Управление памятью ядром

Лекция №8 АКОС ФИВТ ПМИ

Физическое адресное пространство

- Процессор адресует 2^{32}, 2^{36} (Intel) или 2^{64} байт на шине адреса
- К шине адреса подключается как оперативная память, так и перефирийные схемы

Адресное пространство процесса

- Каждый процесс (программа) работает в своем изолированном адресном пространстве
- Адресация начинается с 0
- Для 32-битный систем: 4Гб
- Для 64-битных систем - только 48 бит - 256Тб

Адресное пространство процесса (IA-32 - x86)

- Диапазон адресов `0x00000000` `0xffffffff`
- Доступно 3Гб (Linux) или 2Гб (Windows)
- В верхней части - область ядра

`0xc0000000`	Пространство ядра
`0xbf800000`	Стек
	Библиотеки, VDSO
	Куча
`0x004d9000`	Программа
`0x00000000`	Заполнено нулями

Адресное пространство процесса

- Программы и библиотеки состоят из отдельных секций: .text, .data, .ro_data
- Стек растёт вниз
- Куча растёт вверх
- VDSO - небольшое отображение области ядра на пользовательское пространство

Адресное пространство процесса

- Программы и библиотеки состоят из отдельных секций: .text, .data, .ro_data

Можно сэкономить физическую память при запуске нескольких экземпляров одной программы, или разных программ, использующих одинаковые библиотеки

/proc/номер_процесса/maps

- Текстовый файл в виртуальной файловой системе procfs
- Подкаталог self - ссылка на текущий процесс

```
cat /proc/self/maps
```

/proc/номер_процесса/maps

```
004d9000-004e1000 r-xp 00000000 08:02 655669     /usr/bin/cat
004e1000-004e2000 r--p 00007000 08:02 655669     /usr/bin/cat
004e2000-004e3000 rw-p 00008000 08:02 655669     /usr/bin/cat
|- адрес в памяти |    |        |- st_dev |      |- имя файла
                  |    |                  |
   права доступа -|    |           inode -|
                       |
              смещение -|
```

Модели адресации

- Сегментная (реальный режим процессора)
- Страничная 2-х уровневая (x86)
- Страничная 3-х уровневая (x86+PAE)
- Страничная 4-х уровневая (x86_64)

Сегментная модель

- Регистры используются в 16-битном виде
- Необходимо адресовать 1Мб адресного пространства
- Виды сегментов:
  - сегмент кода (CS)
  - сегмент стека (SS)
  - сегмент данных (DS)
- Addr = (Segment << 4) + Offset

Страничная модель

- Вся память делится на страницы одинакового размера
- У страниц есть набор аттрибутов, в том числе права доступа

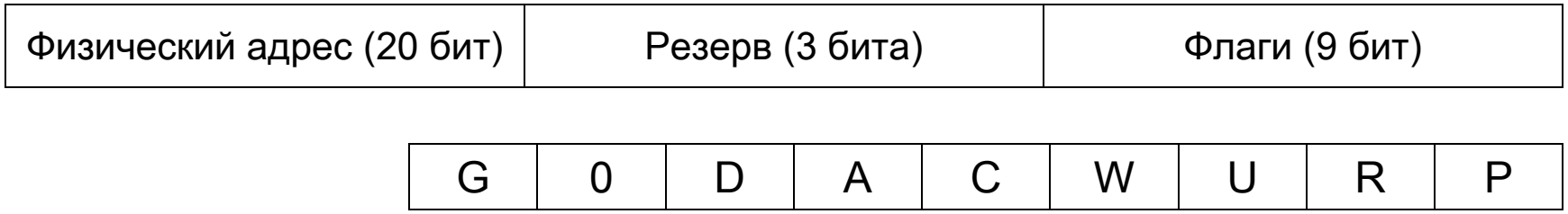
Страничная адресация x86

Номер в каталоге (10 бит)	Номер в таблиц (10 бит)	Смещение (12 бит)

- Регистр CR2 указывает на адрес каталога страниц
- Каждая запись каталога - адрес начала таблицы
- Таблицы содержат записи о страницах

- Для 32-битных систем:
  - –размер страницы 4К
  - –в каждой таблице 1024 записи
  - –в каждом каталоге 1024 записи

Страничная адресация x86



- G (global) - страница глобальная
- D (dirty) - страница была модицифирована
- A (accessed) - был доступ к странице
- C (caching) - запрещено кеширование
- W (write-throught) - разрешена сквозная запись
- U (userspace) - есть доступ обычному процессу
- R (read+write) - есть права на запись
- P (present) - страница находится в физической памяти

Page Fault

- Ситуация, когда P==0
- Генерируется исключительная ситуация процессором
- Не обязательно ошибка:
  - для обычной памяти страница может быть в swap'e
  - для отображаемого файла - данные на буферизованы
  - запись в страницу для Copy-On-Write
- Ядро обрабатывает исключение

