Лекции 23-24
Управление памятью

Управление памятью

- С точки зрения процесса
  - Адресное пространство процесса
  - Управление адресным пространством
  - Отображаемые файлы в память
  - Динамические (разделяемые) библиотеки
- С точки зрения ядра
  - Управление виртуальной адресацией
  - Управление страничным/буферным кешем

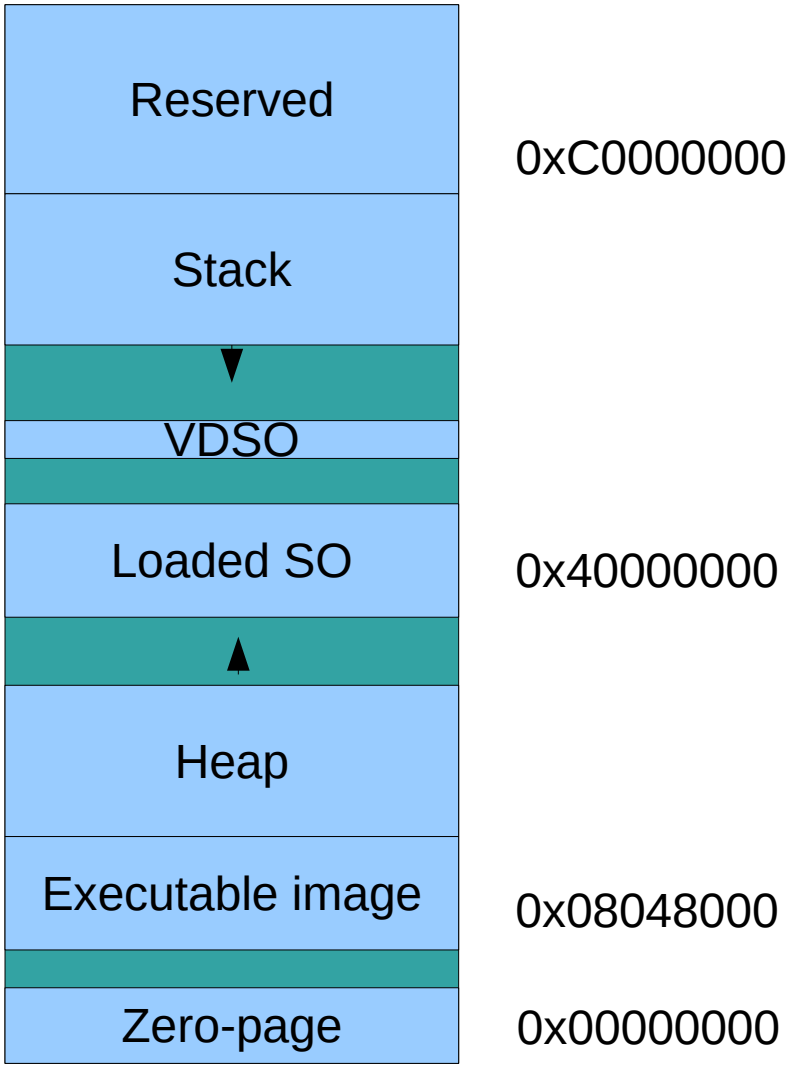
Адресное пространство процесса

- Каждый процесс работает в своем изолированном виртуальном адресном пространстве
- Иллюзия того, что процесс монопольно владеет всей памятью
- Пример: x86 — 32-битное адресное пространство, 2^32 = 4GiB
- X64 — 48-битное адресное пространство, 2^48 = 256 TiB
- Процессор x86 в 32-битном режиме может работать с > 4GiB ОЗУ, но не более 4GiB на процесс

Адресное пространство x86

- Указатели — 32-битные
- Диапазон адресов: 0x00000000 — 0xffffffff
- Как правило, ОС не дает использовать все 4 GiB:
  - Linux: 3GiB доступны, 1GiB зарезервирован
  - Win32: 2GiB / 2GiB
- Попытка обращения в зарезерв. область — segmentation fault
- В зарезервированный 1GiB (не доступный из user-space) каждого процесса отображается память ядра — ускорение переключения user->kernel

Адресное пространство процесса



- Нулевая страница — защита от обращений по указателю NULL
- Стек расширяется вниз автоматически
- Куча растет вверх по запросу
- Текущее состояние карты памяти: /proc/${PID}/maps

Адресное пространство

- VDSO — спец. разделяемая библиотека — ускорение частых системных вызовов (time, gettimeofday, etc)
- Исполняемый образ — ELF-файл, отображенный на память. Состоит из секций:
  - .text — секция кода, read-only, executable — содержит инструкции программы и константные данные
  - .data — секция данных, read-write
  - .bss — секция данных, инициализированных 0
- Каждый SO-файл (разделяемая библиотека) — ELF-файл, отображаемый в память

/proc/${PID}/maps

45e55000-45e74000 r-xp 00000000 08:02 1508434 /usr/lib/ld-2.17.so

45e74000-45e75000 r--p 0001e000 08:02 1508434 /usr/lib/ld-2.17.so

- Диапазон виртуальных адресов отображения
- Права: rwx, p — private COW mapping, s — shared
- Смещение в файле
- Major:Minor Inode
- Путь к файлу

