

IBM 360 - мультипрограммность, exec. упр. буф. name.

PK - exec. буф. обр.

компьютерные программы работают со стар. DOS

DOS - однозадачная  
но! могут вмешиваться программы  
→ переход на более новых версий.

\* int 8h вызывает в режиме реального DOS - он многоцелев.

Intel (Microsoft) - отработан совместимость  
т.е. один и тот же языки могут работать  
как стоящие

Различные разделы компьютеров

1) реалモд (16р.)

2) защищенный (32р.) - отсутствие языка высокого уровня.  
• многоцелевая  
• поддержка параллельных процессов

2a) специализированный языков высокого уровня V86.  
• многоцелевая

3) long (64р.)

3a) ~~real~~ compatibility - есть 32р.

low / high

\$

амортизация дискуSSIONS.  
обесечение обратной совместимости

ISA - старые чипы

применение концептуальных схем

стартапизация для новых.

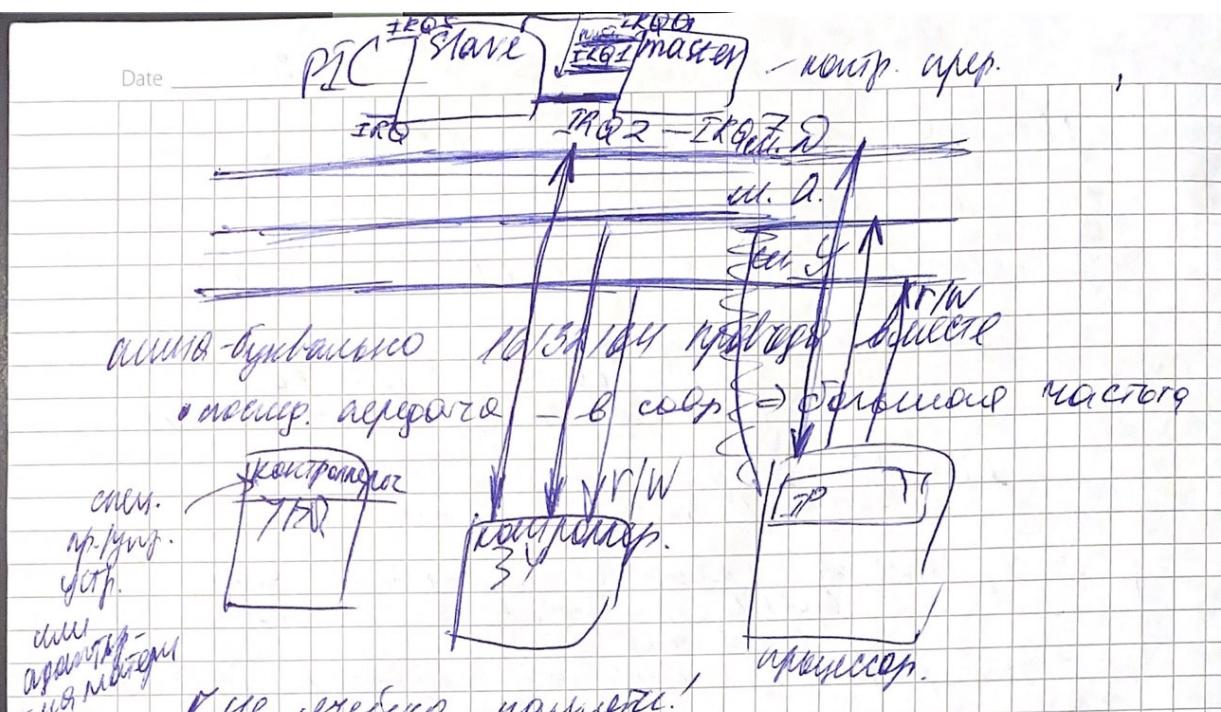
OS - управление процессором; основная задача - обработка запросов на обработку языка высокого уровня языка. Прим. обработка отмены

использование языка.

1) языковое (языковое, компилятор)

2) адрес

3) машинное управление



У нее есть одна наименование!

useful for us.  
Op-tr (by men.)

## Линейное преобразование

ans.  
negative answer  
(intercept)

Intel 8086

MACK Say. b-p. + N° TPA

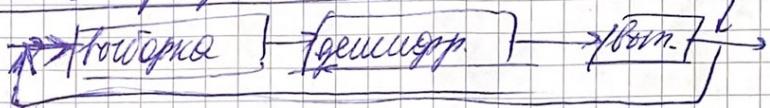


- IRA12 - known PC12
  - IRA0 - int 8b
  - IRA1 - membratyp PC12

Bei W.W. - Mercede-  
Mühle  
(Kalkal-TD nach  
Johannesburg)

уровень  
Date  
сигнала INT

Боcнe аpгрaммc.



INTA - обработка от нос.  $\rightarrow$  конт. на в. д.  
1MS  
бокс. б-р.

FFFF

x4, -16cc

$\rightarrow$  Таб. б. нр. 00000

Применение инт. наимено.

Применение команд lock - для (неизвестно)  
команды/вс. форм. как ожидается -  
прерывания/вызов  
о блоке/весь цикл. можно использовать  
(использование зап. команды/вс. форм. не ожидается)

Коды 3(4)

L1, L2 - для в. эпиз. / команды, генерир  
L3 - вызовы  
TLB -

PCIe

PCIe.

Однокарта соед. на уровне оп. системы.

MOW - с памятью

IN, OUT - порты ввода-вывода

Многократный вызов не требует времени.

1) функция return - exit(1)

2) child or parent. exit(0)

exit() - сущ. вызов just-in процесса  
return - выз exec'а

### 3. Насл. и производн.

- массив временных переменных потомков - процесс ясно

DEFINE - красиво, но не нужно бт

просто написать размер массива - лучше)

- 80 симв. на 50 строк -  
- давно, временно вен

- ищ/р строк - израсход,  
использование памяти,  
(использование 8. памяти строк)  
(не будет ошибок)

строка должна помещаться  
в экран

все тек. значение существует  
помимо начального в экран,

иначе - непод

также для 2<sup>nd</sup> child

main()

```
pid_t childpid;
if ((childpid = fork()) == -1)
{
    ...
    exit(1);
}
```

else if (childpid == 0)

```
printf("child pid=%d, ppid=%d, grid=%d\n",
getpid(), getppid(), getpgid());
```

усл. вспл.

parent sleep(2); // чтобы получить ожидание процессов

затем printf(..., ...);

или exit();

else { ... } return 0; }

! В новом - parent kill. parent, child, это не do require.

readfile.

printf

Процесс - параллельный т.е. том-то тока, новый  
производное тело

- многоязычная система  
struct task\_struct - есть информация о процессе.

2) в parent вызывают waitpid.

