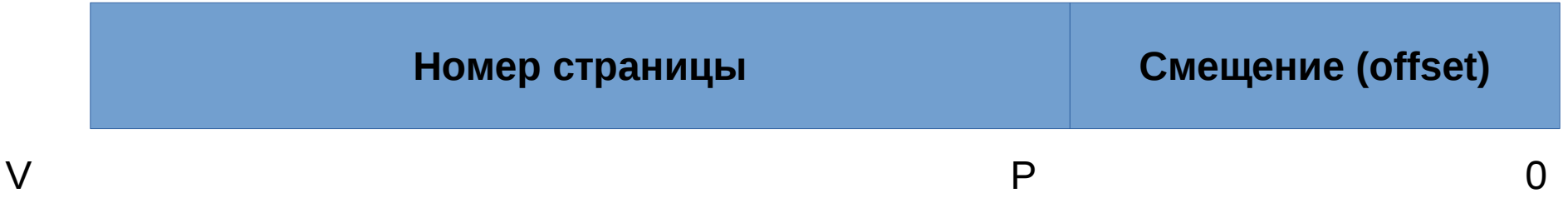
Лекция 20
Управление памятью

Страничная виртуальная память

- Виртуальное адресное пространство процесса разбивается на страницы
- Страницы имеют фиксированный размер (обычно на x86/x64 – 4KiB)
- Виртуальный адрес состоит из двух частей: номер страницы и смещение в странице

Виртуальный адрес



- V — количество бит виртуального адреса
- P — количество бит на смещение в страницу
- (V - P) — количество бит на номер страницы
- Для x86: V = 32, P = 12, V-P = 20
  - Виртуальное адресное пространство 4GiB
  - Размер страницы — 4096 байт (4KiB)
  - 2^20 (~1 Mi виртуальных страниц)

Физическая память

- Физическая память (ОЗУ) разбивается на физические страницы (page frames)
- Каждая виртуальная страница может быть отображена в некоторую физическую страницу
- Несколько виртуальных страниц разных процессов могут отображаться в одну и ту же физическую страницу
- Но виртуальная страница может никуда не отображаться

Demand paging

- Логическое отображение – то, как должно быть (/proc/self/maps)
- Физическое отображение – то, как есть на самом деле (/proc/self/pagemap)
- Если страница есть в логическом отображении, но нет в физическом, то при первом обращении к этой странице ядро выделит новую физическую страницу ОЗУ или возьмет существующую и добавит ее в физическое отображение

Demand paging

- Процесс начинает работу с настроенным логическим отображением и пустым физическим отображением (см. VmVSZ)
- Постепенно по мере обращения к страницам заполняется физическое отображение (см. VmRSS)
- Если к странице не было обращений, она не будет загружена в физическую память (ОЗУ)

Страничная подкачка

- Физические страницы – ценный ресурс, в какой-то момент их может не хватить
- Ядро попытается освободить физические страницы для выполнения текущего запроса
- Если физическая страница соответствует отображению файла в память и не модифицировалась, она просто освобождается
- Страницы MAP_SHARED и модифицированные (dirty) сохраняются в файл и освобождаются
- Прочие страницы сохраняются в файл (раздел) страничной подкачки – swap file: стек, куча и т. п.

Типы страниц в памяти

- Выгружаемые (страница может быть выгружена в область подкачки)
- Невыгружаемые (locked) — должны находиться в ОЗУ
- Процесс может пометить часть страниц как невыгружаемые (системный вызов mlock)
- Непривилегированный — макс. 32 KiB
- Все страницы ядра — невыгружаемые

Резервирование swap

- Место в файле подкачки может быть зарезервировано при создании страницы, которую **может быть** потребуется сохранить в swap
  - Стек, куча
  - Все файлы, отображаемые в память с MAP_PRIVATE (т. е. исполняемые файлы и библиотеки) для каждого процесса
- В Linux место в файле подкачки выделяется при сохранении страницы в файле подкачки
- Возможны ситуации overcommit memory

