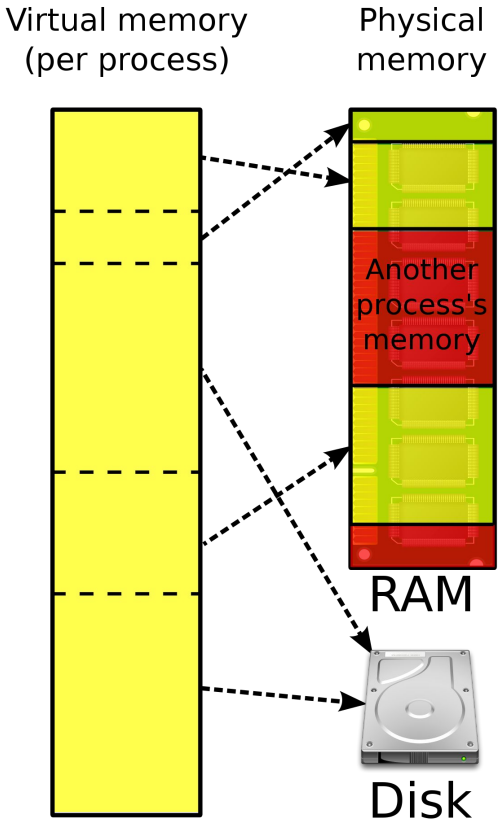
Программирование в Linux

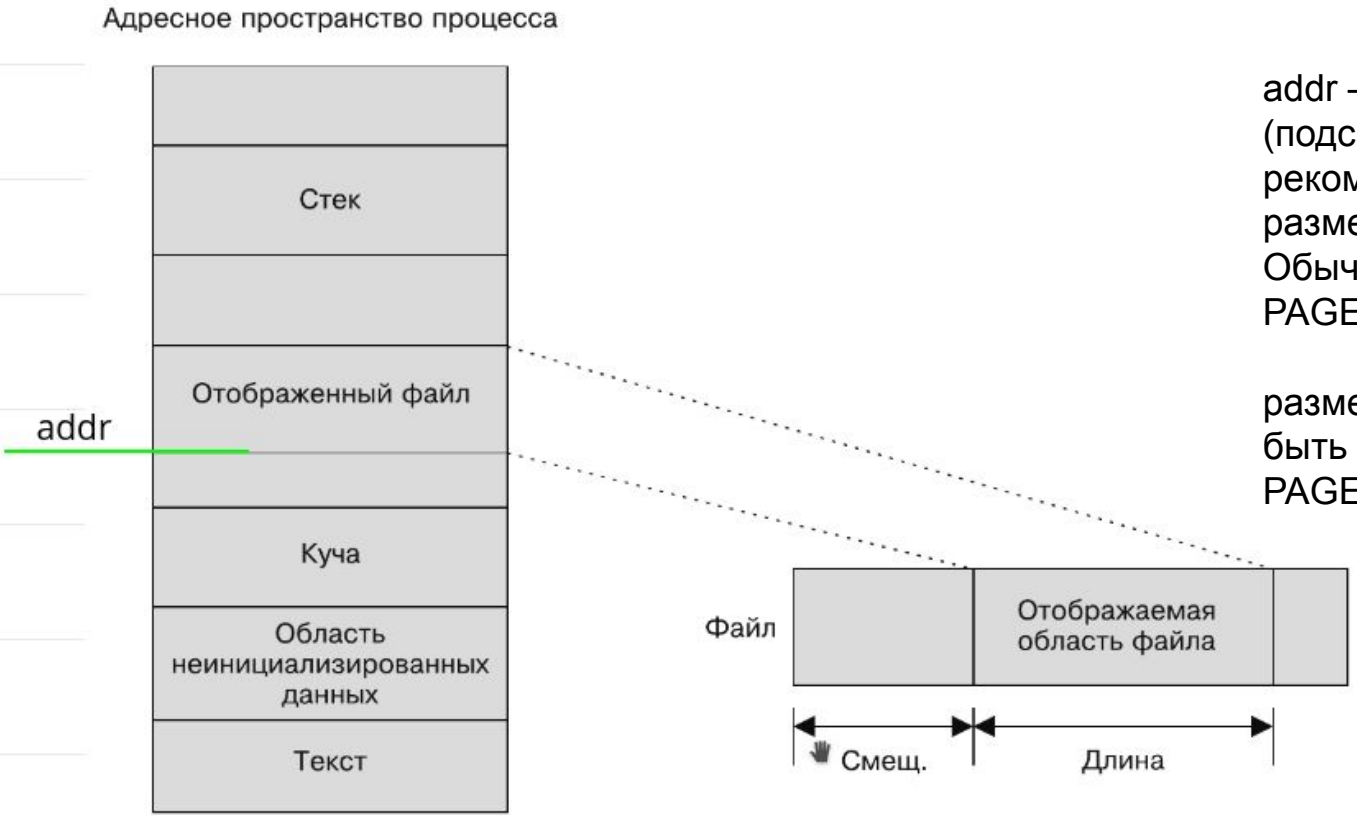
Файлы и IO

Отображение файлов в память



1. Ядро: отображение виртуальные страницы <-> физические — прописывает в спец. регистры CPU (адрес LUT)
2. Ядро: настраивает, что делать, если страница не найдена
   a. SIGSEGV
   b. подгрузка
3. Можно настроить отображение виртуальных страниц на подгрузку из определенного файла (mmap)

```c
void *mmap(void *addr, size_t length, int prot, int flags,
           int fd, off_t offset);
int munmap(void *addr, size_t length);
```



addr — опционален (подсказка, рекомендация по размещению). Обычно выровнен по PAGE_SIZE

размеры должны быть кратны PAGE_SIZE

boost::mapped_file — обертка

Анонимные отображения

Отображение файла /dev/zero — эквивалентно выделению памяти

Более удобно:

```
1 void* anon_memory = mmap(NULL,
2                           size,
3                           PROT_READ | PROT_WRITE,
4                           MAP_ANONYMOUS | MAP_PRIVATE,
5                           -1, // fd игнорируется
6                           0); // смещение тоже
```

Полученные страницы памяти будут заполнены нулями

Настройка прав доступа к отображениям

```c
#include <sys/mman.h>

int mprotect(void *addr, size_t len, int prot);

#define _GNU_SOURCE             /* See feature_test_macros(7) */
#include <sys/mman.h>

int pkey_mprotect(void *addr, size_t len, int prot, int pkey);
```

```c
int main(int argc, char *argv[])
{
