Shared memory

Файлы отображаемые в память (memory mapped file)

- Файл или его часть отображаются непосредственно в адресное пространство процесса
- Содержимое файла можно читать просто обращаясь в оперативную память
- При изменении данных в памяти они могут быть сохранены в файле
- Момент сохранения в файле выбирается ядром, но можно им управлять msync

Системный вызов mmap

```
#include <sys/mman.h>
void *mmap(void *addr, size_t length, int prot,
           int flags, int fd, off_t offset);
```

- start – желаемый адрес подключения к адресному пространству
- length – размер подключаемого блока памяти
- prot – флаги: PROT_EXEC, PROT_READ, PROT_WRITE
- fd – файловый дескриптор (-1 в некоторых случаях)
- offset – смещение в файле

`int munmap(void *addr, size_t length)` - отключает отображение с адреса addr размера length

Системный вызов mmap (flags)

- **MAP_SHARED** – разделяемое отображение, изменения в памяти отображаются обратно в файл
- **MAP_PRIVATE** – неразделяемое отображение, copy-on-write
- **MAP_ANONYMOUS** – анонимное отображение (не соответствует никакому файлу)
- **MAP_FIXED** – не пытаться размещать отображение по адресу, отличному от start

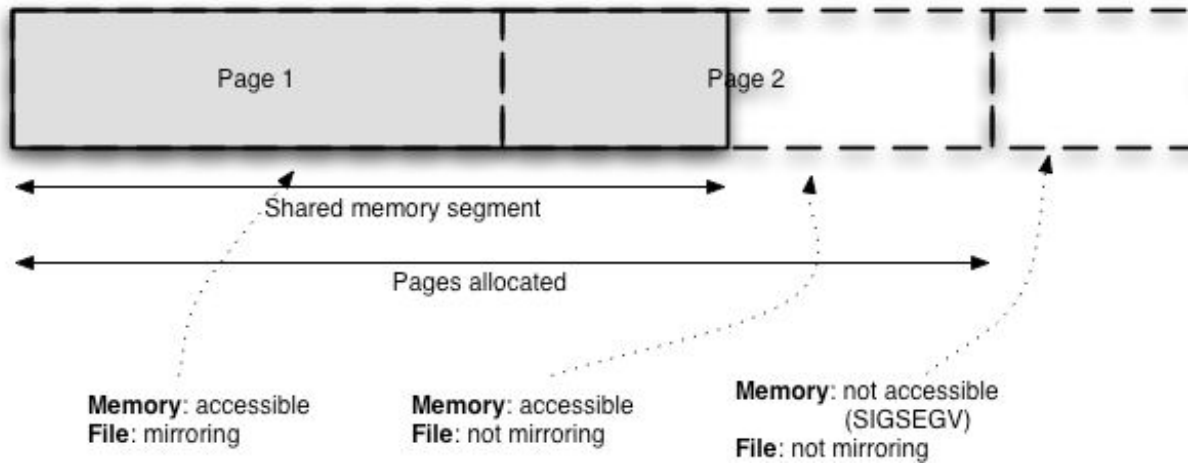
Особенности mmap

Гранулярность работы – одна страница памяти (x86 – 4KiB):