Поддержка со стороны процессора

- Модуль MMU (Memory Management Unit) выполняет вычисления адресов по таблицам
- Кеш TLB (Translation Lookaside Buffer) хранит таблицы памяти для текущего процесса

Пространство ядра

- Некоторые страницы имеют флаг U==0 (верхний 1Гб для 32-битных систем)
- Данные недоступны из обычного режима
- Но доступны из режима ядра
- Системный вызов через sysenter/syscall не перезагружает TLB, а только переключает режим

Выделение памяти

- C++: операторы new и new[] - используют malloc/calloc
- Си: **функции** malloc/calloc выделяют память на куче
- malloc/calloc/free - это функции, а не системные вызовы!
- В разных реализациях libc реализации не совместимы (проблема обычно заметна в Windows)

Выделение памяти на куче

- Системный вызов brk (legacy, но очень простой для понимания способ)
- Системный вызов mmap - выделение некоторой области памяти, кратной количеству страниц

mmap

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

- addr - рекомендуемый адрес размещения
- length - размер, кратный размеру страниц, который можно получить как sysconf(_SC_PAGE_SIZE)
- prot - флаги защиты памяти (EXEC/READ/WRITE/NONE)
- flags - флаги доступа (SHARED - совместный доступ разным процессам, PRIVATE - использование Copy-On-Write)
- fd, off_t - файл для отображения (если отображение на файл)

malloc/calloc

- При необходимости запрашивают у ядра новые страницы памяти с помощью mmap
- Поддерживают внутреннюю таблицу с информацией о выделенных страницах памяти
- calloc гарантирует заполнение нулями, и может работать быстрее, чем два вызова malloc + memset

Контроль за double free

```
void* ptr = malloc(SIZE);
free(ptr);
free(ptr);  // Повторное "освобождение"
```

If the argument does not match a pointer earlier returned by a function in POSIX.1-2008 that allocates memory as if by malloc(), or if the space has been deallocated by a call to free() or realloc(), the behavior is undefined.

Вариант 1. Ошибка выполнения

```
int main() {
  void* ptr = malloc(1);
  free(ptr); free(ptr); }

> gcc -o test main.cpp
> MALLOC_CHECK_=1 ./test
> *** Error in `./test':
double free or corruption
(fasttop): . . . . ***
```

Вариант 2. Нет ошибки

```
> gcc -o test main.cpp
> MALLOC_CHECK_=0 ./test
> ./test  # OK
```

Вариант 3. Сигнал SIGSEGV

```
int main() {
 void* ptr = malloc(9999999);
 free(ptr); free(ptr); }
> gcc -o test main.cpp
>./test
Segmentation fault
```

Разные реализации malloc/free

```
/* myclass.h */

class MyClass {
public:
  static char* getString();
};
```

MSVC

```
/* myclass.cpp */

char* MyClass::getString()
{
  result=(char*)malloc(10);
  memcpy(result, "Hello");
  return result;
}
```

MSVC

```
/* program.cpp */

void someFunc();
{
  char* r = MyClass::getString();
  printf("%s", r);
  free(r);
}
```

Разные реализации malloc/free

```
/* myclass.h */

class MyClass {
public:
  static char* getString();
};
```

MSVC

```
/* myclass.cpp */

char* MyClass::getString()
{
  result=(char*)malloc(10);
  memcpy(result, "Hello");
  return result;
}
```

MinGW

```
/* program.cpp */

void someFunc();
{
  char* r = MyClass::getString();
  printf("%s", r);
  free(r); // Segmentation Fault
}
```

Реализации malloc

- Выделяется одна или несколько *арен* - непрерывных областей памяти
- При использовании `sbrk` - арена может быть только одна
- Разные арены используются разными потоками в пределах одного процесса
- Главная проблема - фрагментация данных
- Эффективная реализация - `jmealloc`

Ограничения памяти

```
[/.../victor]$ ulimit -a
core file size          (blocks, -c) unlimited
data seg size           (kbytes, -d) unlimited
scheduling priority             (-e) 0
file size               (blocks, -f) unlimited
pending signals                 (-i) 45543
max locked memory       (kbytes, -l) 64          # невыгружаемая память
max memory size         (kbytes, -m) unlimited   # максимальная память
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        # размер стека
cpu time               (seconds, -t) unlimited
max user processes              (-u) 4096
virtual memory          (kbytes, -v) unlimited   # размер адресного простр.
file locks                      (-x) unlimited
```

```
int getrlimit(int resource, struct rlimit *rlim);
int setrlimit(int resource, const struct rlimit *rlim);
```

Если не хватит памяти

OOM Killer - механизм ядра, который выберает "жертву" и снимает процесс для освобождения памяти.

/proc/.../oom_score
 - по умолчанию 0
 - знаковое значение
 - чем выше - тем больше риск быть прибитым