Расположение виртуальной страницы

- В физической памяти (ОЗУ) – после первого обращения к ней и пока она не выгружена
- В файле (при отображении файла в память) – подгрузится в ОЗУ при обращении к ней
- В области подкачки (swap file) – подгрузится обратно в ОЗУ при обращении к ней
- НИГДЕ – будет выделена в ОЗУ при обращении к ней (overcommited pages)

MAP_PRIVATE

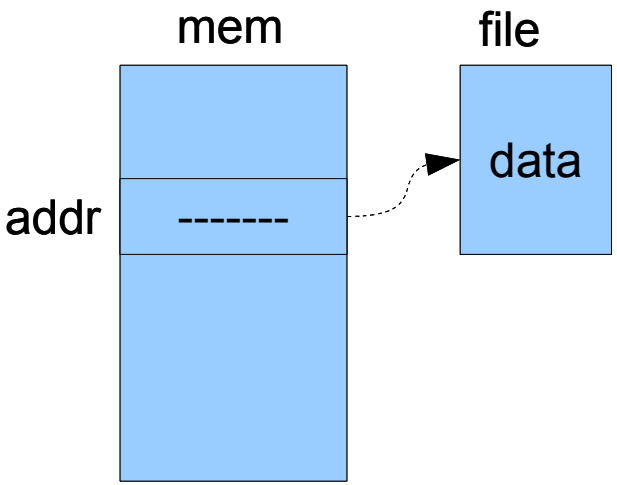
- Флаг MAP_PRIVATE в mmap – приватное отображение
- Изначально содержимое страницы берется из файла
- Но если страница модифицирована, то она “отвязывается” от файла
- Изменения модифицированных страниц обратно в файл не попадут

Copy-on-write

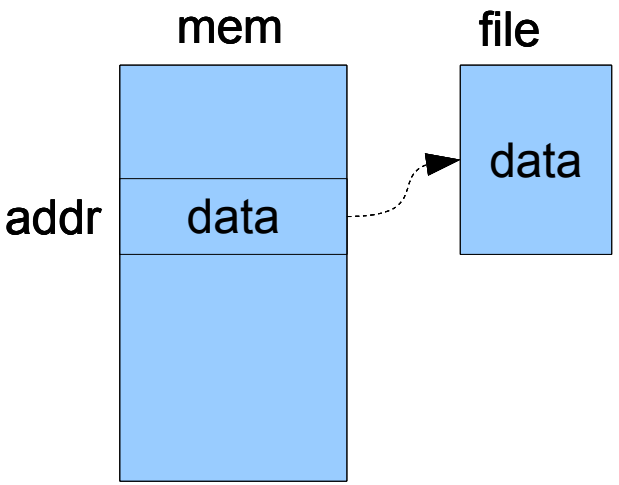
- Механизм оптимизации копирования страниц
- При обычном механизме копия страницы в физической памяти создается немедленно
- При механизме copy-on-write создание копии страницы откладывается до первой записи в страницу

Copy-on-write

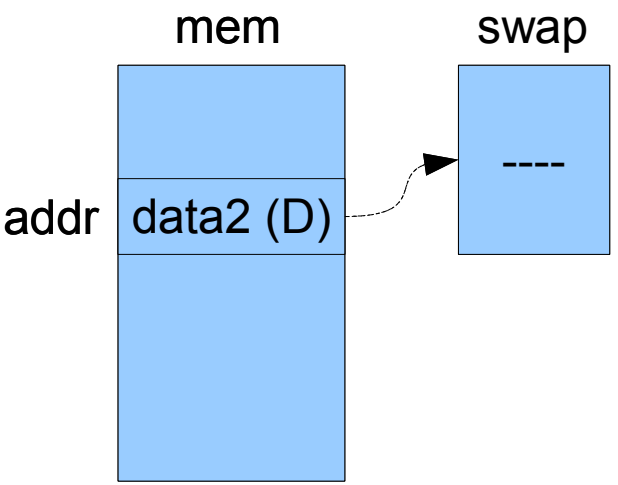
```
fd = open("file", O_RDWR, 0);
addr = mmap(0, 4096, PROT_READ|PROT_WRITE,MAP_PRIVATE,fd, 0);
```