    congifigure_signal_handler();
    int pagesize = sysconf(_SC_PAGE_SIZE);
    if (pagesize == -1)
        handle_error("sysconf");

    /* Allocate a buffer aligned on a page boundary;
       initial protection is PROT_READ | PROT_WRITE */

    char* buffer = memalign(pagesize, 4 * pagesize);
    if (buffer == NULL)
        handle_error("memalign");

    printf("Start of region:        0x%lx\n", (long) buffer);

    if (mprotect(buffer + pagesize * 2, pagesize,
                 PROT_READ) == -1)
        handle_error("mprotect");

    for (char* p = buffer ; ; )
        *(p++) = 'a';

    printf("Loop completed\n");     /* Should never happen */
    exit(EXIT_SUCCESS);
}
```

```c
#define handle_error(msg) \
    do { perror(msg); exit(EXIT_FAILURE); } while (0)

static void
handler(int sig, siginfo_t *si, void *unused) {
    printf("Got SIGSEGV at address: 0x%lx\n",
            (long) si->si_addr);
    exit(EXIT_FAILURE);
}

void congifigure_signal_handler(void) {
    struct sigaction sa;
    sa.sa_flags = SA_SIGINFO;
    sigemptyset(&sa.sa_mask);
    sa.sa_sigaction = handler;
    if (sigaction(SIGSEGV, &sa, NULL) == -1)
        handle_error("sigaction");
}
```

Специальные файлы

/dev/zero: read -> нули; write -> success, discarded

/dev/null: read -> EOF; write -> success discarded

/dev/full: read -> нули; write -> error (no space)

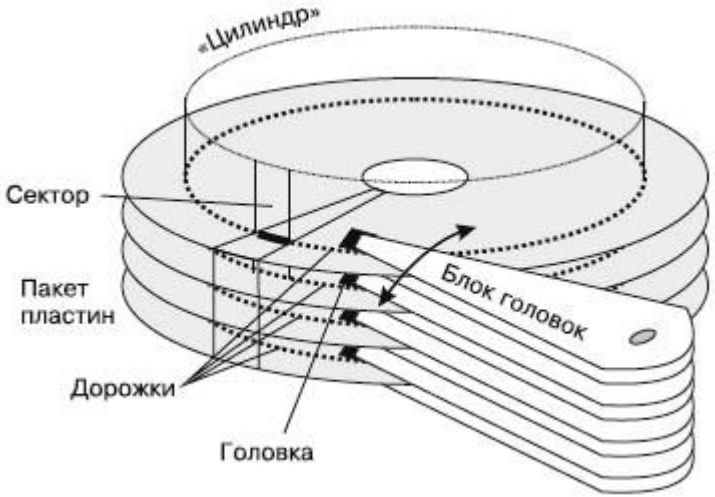
/dev/random, /dev/urandom — генераторы псевдослучайных чисел. random блокируется, если недостаточно энтропии