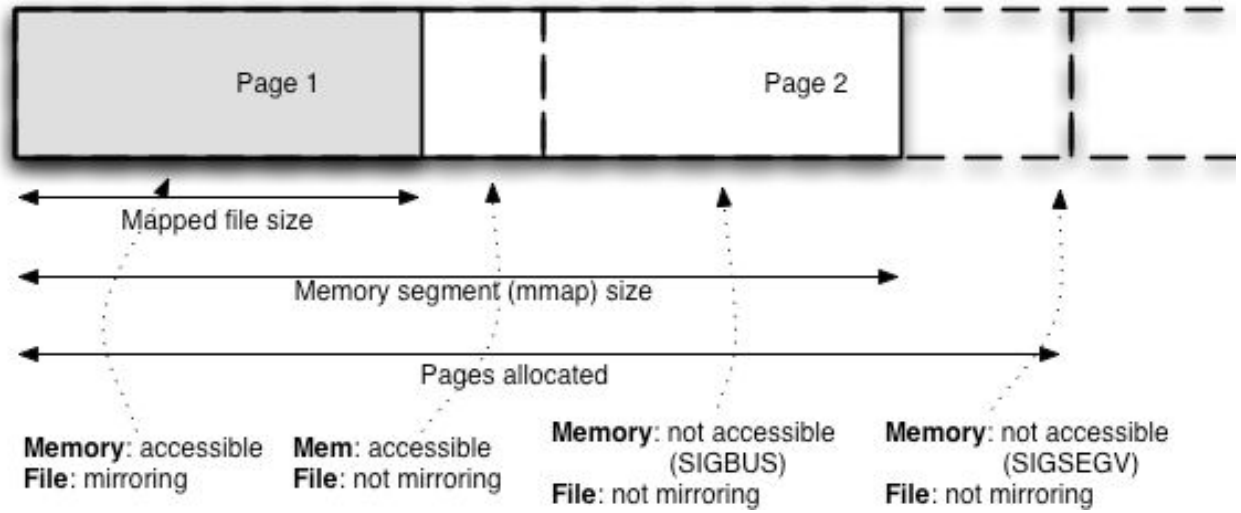
- Размер length должен быть кратен размеру страницы
- Смещение в файле offset должно быть кратно одной странице
- Файл не должен быть пустым

Хвост файла (< размера страницы) отображается на целую страницу, но размер не меняется

- Чтение данных после конца файла вернет 0
- Запись данных после конца файла не попадет в файл



memory segment size and mapped file size requested are equal, but not multiple of the page size



size of the mapped file is shorter than the size requested for the memory segment.

Типичное использование

MAP_SHARED – если несколько процессов отобразят файл, они будут видеть изменения друг друга, измененное содержимое будет сохранено в файле – реализация общей памяти (shared memory) процессов

MAP_PRIVATE – содержимое файла доступно для чтения, при модификации содержимого другие процессы не увидят изменений, они не будут сохранены в файле – отображение исполняемых файлов в память

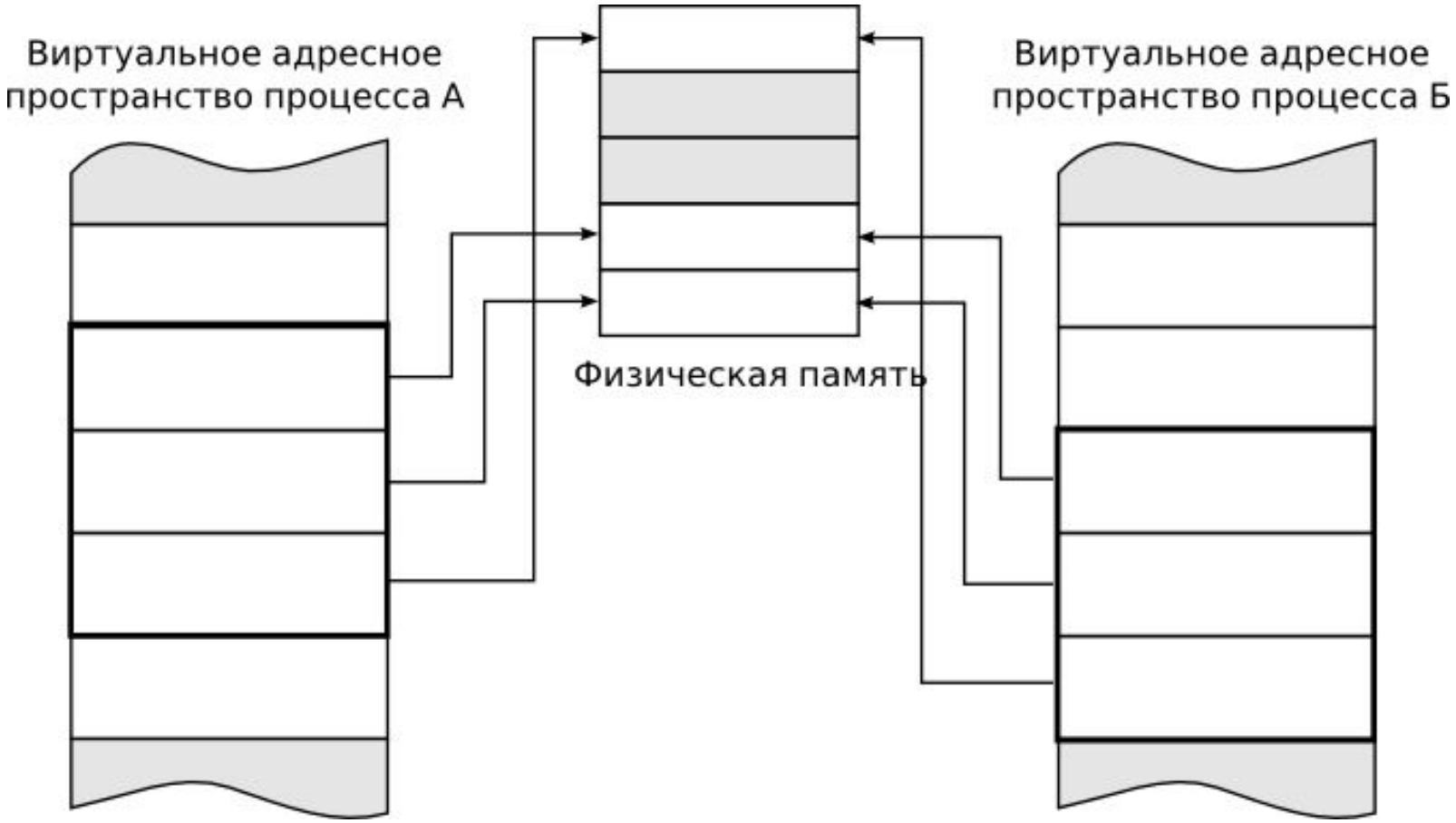
MAP_SHARED | **MAP_ANONYMOUS** – отображенная память доступна самому процессу и порожденным им процессам (они видят изменения) – реализация общей памяти для родственных процессов