При создании отображения страница в памяти помечена как отсутствующая, но отображенная на соответствующий файл



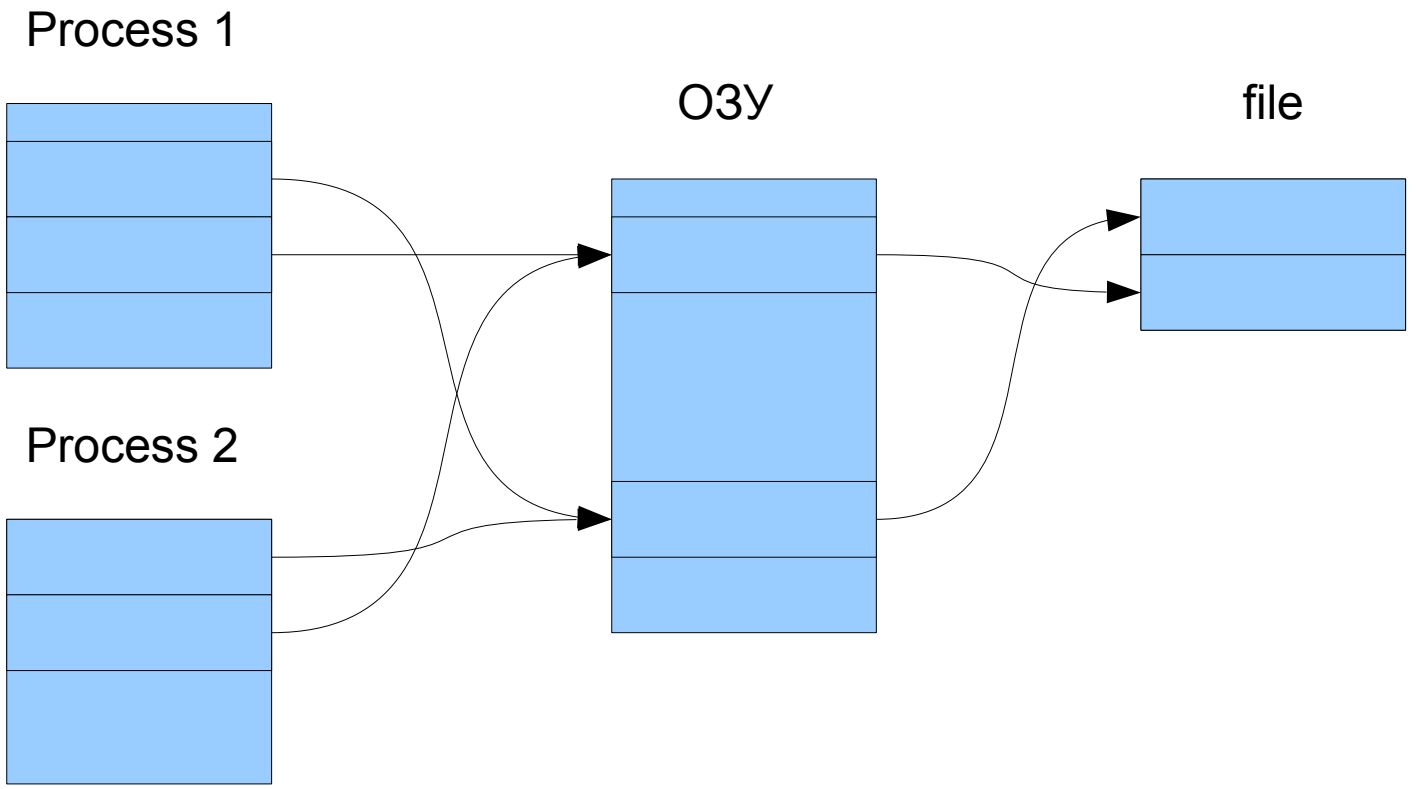
При чтении содержимое страницы подгружается из файла, страница помечается как «только для чтения»



