гб
64

64кБ





Адр памяти
$16 \times 2^{10} \times 2^{10}$

CS
ds
SS

уст 1 мб, но можно было адр до 16 мб

32 разряда

Сейчас сегментные регистры можем исп статически

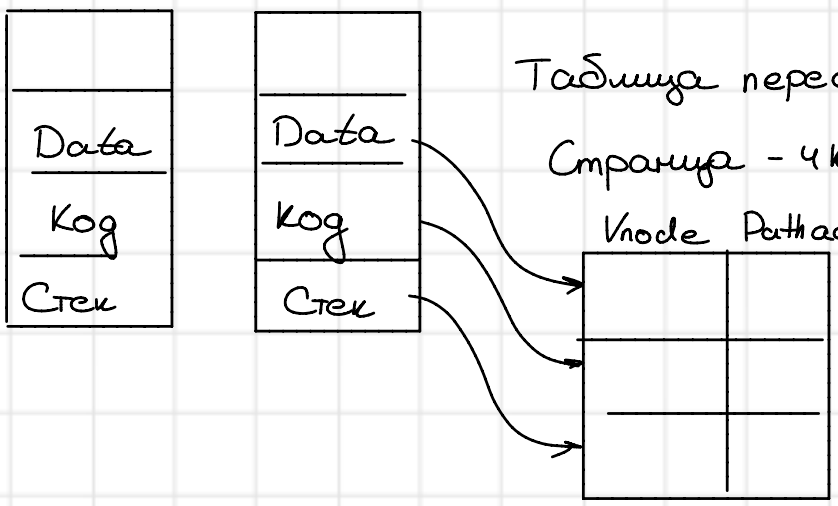


Таблица пересчета адр TLB

Страница – 4КБ. Записи: преобр 4КБ

Для кажд проц. – своя таблица

- Табл не отобр в пам табл
- Процес не может изм указатель на свою табл адр.

При смене проц:

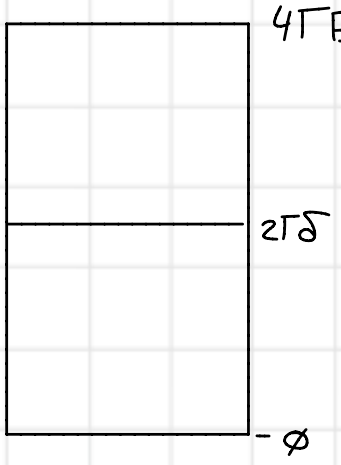
1) ip, cs, ds, cs – сохр в табл процессов
   ax, bx, cx
2) Табл отобр сохр в ядре
3) Загр новую табл
4) Передаем упр на ip

read(fd, buf, len)

- Прив - передача упр ядру
- Передача упр на опр адр

Ядро хочет дост к своим и к нашим данным

Всю память делим /2



Старшие адр перест на свой код ядра в привелег режиме

В польз режиме не видим адреса ядра

Форматы COFF, ELF (формат встраивания и связывания)

Виды памяти в прогр readelf

PROG_BITS

.init (созд потоки cin, cout, cerr)
готовит ост

```
main()
{
  return 0;
}
```

вызывает main

.fini

В момент заверш - вит буфера на диск

-exit(∅) - заверш без . fini

.text

.data - для иниц. перем int b=1

.bss - для не иниц. переменных int a;
имена и адр глоб перем

Отдельный раздел для стека

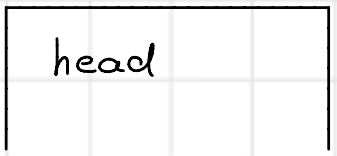
Еще дин выделяемая память - heap

malloc(size)
new type

```
main() {
  int c
  int *d=malloc(1)
  printf("%p\n", main)
}
```

Формат ELF

objdump -f /bin/ls

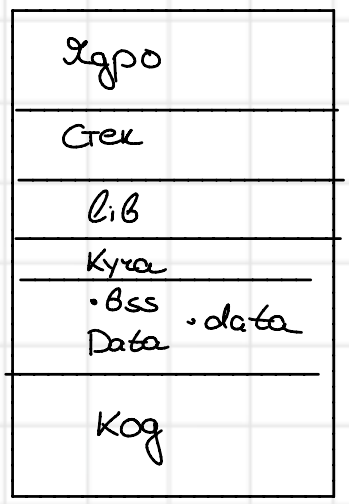


.interp

chmod -x /bin/chmod

linux. ld.so Загрузчик /usr/lib/ld-2.25.so

cat /proc/428/maps



← распр памяти после загрузки

← еще что-то в младших адресах

Для кучи нет смысла резервировать большую кучу.