урут процесс  
(массив child[0..8])

на  
максимум

- parent будет передавать к 20 и ненужной  
линии (11-ю) ошибок. Высаживаем else,  
а в main

Wait - в for'e, но möchte better else ?  
for  
else

одинаково for

нечто: машинное слово, не напоминает

+ anomaly по микросхеме

(3) exec

в fd оп-ах

Следующий шаг exec, как  
запуск, создает низкоуровневый процесс.

sleep считало 10 единиц  
т.к. parent выставляя waitpid  
шаг не делает, most sleep  
заперегревает машину в теч  
о машиной считало 10 единиц

"они" говорят без этого  
процесс не соз., не идет, но /bin/mount не останавливается -  
показывает что процесс адресное пространство.

это значит, что созданный машиной отрывается.  
другими словами, если программа не находит  
сопр. машиной страницы, сопр. раскидывает страницы.  
если нет страницы, то она уходит.

Exec начинается с проверок

↓ проходит, создают

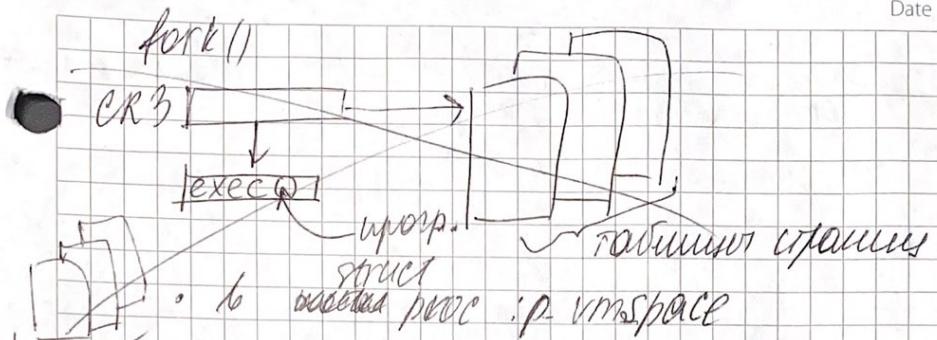
некоторые данные для них. mount при этом в ре  
зультате 0, а в IP - адрес точки входа (main)

↑ главное  
так, не зря начали!  
(если обратить внимание)  
(если обратить внимание)

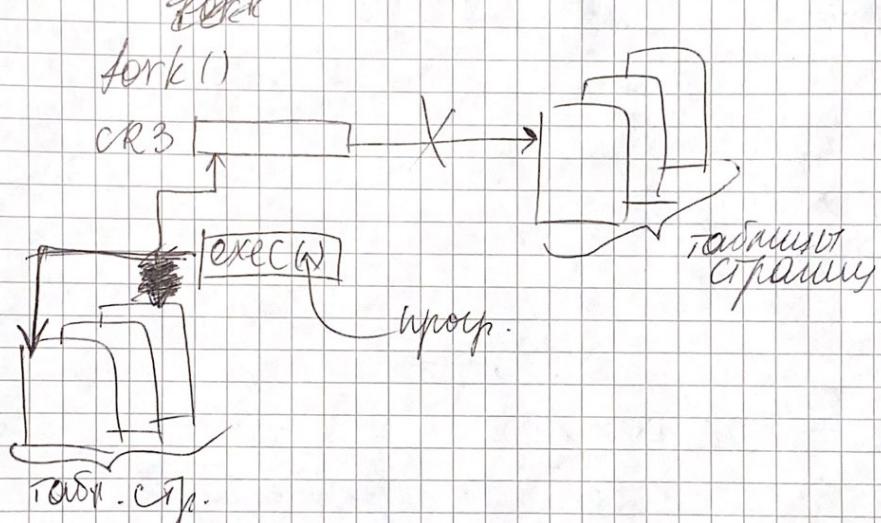
. точки входа 0: сис. вызовы, осевая, они. пропорционально  
длины точки входа - драйвер.  
драйвер преобразует - . . .

также инициирует они. контекст

→ в примере CF3 - я инициирую контекст.  
т.к. запрещено - запрещено = некий. страницы  
или преобразование.



такж.: exec() бываете не наше вложенные fork.  
стк. адреса памяти адреса вложенных



на процессорах:

- 5 пер-роб упр-я CR (0-4)
- 0 - самое чест-е присвоение
- 1 - не чест.
- 2 - мин. адрес стк. keygen.
- 3 - адрес вложенных табл. CR (x3d) / DAE (x64)
- 4 - адрес общих табл. CR (x3d) / DAE (x64)

1. Проблемы myth (no cache)
2. Имеет ли пот, кто пишет, права на бл-р.
3. Проблема запоминки, но то, что-ли исчезает не расширяется
- 9.

sec Проблемы таблицы страниц.

✓ 12 - нет CR3

exec1, execV  
null Таблицы  
 массив

(3)

• не менее 2<sup>х</sup> аргументов кор. боя. прер.  
с блоком или else, если блоков, это прерывание  
будет блокированным из-за него

- из volatile - нет, T.II. булевоуорер - как не-  
действ. монитор - нет.

если внутри child - exec(), то приставка никогда не нападает,  
так как exit

(4)

Потомки и предок один. сообч. от изменяющихся  
предков прор. канал (не pipe)

int main()

int fd[2];

pid\_t childpid;

→ pipe(fd);

дескт.  
боя.

... . . .

Меню. прор. канал

на прор. каналах реальн. ведется

каналы не имеют адреса, если у него нет.

→ гол.ство address. массив от групп.

один файл...

Файл write, close, read - никаких ошибок.

- не менее 2<sup>х</sup> аргументов

одн. потоки пишут в один прор. канал!

- замкнутый цикл. исходя из этого. предположим

1 child - 3 блоков 2<sup>3</sup> - 10 блоков

parent пишет

где разб., что

удобно

избр.

рекл.

размер файла, он

Прор. каналы конт. фативное исключение, т.к. они не  
дают спец. ошибок.

Прор. каналы будут на 3<sup>х</sup> уровнях

1) спец. кр. памяти Если спец. память перед. то

Много, или больше нет. память на диске.

Если адреса памяти одинак. в концом > 1024,  
то будут будут по времени со тек. мон. пока есть  
места - и это надо не будет считывать

Если процесс завершил 90% то он будет готов, когда другой процесс ее ожидает то ему можно дать членство в кластере.

Нет памяти, есть нет - ошибки.

- выдает оп-бю fy-a всех первых в очередь.

Первый оп. б 30 раз быстрее, чем алгоритм рефлекции (размытие-написание)

Оп-бю fy-s, б UNDO для S

II метод.

fy-e и fy-mem-mem - только представители

- ~~fy~~ S.M. последовательно обрабатывают открытые страницы

FIFO - в сист. ОЗУ например, потому что оп. нр. 100 проходит проверку. Там один процесс не может обрабатывать 2 оп. нр. 100 другого процесса - другой процесс занят процессом. Исправление.

