**Лекция 8**

Напоминание о том, как устроена память:

Allocator -> new -> malloc -> mmap

**mmap** — сискол, с помощью которого можно запросить у ОС виртуальную память

*void\* mmap(void\* addr, size\_t length, int prot, int flags, int fd, off\_t offset)*

addr — адрес, с которого нужно начать мапинг. Если передан NULL, то никакой адрес не запрошен.

length — кол-во памяти, которое нужно запросить.

prot — флаг, который указывает, с каким доступом запросить кусок памятиf

flag — определяет, будут ли обновления отображения видны другим процессам, отображающим ту же область, и будут ли обновления перенесены в базовый файл.

fd — переданный через файловый дескриптор файл можно загрузить в виртуадьную память.

offset — смещение, кратное размеру страницы, начиная с которого выделяются байты памяти.

**munmap** — сискол, освобождающий память

*int munmap(void\* addr, size\_t length)*

**Пример выделения 1 байта памяти:**

#include <iostream>  
#include <sys/mman.h>  
  
int main() {  
 void\* ptr = mmap(NULL, 1, PROT\_READ | PROT\_WRITE,  
 MAP\_PRIVATE | MAP\_ANONYMOUS, -1, 0);  
 std::cout << ptr << '\n';  
 munmap(ptr, 1);  
}

В *cat /proc/$(pgrep a.out)/maps* можно увидеть, что замапилась целая страница (4096 байт).

Можно делать munmap на меньшее число байт, чем было выделено.

**Пример с мапингом на фиксированный адрес:**

#include <iostream>  
#include <sys/mman.h>  
  
int main() {  
 void\* ptr = mmap((void\*)0x600000000001, 1, PROT\_READ | PROT\_WRITE,  
 MAP\_PRIVATE | MAP\_ANONYMOUS, -1, 0);  
 std::cout << ptr << '\n';  
 getchar();  
 munmap(ptr, 1);  
}

**Права**

Если попытаться выполнить то, что находится под выделенным поинтером, когда у нас нет на это прав, будет segfault (компилятор попытался интерпретировать участок памяти как бинарный код).

#include <iostream>  
#include <sys/mman.h>  
  
int main() {  
 void\* ptr = mmap((void\*)0x600000000001, 1, PROT\_READ | PROT\_WRITE,  
 MAP\_PRIVATE | MAP\_ANONYMOUS, -1, 0);  
 std::cout << ptr << '\n';  
 void (\*f)() = (void(\*)()) ptr;  
 f();  
 munmap(ptr, 1);  
}

Право на чтение != праву на выполнение для обеспечения безопасности.

**mprotect** — сискол, который меняет права выделенной памяти

*int mprotect(void\* addr, size\_t len, int prot)*

Причем передаваемый адрес должен быть выровнен по размеру страницы

**Пример SIGILL**

#include <iostream>  
#include <sys/mman.h>  
#include <vector>  
  
int main() {  
 std::vector<char> v;  
  
 for (int i = 0; i < 1000000; ++i) {  
 v.push\_back(i);  
 }  
 std::cout << "vector starts at " << (int\*)&v[0] << std::endl;  
 getchar();  
 std::cout << mprotect(&v[0]-16, 10000, PROT\_READ|PROT\_WRITE|PROT\_EXEC) << std::endl;  
 getchar();  
  
 v[0] = 0x33;  
 void(\*f)() = (void(\*)()) &v[0];  
 f();  
}

Добавили права на выполнение для памяти, выделенной под вектор, затем испортили бинарный код, на который есть права на исполнение — получили **illegal instruction**.

**mremap** — сискол, который позволяет реаллоцировать память

*void \*mremap(void\* old\_address, size\_t old\_size,*

*size\_t new\_size, int flags, ... /\* void\* new\_address \*/);*

**Пример: мапинг файла**

#include <iostream>  
#include <sys/mman.h>  
#include <stdlib.h>  
#include <sys/stat.h>  
#include <fcntl.h>  
#include <unistd.h>  
#include <vector>  
  
int main() {  
 int fd = open("example.txt", O\_RDWR);  
 if (fd == -1) {  
 exit(1);  
 }  
 struct stat sb;  
 if (fstat(fd, &sb) == -1) {  
 exit(1);  
 }  
 char\* data = (char\*)mmap(NULL, sb.st\_size, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);  
 if (data == MAP\_FAILED) {  
 exit(1);  
 }  
 printf("%s\n", data);  
 data[0] = 'x';  
  
 if (msync(data, sb.st\_size, MS\_SYNC) == -1) {  
 exit(1);  
 }  
  
 if (munmap(data, sb.st\_size) == -1) {  
 exit(1);  
 }  
 close(fd);  
}

Если вместо MAP\_SHARED указать флаг MAP\_PRIVATE, то несмотря на то, что файл был открыт с правами только на чтение, мы сможем поменять data, потому что создастся локальная копия этого файла и изменения в нем не будут влиять на изначальный файл. MAP\_SHARED же позволит поменять информацию в самом изначальном файле.

Тем не менее, когда мы пишем data[0] = `x`, то необязательно в самом файле сразу что-то поменяется, потому что обновлять файл при каждом изменении было бы неэффективно, поэтому ОС делает это время от времени, синхронизируя изменения с файлом. Чтобы гарантировать, что эти изменения сохранились, нужно использовать сискол **msync** — он синхронизирует файл с виртуальной памятью.

*int msync(void\* addr, size\_t length, int flags);*