Вводится спец указатель Brake. Ниже-есть отобр Вирт → физ; Выше-нет в Вирт памяти

ecx = brk (addr) # явное указ адр

addr = sbrk (offset) # переместить Вверх или Вниз

ленивый

#Можно напис malloc

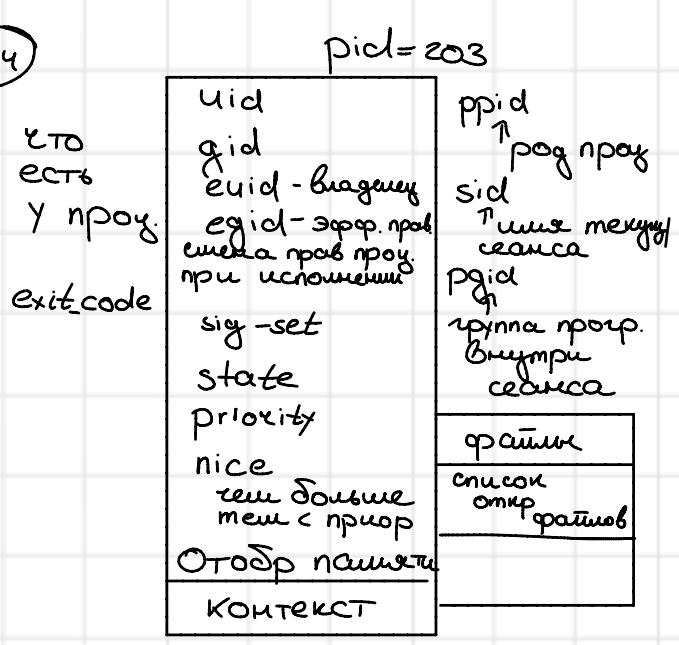
OOM_killer - кого завершить, если не хватает памяти

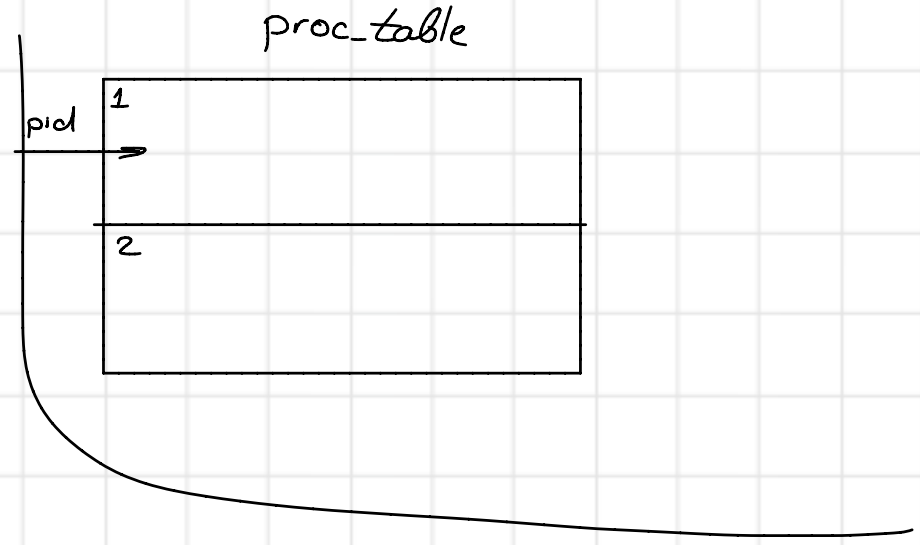
ls /proc/self oom_adj - тонкая настройка

1) /proc/self/oom_adj → 128 или планировщик служб

2) exec("prog.elf") - замена bash

26.04





Pid должен вмещаться в int32, uint32

Shell

имя → :(){ :|:& };:

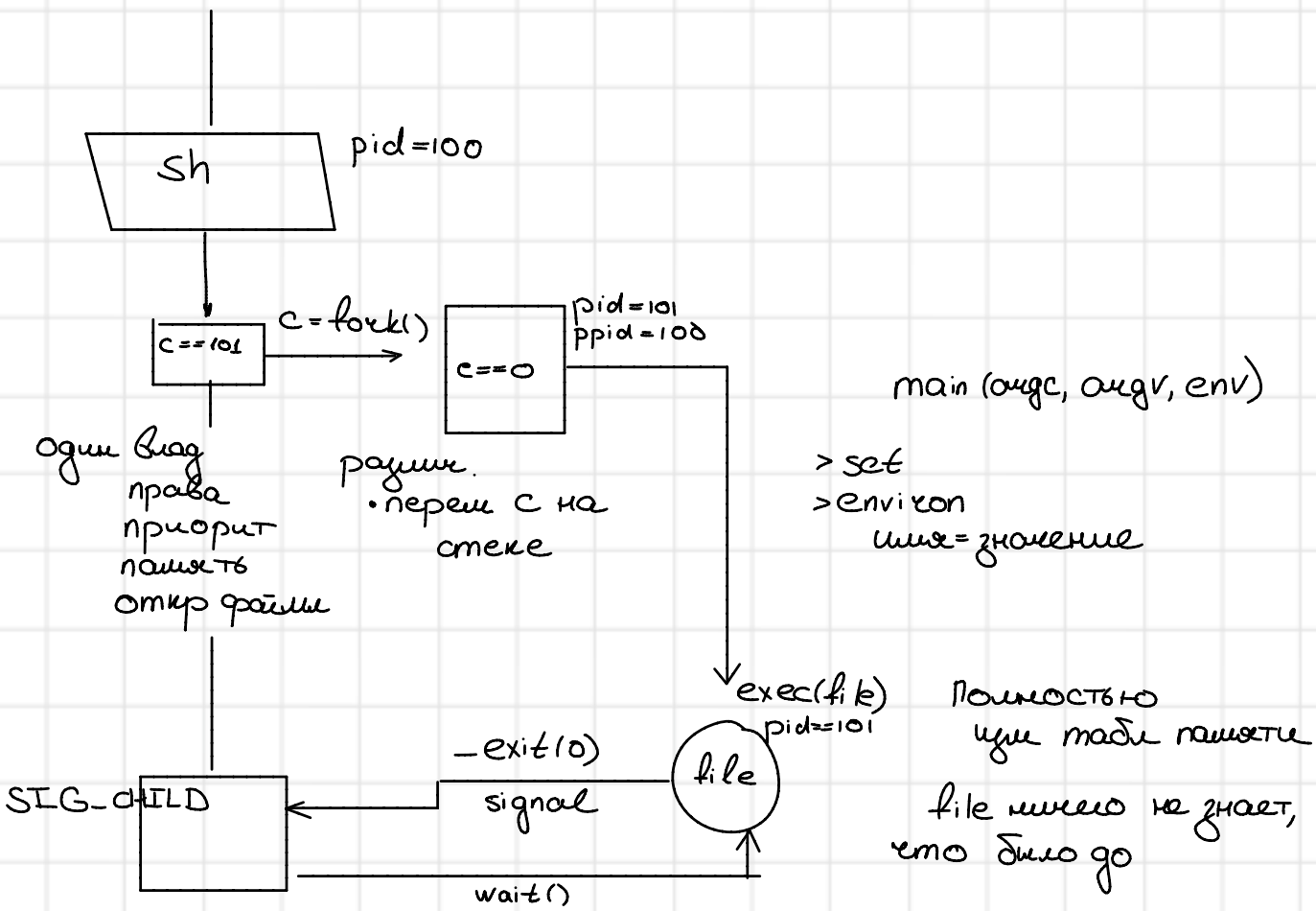
обязат ф-ции

копия ф-ции отдает другой копии

фон режим

ulimit -a

killall берет процессы (grep) из списка на опр момент времени



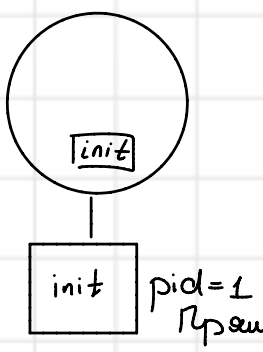
Родит проц. должен с пом команды wait почистить детей

pid=1 init А если род завер раньше?

Все осиротевшие проц. получают в родители pid-1. Он пробегает и чистит

Откуда берется pid=1? → Созд ядро в момент загрузки

fork() в ядре => clone(...)