/proc/${PID}/status

- Статистика работы процесса, в т. ч. по памяти

```
VmPeak:     4300 kB // макс. Размер VM
VmSize:     4300 kB // текущий размер VM
VmLck:         0 kB // locked in memory
VmPin:         0 kB // pinned in memory
VmHWM:       456 kB // макс. RSS
VmRSS:       456 kB // resident set size
VmData:      156 kB // размер данных
VmStk:       136 kB // размер стека
VmExe:        48 kB // размер исп. файла
VmLib:      1884 kB // размер SO-библиотек
VmPTE:        24 kB // размер таблиц страниц
VmSwap:        0 kB // использование swap
```

Статистика использования памяти

- Virtual Memory Size — суммарный размер отображенных страниц виртуальной памяти
- Resident Set Size — размер страниц, находящихся в оперативной памяти
- Страницы могут находиться:
  - В ОЗУ
  - В swap-файле
  - В файле (исполняемого файла или SO)
  - Нигде (overcommit)

Ограничения адресного пространства

- Команда ulimit — установка ограничений процесса

```
core file size          (blocks, -c) 0
data seg size           (kbytes, -d) unlimited
scheduling priority             (-e) 0
file size               (blocks, -f) unlimited
pending signals                 (-i) 57326
max locked memory       (kbytes, -l) 32
max memory size         (kbytes, -m) unlimited
open files                      (-n) 1024
pipe size            (512 bytes, -p) 8
POSIX message queues     (bytes, -q) 819200
real-time priority              (-r) 0
stack size              (kbytes, -s) 8192
cpu time               (seconds, -t) unlimited
max user processes              (-u) 1024
virtual memory          (kbytes, -v) unlimited
file locks                      (-x) unlimited
```

Ограничения адресного пространства

- Системные вызовы setrlimit/getrlimit
- Жесткий лимит (hard limit) — нельзя превышать
- Мягкий лимит (soft limit) — процесс может увеличивать и уменьшать
- RLIMIT_AS — лимит адресного пространства
- RLIMIT_STACK — лимит размера стека

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

Управление адресным пространством процесса

- Системный вызов sbrk() - изменить адрес конца сегмента данных

```
void  *sbrk(intptr_t increment);
```

- Сразу после загрузки исполняемого образа break addess — это конец сегмента данных
- sbrk возвращает предыдущее значение

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 *start, 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

Системный вызов munmap

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

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
- MAP_NORESERVE — не резервировать область подкачки (для отображений, допускающих запись)

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

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

Типичное использование

- 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

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

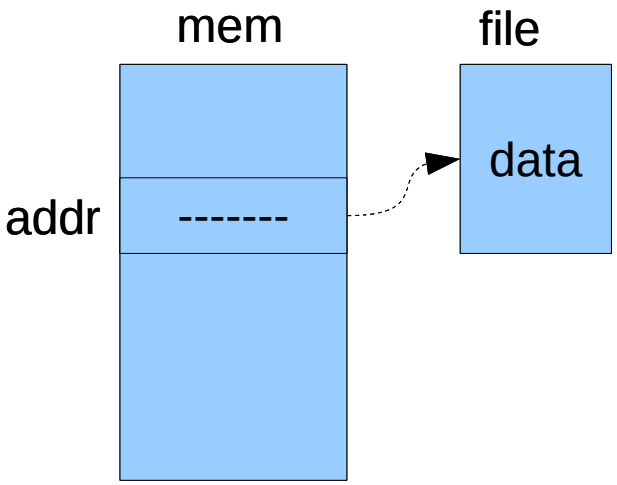
- Флаг MAP_PRIVATE в mmap – приватное отображение
- Изначально содержимое страницы берется из файла
- Но если страница модифицирована, то она “отвязывается” от файла
- Изменения модифицированных страниц обратно в файл не попадут

Copy-on-write

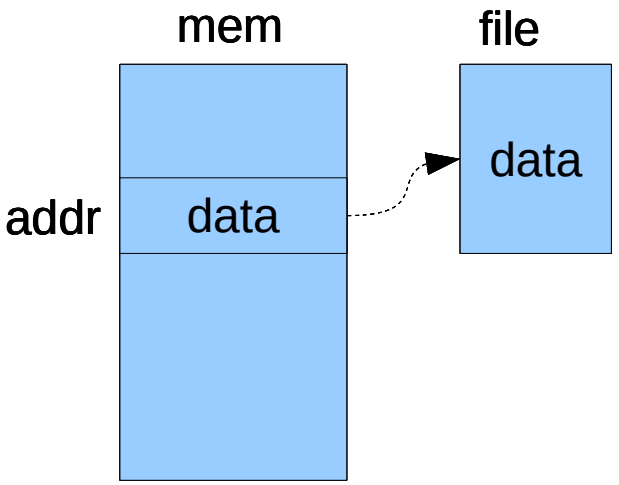
- Механизм оптимизации копирования страниц
- При обычном механизме копия страницы в физической памяти создается немедленно
- При механизме copy-on-write создание копии страницы откладывается до первой записи в страницу