/dev/uinput — файл для имитации пользовательского ввода (мышь, клавиатура, джойстик...)

```c
int fd = open("/dev/uinput", O_WRONLY | O_NONBLOCK);

/* enable mouse button left and relative events */
ioctl(fd, UI_SET_EVBIT, EV_KEY);
ioctl(fd, UI_SET_KEYBIT, BTN_LEFT);

ioctl(fd, UI_SET_EVBIT, EV_REL);
ioctl(fd, UI_SET_RELBIT, REL_X);
ioctl(fd, UI_SET_RELBIT, REL_Y);

struct uinput_setup usetup;
memset(&usetup, 0, sizeof(usetup));
usetup.id.bustype = BUS_USB;
usetup.id.vendor = 0x1234; /* sample vendor */
usetup.id.product = 0x5678; /* sample product */
strcpy(usetup.name, "Example device");

ioctl(fd, UI_DEV_SETUP, &usetup);
ioctl(fd, UI_DEV_CREATE);

/*Give userspace some time to detect `new` device*/
sleep(1);

int i = 50;
/* Move the mouse diagonally, 5 units per axis */
while (i--) {
    emit(fd, EV_REL, REL_X, 5);
    emit(fd, EV_REL, REL_Y, 5);
    emit(fd, EV_SYN, SYN_REPORT, 0);
    usleep(15000);
}
/*
 * Give userspace some time to read the events before we destroy the
 * device with UI_DEV_DESTOY.
 */
sleep(1);

ioctl(fd, UI_DEV_DESTROY);
close(fd);
```

```c
void emit(int fd, int type, int code, int val)
{
    struct input_event ie;

    ie.type = type;
    ie.code = code;
    ie.value = val;
    /* timestamp values below are ignored */
    ie.time.tv_sec = 0;
    ie.time.tv_usec = 0;

    write(fd, &ie, sizeof(ie));
}
```

Планировщик IO



Скорость вращения 5400 - 15000 rpm

Произвольный доступ —> постоянное позиционирование головок

Время позиционирования: 4 - 12 ms!

Основные планировщики

У всех* — sort по позиции на устройстве и объединение смежных запросов

- NOOP
  - общий FIFO
  - не сортирует
  - пытается объединить смежные
  - хорош для SSD
- Deadline
  - Общая отсортированная очередь
  - Timeout устаревания запроса
  - + Две очереди (Read/Write) по времени устаревания
  - Если в голове очереди устаревший запрос — разбирай эту очередь
  - default на сегодняшний день
- CFQ (Completely fair queue)
  - блокирующие запросы — у каждого процесса своя очередь
  - асинхронные запросы — группируем по устройствам
  - Квант времени на очередь
  - очереди переключаются каруселью (Round Robin, RR)
- BFQ (Budget Fair Queue)
  - Модификация CFQ
  - RR учитывает размеры запросов

События файловой системы. inotify

```
4  int fd = inotify_init1(IN_NONBLOCK);
5  int wd = inotify_add_watch(fd, "/path/", IN_ALL_EVENTS);
6  ...
7  char buf[4096] __attribute__ ((aligned(__alignof__(struct inotify_event))));
8  int len = read(fd, buf, sizeof buf);
9
10  /* Loop over all events in the buffer */
11 for (char* ptr = buf; ptr < buf + len;
12      ptr += sizeof(struct inotify_event) + event->len) {
13      const struct inotify_event * event = (const struct inotify_event *) ptr;
14      // process events
15 }
16
```

```
struct inotify_event {
    int      wd;       /* Watch descriptor */
    uint32_t mask;     /* Mask describing event */
    uint32_t cookie;   /* Unique cookie associating related
                          events (for rename(2)) */
    uint32_t len;      /* Size of name field */
    char     name[];   /* Optional null-terminated name */
};
```

```
fd = open("dir/myfile", O_RDWR);
       Generates IN_OPEN events for both dir and dir/myfile.

read(fd, buf, count);
       Generates IN_ACCESS events for both dir and dir/myfile.

write(fd, buf, count);
       Generates IN_MODIFY events for both dir and dir/myfile.

fchmod(fd, mode);
       Generates IN_ATTRIB events for both dir and dir/myfile.

close(fd);
       Generates IN_CLOSE_WRITE events for both dir and dir/myfile.
```