Ядро → асинхр
↘ синх для прерив
Дальше init будет получ часть врем.

Прямо в ядре хранится
т.е скомпилир. вместе с ядром
Копируется clon'ом в обе процессов

initrd ищет в /init
/sbin/init
/etc/init

загрузчик

ядро, вирт диск, файловые разделы ← варианты расп init

init нельзя заверишть средствами ядра (защищен)

Есть уровни запуска

1: wait: /bin/bash

2,3,5: restart getty

0 reboot
1 single
2 ¬NFS
3 multiuser
4 не исп
5
6 shutdown

Переход на уровень - - вып набора скриптов

Теперь есть systemd, т.е /sbin/init - ссылка на systemd

/lib/systemd/system - зависимости

(3.05) cout<<"before" endl - опасно с фрашами

fork() Буферезированный

cout<< "after"<<"\n"

```
if (n=fork())
   parent()
else
   child()
```
← надо еще проверять на ошибку

```
int main (argc, argv)
  {
    printf("%s\n", argv[0])
  }
```
← нет return 0
т.е на вершине стека должно быть знак на вершине стека

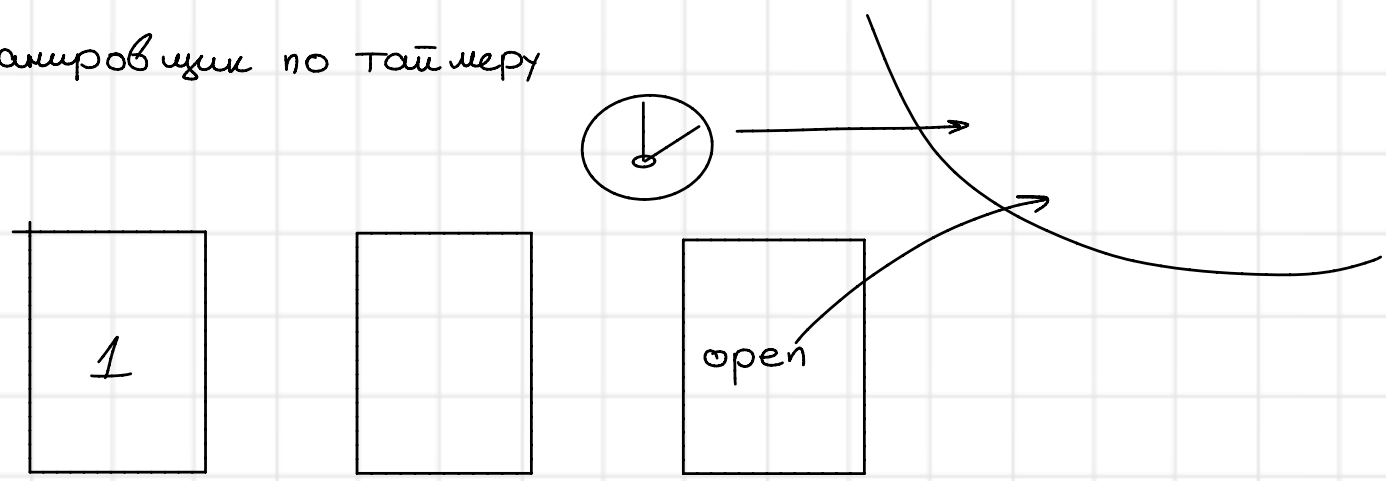
```
>mytest
>echo $?
```

execl
execve
Все про fork, wait, exec

execl("/bin/ls", "ps") Изменение своих имен