В голове я/з лучше - оп. нр.-бо опред

Оп. - Оп-то исп-р. в процессах о событиях, которые происходят или в самом процессе, так и вднх процессов.

• Такое событие называется сигналом по ул. к проц. (encl. sig, sign.)

• активизацию, (акт. перев-бачен)  
им явл. от родств. процеk. в системе)

скорость. +1 аудит  
события с сигнализацией и гашением

• Установка-исчезновение не явного сигнализирования, таки - факт  
исчезновение (исч. явления процесса, сигнал/нуль.)

Как правило, бывшее событие в OC - яв-е процесса.

Идея. сигналы, сопровождяющие процесса.  
Основно, полученные → яв-е, но! сам процесс может  
опр. свою реализацию:

- 1) сигналы и.т. пример.
- 2) определение способов получения. (т.е. си., опр. ф. ОС)
- 3) определить каки. реагит на каки. сигналы

• Правило в методике, неправильные сигналы - синдром  
(синдром - крит. Unix (PDP-11), 20 сигн.)

• Самое первое имеет свойство отображения сигналов.

#define NSIG 20  
#define SIGHUP 1 Номер сигнала  
(1) - исключительное яв-е сигнала

• Всего. сигналов будет 20 ⇒ не ясно каких сигналов будет.

- SIGHUP - сигнал, возн. при разрыве связи процесса с упр. терминалом (ремот. с ? - нет связи с терминалом. Т.к. SIGHUP инициируется) { все сигналы не имеют упр. Т.к. перед SIGHUP не выполняли, поэтому сигнал инициируется.} 18:00
- #define SIGINT 2. - сигнал завершения процесса  
(CTRL-C - принуд. заверш. процесса -  
пользовательский сигнал не получит, в результате от kill - signal 15)

#define SIGQUIT 3 - CTRL-1

#define SIGKILL 9 - сигнал, возникает когда б  
нет под. сигн. для kill. существует kill - сигналами.

#define SIGSEGV 11 - (напоминание о памяти)

т.к. в класс. ОСХ - система проверяет  
память, а в си. - ограничено памятью

• Проверка памяти  
ошибки

Date

- Потомки наследуют статус копа памяти.  
Но если копа - не существует, то статус.
- Гарантийное значение - архивное значение копа
  - Аварийное значение

Следует упомянуть еще один.

#define SIGSYS 12	- ошибка бинарного сист. языка (исключение) непр. - 1, значение SIGSYS)
#define SIGPIPE 13	- замыкание трубы (исключение)
#define SIGALARM 14	- сигнал будильника (исключение)
#define SIGTERM 15	- kill в норм. спос.
#define SIGUSR1 16	{ замыкание
#define SIGUSR2 17	
#define SIGCHLD 18	- уничтожение
#define SIGPWR 19	- нарушение напряжения, исчезновение питания реактивных, сопр. ядерных.

Если в UNIX 0-8 биты в коде сигнала определяют приоритет сигнала (приоритет сигнала, напр., 0, 1, ..., 7), то в WINDOWS сигнал определяет уровень сигнала - 0, 1, ..., 7 или 8 - IREL (Interrupt Request level).

То есть арх. опред. имеет набор сигн. приоритетов - набору сигналов соответствует определенное значение.

#define SIGPOLL 20	- событие в stream-устройстве (для флагов типа pipe)
--------------------	---

Использование.

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

Операторы надстройки и функции командной обол.: kill, signal

kill

int kill (int pid, int sig);

pid - идентификатор.

sig - номер сигнала.

если номера совпадают:

если pid <= 1, то sig будет передан всему процессу.

Группы процессов могут назначаться один и те же  
автоматическое назначение на них реальных групп  
предоставлено - ветвление по группам)

Функциональные группы процессов - временные  
(все процессы, вышедшие из одной группировки)

- если  $pid = 0$  - текущий sig будет менять  
личную группу с sig. группы, если с sig. группы  
процесса, выб. kill, изменяется процесс с sig. Out.
- и т.д. (см. manual)

8 Наиболее интересно:  $kill(getpid(), SIGALARM);$

$\uparrow$  current  
sig. current  
process  
 $\downarrow$  current  
group

### signal

- не рекомендуется использовать сигнальный ID.  
(см. signal Linux man)

### POSIX

1) POSIX.1 FIRST Portable Operating System Interface, based on UNIX,

Federal Information Processing Standard

- определяет минимальный уровень с POSIX.

8 ISO - не единственный стандарт.

- напр. определено: поддерживает POSIX.1 - 1988.  
(около 1000 сигн. поддерж., кот. помимо подд. OS  
с пересечением суп.)

POSIX.2 - расширение, под. еще один стандарт.

2) X/Open Portability Guide - б. Elphose

XPG3

XPG4 - 1994г.

- ANSI C, POSIX.2 + гос. конструкции.



↑ есть выбор, кот. рекомендуется использовать по умолчанию SIGACTION:

```
#include <signal.h>
int main()
{
    void (*old_handler)(int) = signal(SIGINT, SIG_IGN);
    ...
    signal(SIGINT, old_handler); // но лучше, SIGDEF настройки
}
```

Чтобы: ↓ отлад. функции → не вызываться по умолчанию, но не блок. сигналов (↑) → это шаги (а также SIG\_DEF)

```
int sigaction(int sig, struct sigaction *action,
             struct sigaction *old_action);
    ↑
    старые структуры
```

1. sigsetjmp()

↑ функция ~~запуска~~ на блокировку  
асинхронное событие

2. siglongjmp()

↑ блок. pause - sleep

открытие  
файлов, а не - можно не пропустить

... навсегда? → always

избывает уст. исключений тоже баги.

избывает неизб. ис. если бы тоже не переходил

Win: Класс RunTime "Вызовы где-то в-е" выходит из mutex

Linux: Действия зав. не прояв- ляются.

но-все. Хорошо  
(сигнализ., разр. памяти)

Семафоры

- одно из хр-в Unix-подобных (System V)

(семафор)  $\uparrow$   
программное  
семафоры

семантическое разрешение памяти

Свойства:

- внутри процесса - синхронное или асинхронное
- вне - асинхронное (т.е. неявно от другого процесса)

Бинар.

один  
бинарн.

двоичн. (абсолютно)

аппаратное

т.к. величины для пред. не  
зависят вообще ни от какого прио-  
ритета в системе (т.е. нет  
изменений)

Все бинарные семафоры можно разнести как индивидуально  
одного процесса другому.  $P_1$  -  $P_2$  |  $V(S)$

? нужно либо, захват и освоб. не должны быть  
"параллельны" (т.е. один захватил, а другой может  
остаться)

Это имеет место действие в системе

одн. семафор или несколько

( $P, V$ ) действует на используемые в системе  $V, P$ )

При исп-ии процессами большого числа семафоров есть  
запрос блокировки, превышающее возможное (такие  
блоки см. в Linux)

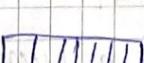
(одн. семаф.: пример Linux)

Создают набор структур семафоров:

- одна определена для каждого хр-ва

В ядре Linux имеется таблица наборов семафоров:

<sys/sem.h>  
struct semid\_ds



Набор семафоров

таблица наборов семафоров.