Copy-on-write

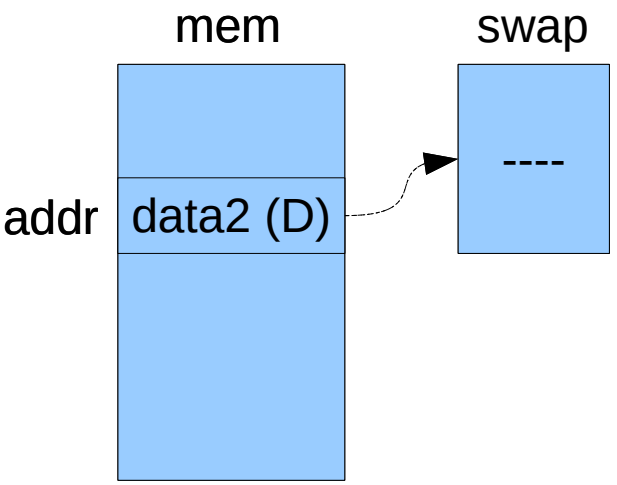
```
fd = open("file", O_RDWR, 0);
addr = mmap(0, 4096, PROT_READ|PROT_WRITE,MAP_PRIVATE,fd, 0);
```



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



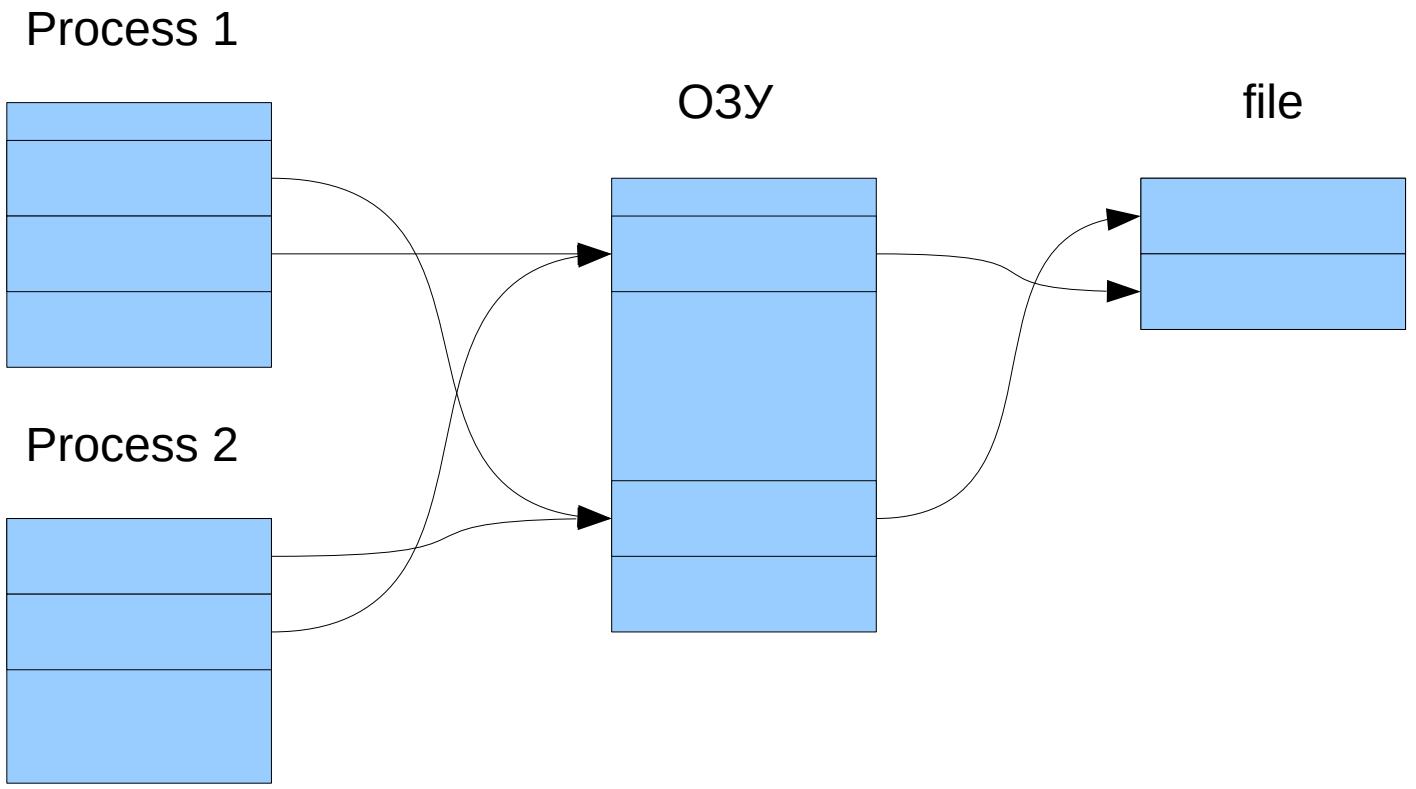
При чтении содержимое страницы подгружается из файла, страница помечается как «только для чтения»



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!