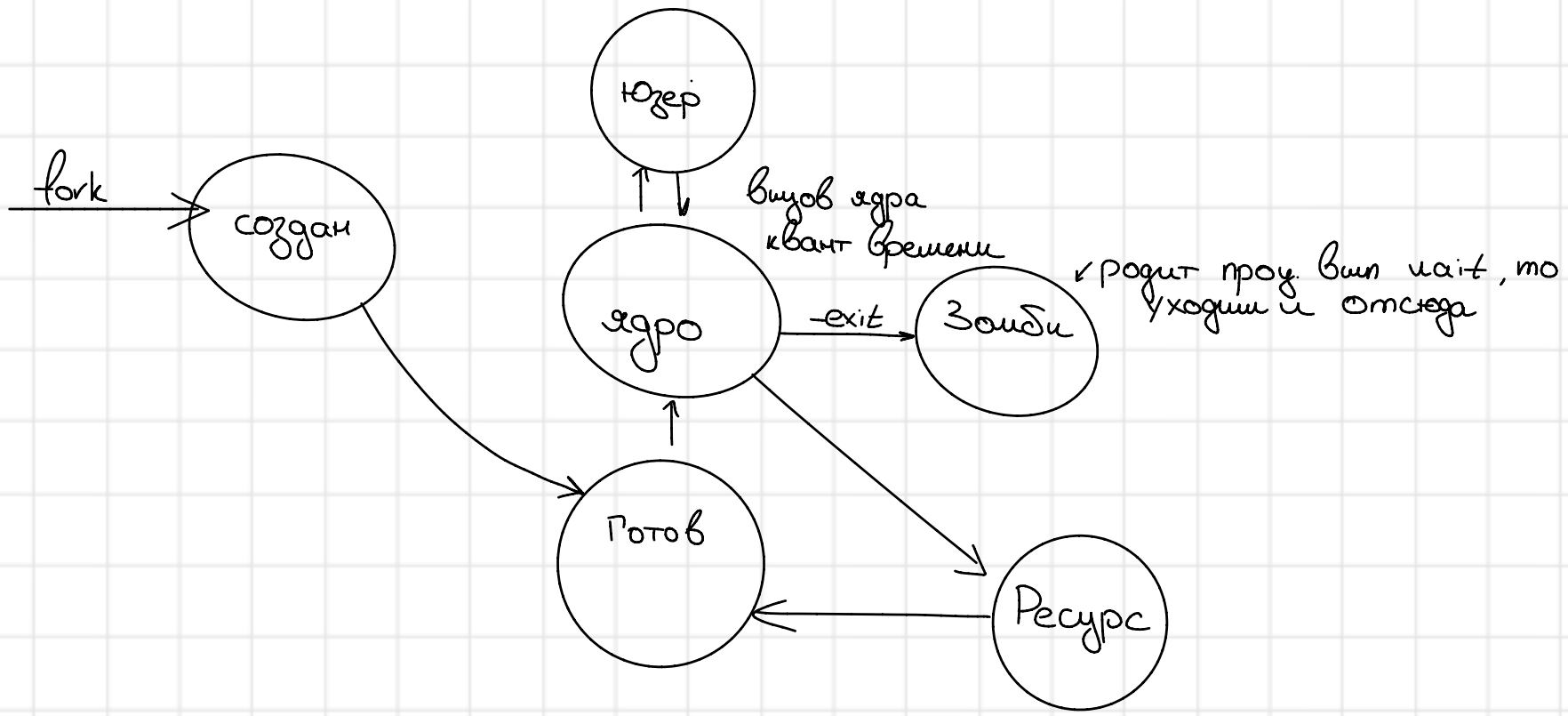
wait pid

Планировщик по таймеру



Приоритет динамический

- Жесткое реальное время
- Мягкое реальное время - ошибка, если не успел



cat & группа фоновых процессов

Пытается печатать - ей драйвер шлет stop

10.05 kill (pid, signal

kill -l #64 сигнала. Есть подозр, что это битовая маска

1) SIGHUP (повесить трубку)

nohup - для запуска в фоновом режиме

2) SIGABRT

3) kill -KILL 25 не может быть перехвачен прогр

4) SIGTERM - заверш, но м.б перехвачен.

При reboot: SIGTERM, потом уже SIGKILL

5) SIGPIPE

6) SIGSTOP - ручное упр планир проц. - перест выделять врем

SIGCONT - продолж

7) SIGTTIN - ост, если проц. яви фоновым

SIGTTOU #при попыт ввода-вывода

8) Аппаратные: "/0", память, устр

SIGIN

SIGBUS - аппаратный сбой

SIGFPE - ошибка плав точки

SIGSEGV - обр памяти

9) SIGALARM - по таймеру - вызовут SIGALARM

SIGRT - сигналы реального врем. Обр по-своему

Мы можем: умереть, уснуть

перехватить, игнорировать
↳ отдельная команда игнорифов.