! Задача: система газации в Г.К. блокировать и разблокировать может лишь одно, а открыть все может лишь с выдачей звук.

Про каждое набор семафоров чувствует:

- 1) имена (число имен)
- присваивается процессом, создавшим набор
- другие проц. по имени имени могут откр. его
- 2) VID
  - { содержит набор и имеет идентификатор}
  - идентиф., VID ког. = VID сог. может указываться и удалять сост. набора
- 3) права доступа (о - a)
- 4) Количество семафоров в наборе
- 5) время ожидания одного или нескольких максимумов семафоров посыпраем просится
  - { син. "ограничение-до" или "после"}
- 6) время ожидания упр. параметров набора к любому проц.
- 7) Указатель на массив семафоров.
  - массив - семантически.

На семафорах определяют след. виды операций:

**control** **semget()** - создание набора семафоров  
**operation** **semctl()** - изменение упр. параметров набора  
**semop()** - опрашивание на семафоре

int semget (key\_t key, int num, int fl);

↑  
ключ и набору  
(num)  
↑  
если-то  
семафоров  
в наборе  
↑  
IPC-CREATE  
при создании

В отличие от сис. Альбакрот не сис. UNIX опр. фундамент.  
Но это определено

struct sembuf

```
{
    ushort sem_num; // номер семафора в наборе
    short sem_op; // операция (+, -, 0)
    short sem_flg; // флаги, опр. на семафоре
}
```

1) sem\_op > 0  
увеличение,  
сост. v

2) sem\_op < 0  
уменьшение,  
сост. p

3) sem\_op = 0  
запр. опрашивание если  
семафора есть. Тогда  
т.е. как минус

## IPC - Inter Process Communication

sem - fl → IPC\_NOWAIT

- процесс не использует переходов в ~~запрошенный~~ блок. ~~в~~ соз.
- опирание об об. семафора. (если семафор занят)

- для того, чтобы при завершении процесса все процессы, блок. в опирании на заня. семафорах, не были они. навсегда  
(т.к. SEM\_WAIT неожиданно перех. и процесс не сменит обоб. заня. или семафоров)

SEM\_NVNO

- необходимость контролировать ж-й процесс пооч. состояниям семафора (ждет - передает) при завершении существ. процесса отменяет сделанные процессом изменения состояния.

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_IROTH;
    int fd = Semget(100, 2, IPC_CREAT | perms);
    if (fd == -1)
    {
        perror("Semget");
        exit(1);
    }
    if (semop(fd, sbuf, 2) == -1) // сразу на двух семафорах набор
        perror("semop");
    return 0; // ж-й семафор занят
}
```

Семильт разделяемой памяти

- базер
- один процесс записи, другие процессы могут читать
- записи не удаляют ж-ю, если они были прочитаны

Соединяется в адр.пр-ве ядра системе, т.к. адр.пр-ве процессов являются делимимыми, ... Глоб. pipe]

В таблице Э таблица семильтов разделяемой памяти:

```
{sys/shm.h}
struct shmid_ds;
```



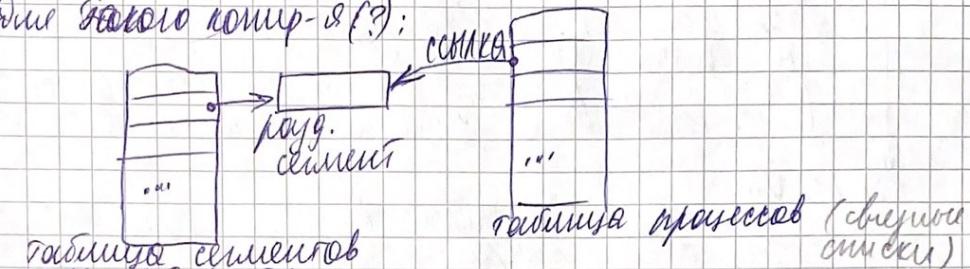
Таблица семильтов  
разделяемой памяти

разделяемый семильт

В структуре от процедур вызовов (если у кого есть inode) page скимает page таблицу имен. И это структура.

• Исключено копирование из буф. ядра в буф. памяти при записи и ком. из буф. памяти в буф. ядре при записи.

две ядерные копии - (?)



• pointer - нечестн. (если забыл память)  
память - система контролирует буф. ядро ярко

Синхронное боябот:

shared  
memory

shmgmt();  
SHMCTL(); control  
. Shmat(); attach  
. SHMDT(); detach

Пример

```
```
#include <sys/shm.h>
int main()
{
    int perms = S_IRWXU | S_IRWXG | S_IROWXO;
    int fd = shmgmt(100, 1024, SPC_CREAT | perms); // создание page. струк.
    if (fd == -1)
    {
        perror("shmgmt");
        exit(1);
    }
    char *addr = (char *)shmat(fd, 0); // подключается к page. структуру
    if (addr == (char *)-1)           // синхронизация с памятью
    {
        perror("shmat");
        exit(1);
    }
    strcpy(addr, "aaa"); // отправляет в page. структуру сообщение
    if (shmdt(addr) == -1) // отключается от page. структуры
    {
        perror("shmdt");
    }
    return 0;               // page. структура не нужна, т.к. сообщение перед
}
```

[ув. manual]

Рассмотрим

void \*shmat(int shmid, const void \*shmaddr, int shmfl);

№ 1. Раб. семафор определяет системное обращение

- SHMMNI - макс. кол-во раб. семафоров, кот. могут ж. в блоке одновр; если превышает - сист. выдаст блок, надо држ.
  - SHMMIN - мин. допустимый рабочий семафор
  - SHM MAX - рабочий максимум
- ↑ превышение ⇒  
запросов

A/p 4

1<sup>я</sup> программа: Файкера "процб.- потр." на 3<sup>х</sup> семафорах1 - счит  
1 - бит.

семафор "Буфер - пуст"

семафор "Буфер - полон"

2<sup>я</sup> программа: монитор Хадра на сем. раб. семафорах

- в общих: сущ. набора семафоров
- внутри себя,

команды раб. действие на раб. звуков семафорами

⇒ 4 массива структур (1000 одинаковых, переданных функции fork() - раб. действие на раб. звуков)

8 бит. семафор - захвачен критической секции

(блокирует)

Требование И.И.О.:

- массив не обнуляется
- все действия - с начальными указателями
- с буферами процб.- потр. т.к. в прохождение - не менее 3х процб. fork(),  
3х потр. - активн. процессов
- не может инициировать оуп.

8 абсолютно случайных ядерных (случ. нач. учт.)