MAP_PRIVATE | **MAP_ANONYMOUS** – содержимое памяти видимо только для одного процесса – дополнительная память в адресном пространстве процесса

MAP_PRIVATE

- Флаг MAP_PRIVATE в mmap – приватное отображение
- Изначально содержимое страницы берется из файла
- Но если страница модифицирована, то она “отвязывается” от файла
- Изменения модифицированных страниц обратно в файл не попадут

Концепция общей памяти



Процессы, выполняющие отображение одного и того же файла, разделяют физические страницы ОЗУ

Реализация общей памяти (System V)

- **key_t ftok(const char *pathname, int proj_id)**

```
ipckey = ftok("/tmp/foo", 42);
```

- **int shmget(key_t key, size_t size, int shmflg);**

```
shmid = shmget(ipckey, sizeof(int), IPC_CREAT|0666);
shmid = shmget(IPC_PRIVATE, sizeof(int), IPC_CREAT|0666);
```

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

```
shared = shmat(shmid, (void *) 0, 0);
```

- **int shmctl(int shmid, int cmd, struct shmid_ds *buf);**

```
shmctl(shmid, IPC_RMID, 0);
```

---

ipcs -m инфо о shared memory (-s о семафорах)
ipcrm удаление “забытых” регионов

Реализация общей памяти (POSIX)

- **int shm_open(const char *name, int oflag, mode_t mode);**

```
int fd = shm_open("mem", O_RDWR | O_CREAT, 0600);
```

- **int ftruncate(int fd, off_t length);**

```
ftruncate(fd, len);
char* mem = mmap(NULL, len, PROT_READ | PROT_WRITE,
                        MAP_SHARED, fd, 0);
munmap(mem, len);
```

- **int shm_unlink(const char *name);**

Семафоры POSIX

Именованные (для процессов):

```
sem_t *sem_open(const char *name, int oflag, mode_t mode,
                  unsigned int value);
int sem_close(sem_t *sem);
int sem_unlink(const char *name);
```

Неименованные (для потоков):

```
int sem_init(sem_t *sem, int pshared, unsigned int value);
int sem_destroy(sem_t *sem);
```

Общие операции:

```
int sem_post(sem_t *sem); (+1)
int sem_wait(sem_t *sem); (-1)
```

(могут быть использованы для процессов, если shared>=1 и отображены в общую память)