	действ
0	умереть
1	уснуть
0	перехв.
1	игнор
0	

↑ флаг "0" если обработали

юзер | signal_handler

ядро

2 способа обр сиг: классический и современный

```
void shand1(int sig)
{
    signal(sig, shang1) # обр сбрасывается на стандартный
    printf("%i\n", sig)
}
```
} - должны усп до след
↑ можем попасться в malloc

```
void *old=signal(SIGINT, shand1)
```
↓ сейчас многие функц потокобезоп

Макросы: SIGIGN, SIGDFL
↑ игнор ↑ восст дефолт

Наши порожд уровни

sh
sh
main

⇒ работаем как с потоками

> 0 proc

0 группа - текущая

-1 всем <-1 группа - берем по модулю и ищем группу

kill(-1, SIGTERM) - убиваем все, killall - по имени
↑ даже от fork бомбы спасет или нет

Про группы

pid=101 pgid=101

fg 1 ↑ номер в списке фоновых
↳ продолж работу фонов проц

setpgid(pid, pgid)

setpgid(0,0) - созд группу имени себя

setpgrp()

Сделать группу фоновой to setpgrp(fd_term, 102)
↑ группа, кот будет сист фоновой
↓ чтобы узнать /dev/tty

Что делает терминал

1) setpgrp()

2) char name[MAX...]

ctermid(name)

int fd_term = open(name...)

if (p = fork() > 0)

tosetpgrp(fd_term, p)

else

setpgrp()

exec()

fd - переводит проц из фоновой в активную

Сессии / Сеансы

setsid()

sid = getsid(pid)

pid=102 pgrp=101, sid=101

setsid()

pgrp=102, sid=102

Если хотим сервер

for(i=0; i < MAX_OPEN, i++)

close(i)

if(fork()>0)

exit(0)

setsid() # Никаких упр терминалов => никто не пришлет сигналов

Новый метод обр сигналов

sigaction(sig, *action, *oldaction)

структура

sigaction

sa_handler SA_SIGINFO

sa_sigaction SA_RESETHAND - явно хотим сбросить на дефолт после обр

sa_flags

sa_mask - если все 1, то можем спокойно работать никто не пришлет сигнал kill проигнор.

SA_RESTART - прод. после сист вызовов читаем из файла, приходит сиг

Обр можно расс как отдельный поток

17.05 Синалы 20 лет в тумане

Правила игры в синалы

raise (signo) # самому себе

kill (getpid(), signo)

abort(), SIGABRT - можно для отладки исп

sig_atomic_t - проц, тип которой можно исп в обработчике

Делать ++ с sigatomic_t не нужно (разная архитектура)

sleep

nanosleep

usleep

struct sigaction act;

act.sa_sigaction = myact

act.sa_flags = SA_SIGINFO

sigaction (signo, &act, NULL)

act.sa_mask - маска того, что недолжны получать

#include <signal.h>

sigset_t set # работа с битовыми полями

sigemptyset (&set)

sigfillset (&set)

sigaddset (&set, num)

sigdelset (&set, num)

sigismember (&set, num)

Можно временно приост дост опр сиг, при этом не убирая обр

sigprocmask (how, &newset, &oldset)
← новая маска

sigint=2 ← позв отложить обр некоторых сиг.

how: SIG_BLOCK

SIG_UNBLOCK

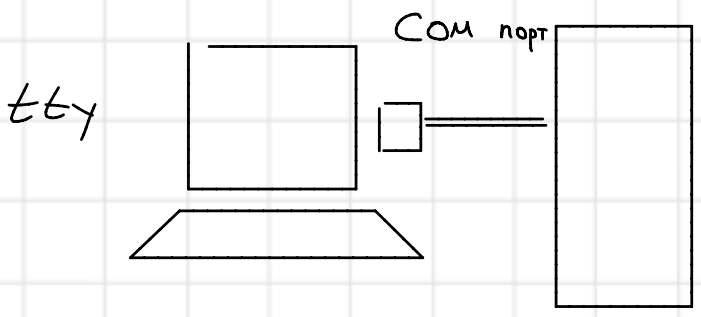
SIG_SETMASK

sigpending (&set) # делаем обработку висящих сигналов

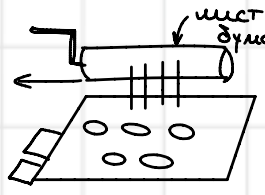
sigsuspend (&mask) # заснуть в ожидании определенных сигналов

myact (signo, siginfo, code)

siginfo_t {
  si_signo; ← из того места, где нас прервали
  si_errno;
  si_code; → SIGCHLD
  si_pid; CLD_EXITED
  si_uid CLD_KILLED
  si_status CLD_STOPPED
}