5 буферов - битовых лог. управ. (26 битов)

(5024, шаг 96,128)

Процессоры не имеют ширину, но какому адресу

бита соответствует буфер / страница ??

сама запись предполагает, что будет обработано первое -

и т.д. по порядку, а не как работает буфер

8 буфер  $\rightarrow$  тек. адрес памяти  $\rightarrow$  физ. адрес  
~~буфера~~  $\rightarrow$  физ. буфера

какой?

и? Видимо буфер ????

Чт. памяти. все может быть, какая буфер назовется  
 $\rightarrow$  надо смотреть?

(2)

! Буфер пам., ведом к напеч. не надо (читка: read,  
 writer  
 creat.)

3 синхрон

1 бин. синхрон

2 языковые. писатели

4 опущ

start write

start read

stop write

stop read

! Можно создать генератор синхрон -  
 ~ обратное 8-е (5 един.)

6 ~~посл~~ памяти - адрес пам. типа integer

назади есть

После и то же значение будет проранжировано пост-  
 назади

7 не меньше

3 нее,

5 нее!

здесь пропущены по синтаксу SIGINT

исключение

исключений

отловлены кризисов

sigbreak

- Задачи:
- и.о. уст. порядок один краевый
  - б) грануляционный краевый (установка маркеров serializable)
  - в) пустяжный краевый

Семинар 6  
Очереди сообщений

21.11.23.

Прим. канал - буфер типа FIFO  
нагг. оп. исполнен  $\rightarrow$  имеет значение  
один! нах. в данный время.

3<sup>е</sup> оп-бо - Очереди сообщений.

Работа с очередями сообщений

- ваге

(sys/msg.h)

struct msgqid\_ds

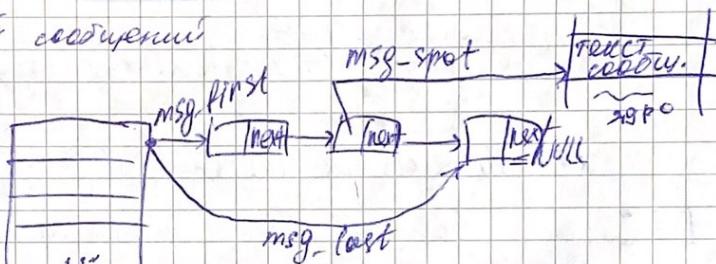


Таблица очередей сообщений

msgget();

msgctl();

msgsnd(); — назначение блокир. вызовы способ обмена

msgrcv(); — прием / формирование вызовов процессора

Причес определена структура:

struct msgbuf;

```

    long mtype;
    char mtext[ MSG_MAX ];
}

```

В application при-бо есть одна интересная таблица очередей сообщений.

Каждая ячейка

делит очередь между ук-бо и другую единицей приложения — msg-spot.

Новая функция перегородки сообщений. В 87-88, когда создавал

дни не было языка и писалась ее

В качестве нового сообщения:

- new message.
- текущее имеет опр. размер сообщ. (размер в б.)
- ук-ко не имеет данных для, где описано как сообщ.

Слого занимает сообщ. и уп-б в отпр-ке в обн. Головной сообщ., после него проходит некий заг-св, а сообщение остается док. группой пд.

Новый профес. формирует сообщение из отчёта, слого занимает сообщ. в обр. ип-б прошлого, после него группируется (напр. rec.)

Помимо формата спр. ему:

1) Конфиденциальность, не заб. от типа  
(из конф. лог. отчёта)

2) по криптографии (нуу?)

[I] Если ф. неиз. , то какое ставят

3) Числовое значение типа конф-го зда. минимальных  
и максимальных значений гарантировано неиз.,  
ипр. профессор

(мин. число 15, а это 14, 13, 12, 11  
важнейшее 11)

Если ф. неиз. убирает  $\rightarrow$  какое ставят

Криптография = new

Дво. симметрия, кредит аудитора не отмен-на опр send и  
receive

Ключи.

```
#include <sys/msg.h>
#ifndef MSGMAX
#define MSGMAX 1024
#endif
struct msqid {
    long mtype;
    char mtext[MSGMAX];
} mobj = {15, "aaa"};
int main()
```

```
    if (fd = msgget(100, IPC_CREAT | IPC_EXCL | 0642));
```

|| не получит  $\rightarrow$  if (fd == -1) perror("msgd");  
 $\Rightarrow$  return 0;

также нет опр.  
на q.  
EXECUTE CONTROL  
• write & exec & pagefault  
использование  
капабилити  
ошибки

~~Ба~~

Т.о. открытие созданных нейлоновых файлов происходит  
автоматически (если не explicitly)

\* Работа с открытыми файлами

daemons

g - super user

l - different.

democratize

already - running

1. umask(0);  
↳ parent

2. fork() в заб. под. процессе.  
рабочий процесс, который генерирует, не создает ни один новый процесс

такой процесс наз. ...



3. setgid();

session identifier

execve - запуск нового исполняемого файла.  
онесёт в новую программу группу  
надирессов / группу Г кот он опред.



↳ Linux 2 раза fork все новые

(3 группы agents!)

4. chattr('r'):

запись этого момента должна стоять. то, что ты пишешь

5. [Работа с открытыми файлами]

• макс. - 1024. (не более 1024 для каждого отп. при.)

тако ограничение открытия: struct rlimit

↓

getrlimit(RLIMIT\_NOFILE, &rlimit)

if (rlimit.rlim\_max == RLIM\_INFINITY)  
rlimit.rlim\_max = 1024;

for (i = 0; i < 1024; rlimit.rlim\_max = i + 1)

close ('');

Date

fd0 = open ("dev/null", O\_RDONLY);

fd1 = dup (0);

fd2 = dup (0);

current already

- openin ~~иницирован~~  
+ from no games

- ~~запущен~~,  
но он не reg.

Можно создав, можно из списка

В это место пройдет адреса super user

more - менюшко: point more. we & go.

STG ACTION ?

Devnull

• в sys. log ← unit 30 ссылью journal файл  
нет буфера

## 29 часть L14.

внешн. атриб.

- 1) пр-е соравнение <sup>пред. от</sup> сист. таймера в исчезах разрешения  
транзит
- Windows
  - UNIX/Linux

2) - Ошибки по тайм., но не тик, но клаву <sup>3)</sup>

Важание, Синий-Русский

и. "таймер" - все неправильное

Обпр от Сист. Тайм - вон. на восс. ур. примерется  
(в любой ОС)

"быстро" - вон. от начало до конца  
(таймер - ег. "быстро" - нечто первое  
вспоминание)

• не может вызывать другие модули ядра

- { 1) переход на адрес ~~назад~~  
2) return возвращает вон.е вызваный программе  
(отложенное действие)

→ неравнозначно для таймера; он шири, отложенное вспоминание

Например.: "вызывает машиновызов" [Важание] - неверно  
Верно: инициализирует ряды машиновызовов

- DPC - different procedure call - в Windows
- S-аргумент (установка) - UNIX

Машиновызов - вызовение первого производного ядра  
транзита

... генерация клави

по иск. явлам

→ Готовый машиновызов - по и. тику

Он Готовый тик - тик, проск. с передачей нек. тиков.

