Лекция 7
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

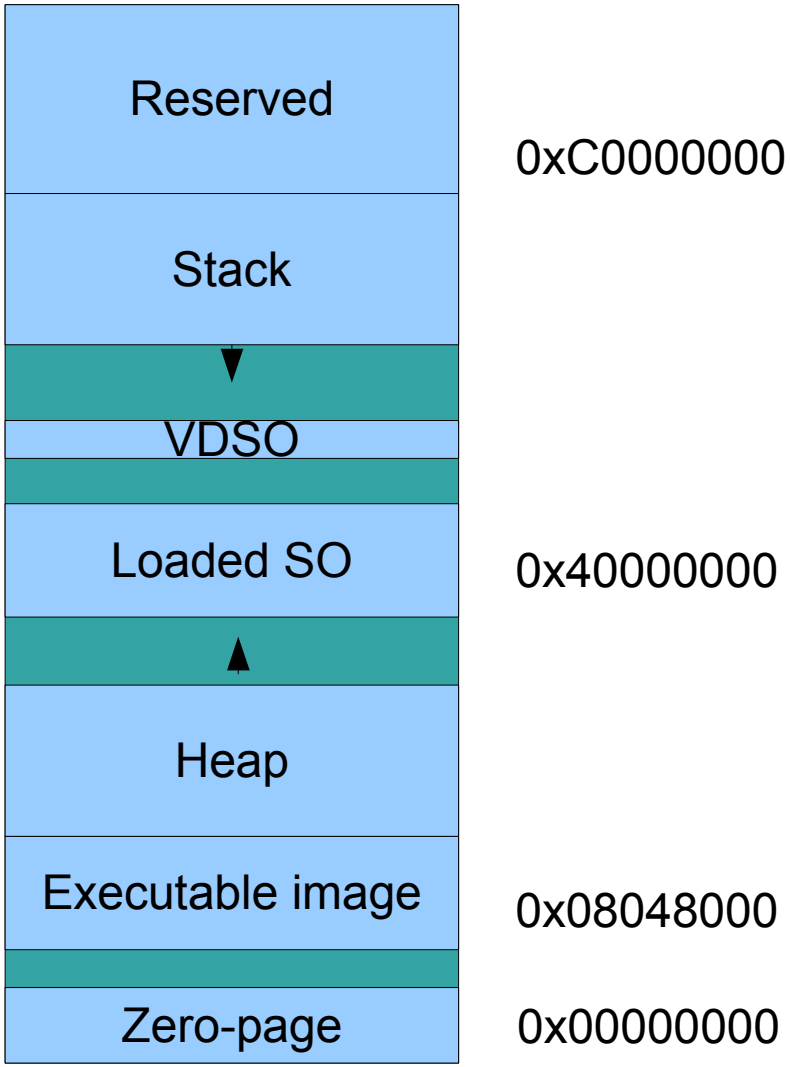
Адресное пространство процесса

- 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:

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

Управление динамической памятью (кучей)

- Обеспечить работу функций malloc, calloc, free, realloc, new, new[], delete, delete[]
- Память может запрашиваться фрагментами произвольного размера
- Память может освобождаться в произвольный момент времени
- Стандартные стратегии обслуживания (стек, очередь) неприменимы

Управление кучей

- Память может запрашиваться и освобождаться в нескольких нитях одновременно
- К структурам данных предъявляются разные требования по выравниванию, поэтому данные выравниваются по максимально жесткому требованию (8 байт)
-
- **Выделение памяти в куче намного медленнее, чем в стеке!**

Проблемы динамической памяти

- Необходимость блокировки в многонитевых программах
- Постепенное дробление больших непрерывных фрагментов памяти на маленькие
- Постепенная фрагментация динамической памяти
- Постепенный рост размера виртуального адресного пространства и невозможность освобождения ни одной страницы

Фрагментация

p1 = malloc(4); p2 = malloc(4);

free(p1); p3 = malloc(2); p4 = malloc(1);

free(p2); p5 = malloc(2); p6 = malloc(1);

free(p3); free(p5);

Запрос памяти у ядра

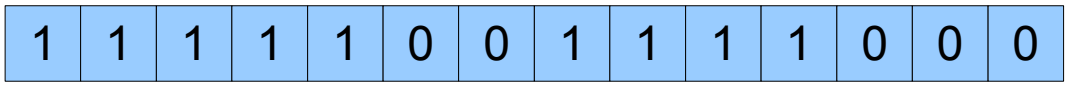
- Когда при очередном вызове функции выделения памяти запрос не может быть удовлетворен, с помощью sbrk запрашивается порция памяти у ядра
- Как правило, память не возвращается ядру, даже если это возможно

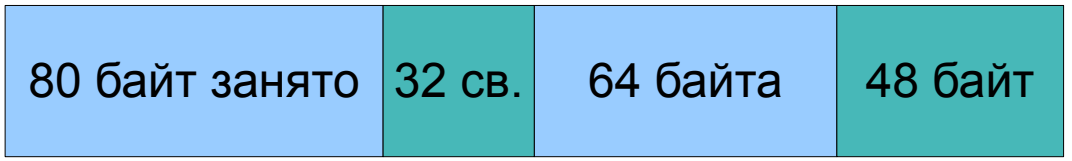
Стратегии распределения

- Битовый массив блоков
- Списки свободных/занятых блоков

Битовый массив блоков

- Память разбивается на блоки выделения фиксированного размера (например, 16 байт)
- Каждому блоку выделения ставится в соответствие 1 бит в битовом массиве: 0 — блок свободен, 1 - занят





Списки блоков

- Вариант с одиним списком:
  - В памяти поддерживается двусвязный список выделенных/свободных блоков
  - Структура дескриптора блока:

```
struct memdesc {
    struct memdesc *prev;
    size_t size;
    unsigned char data[0];
};
```

  - Флаг свободный/занятый - младший бит size

Выделение памяти malloc

- Размер выделения округляется вверх до размера выравнивания (8 байт), 0 байт → 8
- В памяти ищется подходящий свободный блок
- Если свободного блока нет, запрашивается блок памяти с помощью sbrk() и помечается как свободный
- Найденный блок при необходимости дробится, формируется дескриптор блока памяти и возвращается указатель на поле data

Освобождение памяти free

- Из переданного указателя ptr вычитается 8, таким образом получаем указатель на дескриптор блока памяти
- Блок памяти помечается как свободный, при необходимости сливается с непосредственно предшествующим и/или следующим свободным блоком

Алгоритмы выделения блоков

- Первый подходящий
- Самый подходящий
- Быстрый подходящий: поддерживаются список свободных блоков наиболее часто запрашиваемых размеров
-
- Существует много различных алгоритмов управления динамической памятью для разных ситуаций, нет однозначно наилучшего

Ошибки работы с динамической памятью

- Типичные ошибки:
  - Memory overrun (выход за положительную границу)
  - Memory underun (выход за 0)
  - Use after free
  - Double free
  - Free of non-allocated pointer
- Приводят к порче списков свободных блоков и падению программы в какой-то момент позже
- Программа valgrind — отладчик работы с динамической памятью