При записи в страницу выделяется место в области подкачки, при необходимости создается копия страницы в ОЗУ, отображение переключается на swap

Разделение страниц между процессами

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



Необеспеченная память (memory overcommit)

- Стратегия выделения copy-on-write и выделение памяти по требованию приводят к тому, что хотя страница присутствует в логическом отображении, невозможно настроить физическое отображение (нет свободных физических страниц, исчерпан swap)
- Надо попытаться удовлетворить запрос этого процесса за счет других процессов
- Необходимо снять с выполнения какой-нибудь процесс и таким образом освободить память (OOM killer)

OOM Killer

- Задача: выбрать минимальное число процессов, чтобы освободить максимальный объем памяти, но нанести минимальный ущерб системе
- Для каждого процесса вычисляется oom_score (/proc/${PID}/oom_score)
  - Чем больше RSS и Swap usage — тем хуже
  - Привилегированные процессы лучше обычных
  - Пользователь может задать поправку: /proc/${PID}/oom_score_adj

Загрузка файла на выполнение

- ELF-файл имеет структуру, оптимизированную для отображения файла mmap
- Секция кода (.text) отображается PROT_READ | PROT_EXECUTE, MAP_PRIVATE
- Константные данные (.rodata): PROT_READ, MAP_PRIVATE
- Данные (.data): PROT_READ | PROT_WRITE, MAP_PRIVATE
- Секции .text и .rodata у всех процессов, запущенных из одного файла, будут использовать одни и те же физические страницы памяти

Разделяемые библиотеки

- Позволяют избежать дублирования кода в процессах (например, все процессы имеют общую реализацию printf)
- Делает возможным разделять код библиотек между процессами разных исполняемых файлов (при статической компоновке реализация printf может располагаться по разным адресам, что делает невозможным разделение)
- Облегчают обновление ПО

Загрузка разделяемых библиотек

- ELF-файл содержит секцию .interp. Эта секция содержит путь к «интерпретатору» - /lib/ld-linux.so.2 — загрузчик динамических библиотек
- Загрузчик проходит по списку зависимостей библиотек, находит их в файловой системе и загружает в память, рекурсивно, пока все зависимости не будут удовлетворены
- Загрузка каждой библиотеки аналогична загрузке исполняемого файла (mmap)
- Но! Одна и та же библиотека может быть загружена в разных процессах по разным адресам

Позиционно-независимый код

- В разделяемой библиотеке секция кода позиционно-независима, то есть страницы, занимаемые кодом, идентичны независимо от их виртуального адреса в каждом процессе
- Требуется одна копия кода в страницах физической памяти, на которую будут отображаться страницы виртуальной памяти разных процессов
- Секции разделяемой библиотеки, индивидуальные для каждого процесса (GOT, .data), малы по сравнению с секцией кода
- Огромная экономия физической памяти!

Работа с SO-файлами

```
#include <dlfcn.h>


void *dlopen(const char *filename, int flag);
void *dlsym(void *handle, const char
*symbol);
int dlclose(void *handle);
```

gcc prog.c -o prog -ldl

Компиляция SO-файла

- gcc -fPIC -DPIC -shared plugin.c -o plugin.so
  - -fPIC – генерация позиционно-независимого кода
  - -DPIC – дополнительный флаг для условной компиляции
  - -shared – результат компиляции – SO-файл

Компиляция главной программы

- gcc -rdynamic program.c -o program -ldl
  - -rdynamic – сгенерировать таблицу видимых символов в исполняемый файл, чтобы из подгружаемых SO-файлов можно было вызывать функции основной программы
  - -ldl – библиотека функций (ldopen и др.)