- ик и.г. несколько

потому что, когда идет явл., ядро, DC - передает явл.  
другому процессу и срывает в соотв. с машиновызовами

## 2) динамические приоритеты

- могут быть только у подр. процессов

Динамич. - при. в процессе работы

Статич. - подр. фиксировано в момент создания

|     |                   |
|-----|-------------------|
| AA  | 1) время горячие  |
| D37 | 2) мед. горячие   |
|     | 3) все времена    |
|     | 4) пакет          |
|     | 5) горячие по ум. |
|     | 6) пакетно        |
|     | 7) горячие        |

1) UNIX/Linux

2) Windows

} наименее оптимальный пересчет приор.

в стеках сменяются;

исключается бек. отк. процессов (засчет дин. приор.)

- упр. время ожидания процессов в от. крат. исп.

■ UNIX: ограничена пересчеты из [Вашингт.] (это старое изв., сейчас - ядро - 7,84)

• упр. "время ожидания" - не только время ожидания, но и полу-  
ченное прог. время ↑ уменьш. приор.

• если форма чист. фн. приоритета, то он fix

■ Windows: [Соловьев - Аксинов] - "нет опт. между мониторами"

- напоминает, как упр. время ожидания  
процессов в очереди готовых процессов

Продолжает практика, начавшуюся в то время, когда  
на "такой был блокировкой процесса" - или [Радченко]?  
или (на звук. карты, сканеры, мониторы, мыши'е) -  
на сколько? (Windows)

## 3) потоки - ~ предпочтение - сумма интенсивности

в многое ОС: • процесс - единица ресурс.; ресурс -

- поток - единица распределения;

Процессу выделается предпочтение; поток имеет пред-  
почтение относительно других потоков.

• без него, разработ. - сложн.

daemonize [dæmənайз]

sigactions - инфр. SIGKILL

setsid - управляет инфр. термином

openlog - инфр. ... журнализм

cyslog - назначает отч.

### Преимущество потоков

Процессы, в кот. не создаются потоки, расходуют гораздо  
(меньшими потоками)

### Процессор

- fork

- создает АП  
(подчиненный СП.)

- габариты СП. СОВ.  
распр. АП превышает

[но ~~но~~ сразу после fork  
→ fork-е не exec  
все условия "прав"  
(имена АП на АП превышают)]

минус fork'a

### Потоки

- не созд. создает АП

- всп. в АП проще

- д-р. собств. стек  
(всп. асинхронные операции)

Основной недостаток - дну. памяти.

Чтобы обойти эту. они. попытка блокир. ядро АП  
помогла (выгружив его).

### Распределение времени

по контексту

по функциям  
блокировав при запуске  
блока ядра - в основе

Принц.: обратном порядке

Принц.: генератор

Закон Мурана,

$$S_p = \frac{1}{f + \frac{T-t}{N}}$$

закономерность;  
 $f$  - время исполнения ядра  
кода;  $N$  - количество ядер?

## pthread\_create

процессор организует потоков

1) стандартная POSIX

threads (pthread)

2) user. библиотека clone () - функция fork

- новые процессоры/подсистемы с подпрограммами
- выделение ресурсов

■ int pthread\_create(pthread\_t \*thread,  
 const pthread\_attr\_t \*attr,  
 void \*(\*start)(void \*),  
 void \*arg);

- 0 - все ок
- не 0 - ошибка

- Если attr = NULL, поток создается с атрибутами по умолчанию
- start - это то же что и в fork, отличие лишь в том что это функция.
- Если arg = NULL, поток не имеет аргументов.

■ int pthread\_join(pthread\_t \*thread, void \*\*data);

- блокирует выполнение, если поток не завершился

■ pthread\_exit();

- выполняет поток
- если pthread\_join & data оба NULL, то
- в этот параметр pthread\_exit возвращает значение

! Несколько потоков не могут ожидать.

! Одна (один) из них

! Все потоки работают.

! При работе нескольких потоков ресурсы надо обходить.

[ Чис-лис - методом для дополнения в программе  
 модели Киттиш - Спирса ]

методы  
 (получение - залога)  
 (возвраты - залога)

Спирс - м.б. реагирует как инвестор.

Не будет реагировать же Аугуст!

■ `pthread-cancel();`  
`pthread-testcancel();`

■ `pthread-detach();`  
• "отделяет поток" (согласно отсеп. потоку)  
◦ если не требуется `pthread-join()`

1/p.

Для параметризации `join`, `cjoin`

- 1 поток
- m. поток - врем.
- SIGTERM - free slot.

↑ `pthread-create` указатель поток  
↓ д. потока над "bbb" (зап-ко start)  
↑ бон. накопление, над бр. в. потока  
существует объект отсепта, бр. в. потока  
↑ не поток бр. в. потока  
↑ беск. врем.  
• free slot. не return  
• a pthread

• Проверка `tid` потока: `gettid()`

Беск. и/у накоплений и потоков использования

Потоки разг. синхрониз. данных процесса (сокр. управл. потоками)

→ беск. врем.

## RPC

Remote Procedure Call

— вызов удаленной процедуры.

Чтобы сделать удаленный вызов мож. параллельно  
использовать.

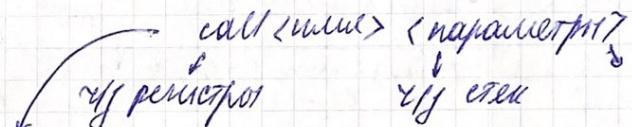
Все особенности си. я-бина <sup>RPC</sup> ~~всего~~.

Это называется RPC.

Т.к. разн о рабоч. единицах, RPC подр. на межклиент-сервер.

- две стороны: клиентская, серверная.