Электр печатающая машина



Появление shift, CAPS, Backspace

CarrigeRetwen - CR

LineFeed - LF

0 ctrl

31

спецсим

64

Latin

128

Задействовали ESC 27_{10} 033_8

- далее идет послед, кот не обр напрямую

vi

man termcap

*Esc и быстро - 5

Stty sane - Возвр к default парам

24.05 Дост kill abort синхронная

Сокеты

FIFO /tmp/fifo1 ← исп на чтение/запись

⊖ не понятно кто пишет, можно надстроить свой протокол

⊖ Атомарно, если мало данных

Архитектура

/tmp/fifo.srv

↑

клиент пишет свой pid

После сервер созд /tmp/fifo.pid и клиент пишет туда

⊖ кто-то туда пишет еще

⊖ не понятно как подчищать

1982 г Beerkly → Сокет

Сокет - объект

connect

имя 1 ⟷ имя 2

имя 1 - имя 2 - уник объект

имя 3 - имя 2 - снова уник пара

имя C ⟷ имя S

Клиент ген случ обр имя свое

Клиент должен знать имя S заранее ⇒ при подкл ген новый сокет

Разбивка сообщений

Есть поток - stream - надежный, упорядоченный

Datagramm - ненадежный, неупоряд.

└либо целиком / либо нет

Раньше кроме сокетов были интерфейс tli

int fd = socket(domain, type, proto) # резерв файл. дескриптор

domain

• AF - простр имен, AF_INET - домен интернета (ip, port) - уник.

AF_UNIX

AF - addr family, PF - proto family

• type SOCK_DGRAM • proto - низлежащий протокол

SIREAM Обычно 0 → сист выб сама

SpQ

RAW

```
int fd = socket(domain,....)
bind(fd, &name, namelen)   ← особ структ

struct_sockaddr {
   int sa_family ← AF_INET
   char sa_data[14]   # обычно char sa_sunpath[108]
}
```

Для TCP

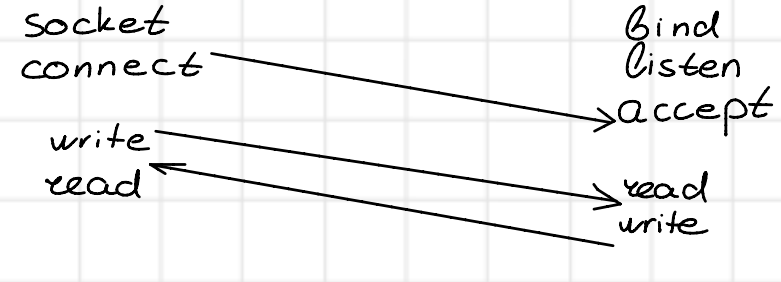
```
struct sockaddr {
   int sin_family
   int sin_port ← это все в BigEndian => нужно преобр
   int sin_addr
}
```

int_n = htonl(int_h)

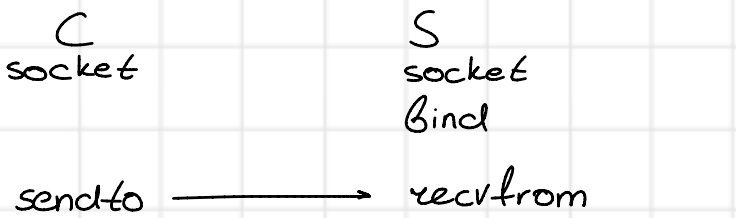
S: listen(fd, backlog) # переводим сокет в режим фабрики
↑ число в очереди

C: connect(fd, &saddr, addr_len)

S: newfd = accept(fd, &saddr, addr_len) # можно сразу fork после этого



С точки зрения Datagramm



(fd, buf, buflen, flag, &servaddr, slen) # в каждой дейтагр. есть инф об отправителе

Флаги: MSG_OOB ← через данные
MSG_PEEK ← проверка, есть ли что в потоке в обычных данных
↑ чтение происх, но данные не удаляются

%windir% \ system32 \ drivers \ etc \ services

libnss

* strace [команда] (system) lsof
ltrace ...
↑ lib

* select(n, in, out, err, timer) ← max ожидание # для асинхр отслеживания

4 - 3 станд + сокет
Если нет данных - засыпаем # часто исп как sleep на миллисек.