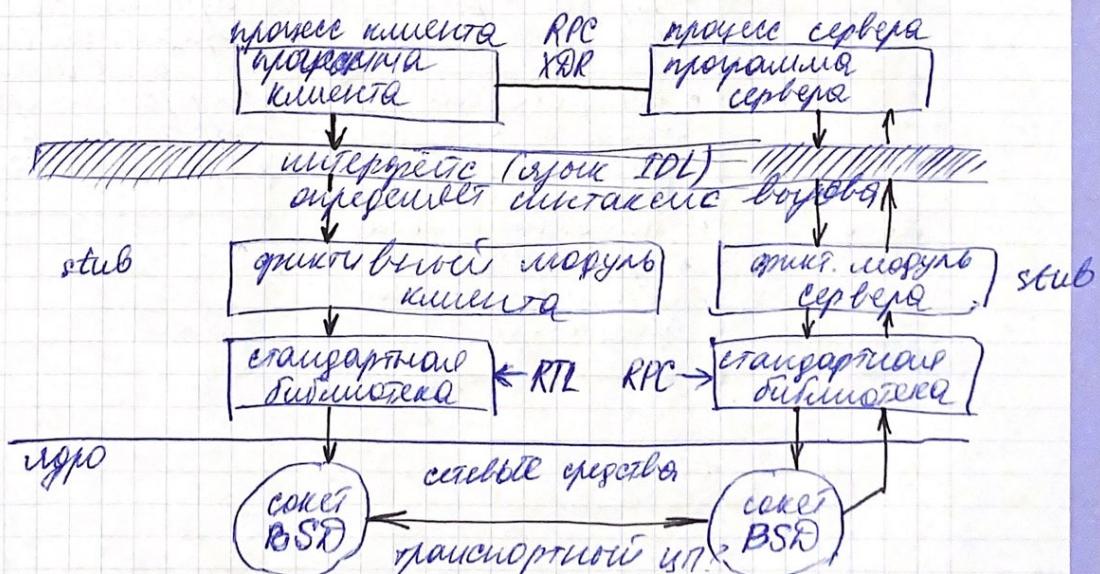
Но! Не согласно так, что запрос с одной стороны  
вызывает, как вызов мож. процедуру.



Но фактически это не так, а значение return value.

не exit'ы, а return'ы [<sup>в RPC традиционно</sup>]

Основное идея: скрыть особенности естественного языка от пользователя.



XDR - External Data Representation

Our socket - однородное конечное точка соединения. (точка между нами)

- введен в UNIX В84 (см. в то время было 4-5 языков)
- в современных системах - socket входит в POSIX.

[socket и на отдельных  
и в различных системах.]

stub - не точка; это специализированный софт. ПО, упрощающий работу

- задача: получив от программы пакет, преобразовать его в сообщение socket

! ПО для передачи пакетов - сообщение, у кот. всегда есть заголовок (пакет: заголовок + адрес) (отр. определенность - адрес - адресация пакетов)

Процесс клиента

Процесс сервера

приложение

клиентский stub

адро

как  
правило,  
усл. японии

• socket и. языка

- создание буфера сообщ-я
- упаковывание пакетов в буфер
- добавление полей заголовка в сообщ.
- переход в режим ядра (сист. socket)

- копирование сообщ-я в ядро
- отр.-е адреса назначение
- изменение адреса в заголовок
- установка сетевого интерфейса
- формирование пакета

process

• приемление пакетов

- socket сервера
- изменение пакетов в стек
- распаковка пак-та
- передачение в режим пользователя
- определение, что нужно сделать ядро
- определение адреса назначение
- определение правильности пакета
- определение времени

(если ошиб. сооб. или обесн  
= ошибка при работе  
(известна)  
soft irq

• в сети нет  
других ср. включенных  
ПО, что работа будет

timeout

26.09.23. Семинар 2 ОС.

Сердцем UNIX является ядро, имеющее структуру:

Все ядро UNIX - структура и ф-ии.

т.е. ядро UNIX не имеет структ. но модули,

→ Оставшиеся остатки примитив - ядра = ядро.  
носило бы ядро.

Linux - UNIX-подобная ОС; строится на базе  
наследников UNIX:

- открытый код
- 

Процесс запускается на страницах ядра,  
которые включают в себя ядро.

• часы времени ядра в реальном мире  
(ядро-это часы ядра), а часы времени - реальный мир ядра.

Принцип перехода в ядро

- 1) в рег-те есть ядро, использующее или нет ядро.
- 2) процесс перех. в рег. состояния ядра (если  
имеются такие же ресурсы - процессу  
передается сервис системы: ~~read write~~)
  - read-write → блокировка процесса  
до тех момента, пока  
внешнее устройство не отдаст  
данные, или отдал данные. Доп. осущ.  
запрос в рег. обратотки сообр. ядр.  
передаванием.
  - блокировка внеш. устройств. осущ. переход  
предусмотренных правил
- 3) в результате исполнения; при выполнении в  
ядре - обновление

свои сам  
1. Адреса  
пред.  
использовано  
также  
• исправлен  
шаблон

в UNIX  
2. Инициализация

• б. п  
→ оп  
3. Работа  
б. ре  
б. конт. под  
и  
— привет  
команды:  
ATO:

• и  
са  
ре  
! где  
и за

Адресное и  
таблицы  
таблицы

нас. самого gene - "безопасный ур. блок".

## 1 Анагностическое вычисление

некст.  
если иного  
также  
и не  
имеет

В UNIX Апроц. создается неск. безопасных fork().

в нескольких sys. clone и т.д.

~~анагност.~~ • исследование

- в revalidate fork() → деление ядра  
→ оп-р ядра (fork или clone)

! Р-м ядра и нест. безопасни изменение ядра.

• нест. безопасни fork() создается

новым процессом (ан. фунц. создания)

— процесс получает, который является  
подчиненным (не родителем!) процесса в том смысле,  
что:

- получает наследует один и то же,  
анон. маску, наследует свой регистров,  
распределение операторов рамки

! Для получения одн. анон. адр. исп. один и тот же (бут.)

Безопасное ур.-во (бут.) процесса соф. сообр.  
таким-то (UNIX-кодом) таким-то образом опр.;  
таким-то образом

В UNIX: к app. пр.-ho можно комбинировать как  
предок  $\Rightarrow$  new. комм  $\Rightarrow$  недороговато!

Постанову

UNIX Sys. file      UNIX BSD  
раздел от fork()

~~стеки~~      одинаковые exec. боятся fork()

UNIX system file

- file, поганка - сообр. файлы страниц  
но! дескриптор ссылается на страницы  
app. пр.-ho процесса-предка  
"поганок касается дескрипторов родительских  
процессов-поганок"

Реализация: 2 или аск. проц. нет. обрат. к  
app. пр.-ho предок - реагирует, т.к.  
новая единица имеет ввиду, а старая -  
всегда - к исходным данным (shareable  
данных)

Постанову права предка на app. пр.-ho:  
read-only и для copy-on-write.

$\Rightarrow$  Если сам предок или поганок non-чтм.  
 $\Rightarrow$  ~~если~~ не управляем доступа  
 $\Rightarrow$  обратим иска-я - одинаковые.  
copy-on-write  $\Rightarrow$  создает новые  
страницы файл, это поганка  
иска-я.

- current user
- current group
  - ↳ read-only

Указ - отмена для файла app. up-to-date  
 (force removes file, then creates it again.)

### UNIX BSD

- curr. boyob vfork()
- создает новый процесс, который забывает старший процесс (неподелен).
- сам процесс блокируется.

А если несколько подпроцессов?

Ситуация с copy-on-write, vfork() cycle.  
 некая инструкция не блокирует curr.  
 boyob exec() или exit().

### UNIX Sys 5

- open/copy.

↑ заверш. процесса

### UNIX BSD

- предок падает -  
 cycle

exit() - curr. boyob fails. exec.

return - boyob fails ->

exec() - curr. boyob, кот. переворот процесса

на

app. up-то программы, кот передаётся  
как аргумент.

⇒ нах. fork-ом из другой программы.

⇒ есть 2 способа UNIX для зап. программы:

- 1) fork()
- 2) exec()

Новый процесс - это то же fork().

!

# ps -al

# man ps

# ls -al

fork()

- процесс-методик - создается (нах-ся в отдельном процессе)
- процесс-прекорд

предки, даже потомок  
а не предок

⇒ UNIX поддерживает идентичные процессы.

Таблица - массив структур.

Регистрирование всего массива - деление  
памяти на ячейки.

- процесс job. записывается
- информации о нем - по порядку, без  
важности завершения (затраченное время)

! Нет синхронизаций, но сам процесс воспроизводится

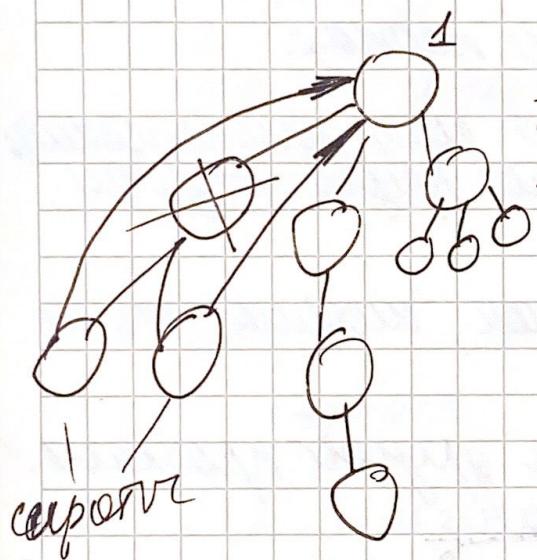
- синхронизацией

- free иект. — return integer

- Учтыв на дескриптор пр-кврка

\* p- cptr к процесс-адресам, т.е. посыпции

! В системе всегда 3 проц. с id=0 — проц., запущенный самим; id=1 — проц., открытый терминал.



Все процессы, зап. на машине терминал, id=1. Потомки проц. с id=1.

При зав-ии процесса в нейз. потомки его дархши получают

Такие процессы — супорук. их управляют терминалы.

- при завершении любого процесса система проверяет, не осталось ли у него незавершенные потомки

p- cptr - проверка

- если они остались, запускается проц. убий. пререк получит уб-ю на подсознательном уровне, а дополнительное действие

! Процесс в VNTX обходится в группу

Чтобы предотвратить зацикливание,  
т.е. wait (& status)

- блок. предотвращение зацикливания
- получает статус завершения процесса, состояния промежуточные.

Процесс просто так не создается.

В UNIX можно создать легко син. fork()

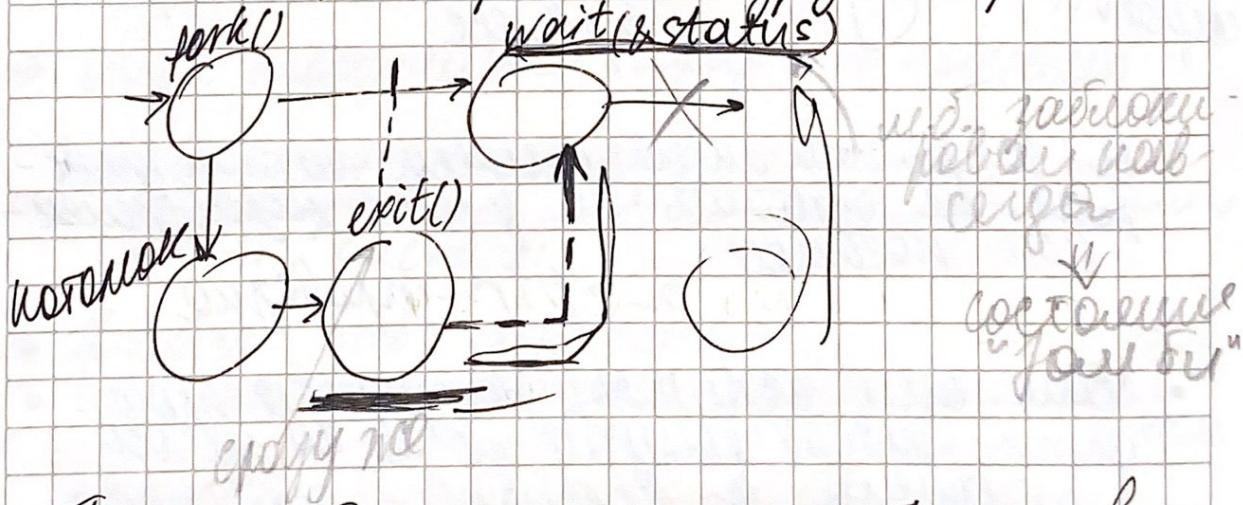
↳ ненорм. (пр. аномалии)

В Win - нужно же явн. о ненорм.

Система не контролирует кода приложения;  
не компр., ненорм. или пред. вызов wait().

exec() - вызывает новую, исчезнувшую не контролирует

Нормальное wait наведено к другому процессу.



Посл. замиг - идет. У ког. открыты все  
ресурсы, кроме которых в памяти.

Процесс (ресурс) заблокирован

В результате стат.

В рез. процесса получает наим. отрасль под-д.  
→ Определение констант

```
int main()
```

```
{  
    int childpid;  
    if ((childpid = fork()) == -1)  
        perror(" ");  
        exit(1);  
}
```

```
{  
    else if (childpid == 0)
```

```
    printf("Child: childpid=%d, parentpid=%d\n",  
           getpid(), getppid());
```

```
{  
    else
```

```
{ parent
```

```
{ return 0;
```

- При выполнении программы управляемые  
результаты определяются посредством  
установки соответствующих правил.

fork • возвращает childpid или parent  
childpid

execve • возвращает 0 при выполнении

openat + атрибуты файла  
user group others

- - exec gup.

..

super-user - акаунт (пользователь), который  
имеет доступ к системе и способен ее  
изменять (входить в него можно)

Блокировку в отладчике вызывает  
запрос на остановку.

kernel thread debugger

• мониторинговая функция  
реализует различные функции.

• докер-драйвер - осуществляющий  
рабочую работу для отладчика

to complete)

• Осн. работы докер-драйвера - управление памя-  
тью.

⇒ Система не контролирует памят

As soft iron

kworker

Системный грейд - запуск, исп. приложения  
на основе ядра Linux, ядро. ~~исп. би-  
блиотек~~. ~~исп. би-  
блиотек~~

mm - memory management

exit\_... - уничтожение страниц, дескт.

Основные ядра ядро, дескт.