Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

Лабораторная работа №4 по курсу

«Операционные системы»

Группа: М8О-213Б-23

Студент: Арсельгов А. Б.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 13.12.24

Москва, 2024

Постановка задачи

Вариант 2.

Списки свободных блоков (первое подходящее) и алгоритм Мак-КьюзиКэрелса

Общий метод и алгоритм решения

Использованные системные вызовы:

- `mmap()` – для выделения блока памяти (аналог malloc), используется для резервирования памяти для аллокатора.
- `munmap()` – для освобождения ранее выделенной памяти (аналог free), применяется для очистки ресурсов, выделенных через `mmap()`.
- `dlopen()` – для загрузки динамической библиотеки во время выполнения программы, используется для подключения аллокатора из внешней библиотеки.
- `dlsym()` – для получения указателей на функции из динамически загруженной библиотеки, позволяет использовать функции аллокатора.
- `dlclose()` – для закрытия загруженной динамической библиотеки, освобождает ресурсы, связанные с загрузкой библиотеки.

В рамках данной лабораторной работы я реализовал два аллокатора памяти с использованием различных стратегий распределения памяти: First Fit и McKusick-Karels. Эти аллокаторы предназначены для работы с памятью в стиле низкоуровневого управления, без использования стандартных средств выделения памяти, таких как `malloc` и `free`. Вместо этого я использовал системные вызовы `mmap`, `munmap` для выделения и освобождения памяти, а также `dlopen`, `dlsym` и `dlclose` для работы с динамическими библиотеками.

- **Аллокаторы памяти**

1. **Аллокатор с первым подходящим блоком (First Fit)**: В этом аллокаторе поиск подходящего свободного блока памяти для выделения происходит с начала списка свободных блоков. Как только найден первый подходящий блок, память выделяется, а остаток блока, если он есть, сохраняется как новый свободный блок.

2. **Аллокатор McKusick-Karels**: Этот аллокатор использует стратегию распределения памяти, где блоки памяти организуются в свободные списки, что позволяет эффективно управлять памятью и избежать фрагментации. Он использует списки свободных блоков и делит их на разные группы по размеру.

Работа программы

Программа состоит из трех частей:

1. **Основная программа (main.c)**: Эта часть программы загружает динамические библиотеки, содержащие различные аллокаторы, с помощью системного вызова `dlopen`. Аллокатор выбирается в зависимости от переданного пути к библиотеке. Далее, с помощью функции `allocator_create`, создается аллокатор, выделяется память, и выполняются операции выделения и освобождения блоков с использованием выбранного аллокатора.

2. **Аллокаторы (first_fit_allocator.c и mckusick_karels_allocator.c)**: Эти файлы содержат реализацию двух разных алгоритмов выделения памяти. Внутри этих библиотек реализованы функции, такие как `allocator_create`, `allocator_alloc`, `allocator_free`, и `allocator_destroy`, которые используются для управления памятью.

3. **Использование системных вызовов**: Для выделения памяти в программе я использовал системный вызов `mmap`. Он позволяет выделить память без использования стандартных функций выделения памяти, таких как `malloc`. В отличие от `malloc`, `mmap` может работать с неструктурированной памятью и позволяет точнее контролировать выделение памяти.

Для освобождения памяти используется `munmap`, который освобождает блок памяти, ранее выделенный с помощью `mmap`.

4. **Динамическая загрузка библиотек**: С помощью `dlopen` загружается библиотека, содержащая код аллокатора. Затем с помощью `dlsym` получаем адреса функций аллокатора. Это позволяет гибко менять аллокатор в программе без перекомпиляции. После завершения работы программы библиотека закрывается с помощью вызова `dlclose`, что освобождает ресурсы.

Ход работы

1. Создал два аллокатора памяти с использованием разных стратегий управления памятью.
2. Реализовал систему для динамической загрузки библиотек, что позволяет выбирать аллокатор во время выполнения программы.
3. Для выделения памяти использовал системный вызов `mmap` для создания области памяти, которая затем используется как хранилище для аллокаторов.
4. Провел тестирование работы программы с обоими аллокаторами, выделяя и освобождая память с помощью методов, определенных в аллокаторах.

Как работает программа

1. При запуске программы в качестве аргумента указывается путь к динамической библиотеке, которая реализует нужный аллокатор.

2. С помощью `dlopen` библиотека загружается, а функции аллокатора извлекаются через `dlsym`.

3. Память выделяется с использованием системного вызова `mmap`.

4. Затем с помощью функций аллокатора выделяются блоки памяти (например, с помощью `allocator_alloc`), и они могут быть освобождены с помощью `allocator_free`.

5. После завершения работы с аллокатором программа вызывает `allocator_destroy` для освобождения ресурсов, связанных с аллокатором, и затем освобождает память через `munmap`.

6. В конце программа закрывает библиотеку с помощью `dlclose`.

Тестирование

Программа была протестирована с двумя различными аллокаторами, реализующими разные алгоритмы. Для каждого аллокатора были проведены тесты на выделение и освобождение памяти, и результаты были проверены с использованием вывода на экран.

В результате работы программы можно увидеть, как память выделяется и освобождается с помощью выбранного аллокатора. Вывод программы подтверждает, что оба аллокатора работают корректно, успешно выделяя и освобождая память.

- **Заключение**

Работа продемонстрировала основные принципы работы с динамическими библиотеками, системными вызовами `mmap`, `munmap`, а также реализацию двух типов аллокаторов памяти. Это позволяет лучше понять, как эффективно управлять памятью на низком уровне без использования стандартных аллокаторов, таких как `malloc` и `free`.

Код программы

allocator.h

```
#ifndef ALLOCATOR_H
#define ALLOCATOR_H

#include <stddef.h>

typedef struct Allocator {
void* memory_start;
size_t memory_size;
void* free_list;
} Allocator;

Allocator* allocator_create(void* const memory, const size_t size);
void allocator_destroy(Allocator* const allocator);
void* allocator_alloc(Allocator* const allocator, const size_t size);
void allocator_free(Allocator* const allocator, void* const memory);

#endif
```

first_fit_allocator.c

```
#include "allocator.h"

typedef struct FreeBlock {
size_t size;
struct FreeBlock* next;
} FreeBlock;

Allocator* allocator_create(void* const memory, const size_t size) {
if (!memory || size < sizeof(FreeBlock)) {
return NULL;
}

Allocator* allocator = (Allocator*)memory;
allocator->memory_start = (char*)memory + sizeof(Allocator);
allocator->memory_size = size - sizeof(Allocator);
allocator->free_list = allocator->memory_start;

FreeBlock* initial_block = (FreeBlock*)allocator->memory_start;
initial_block->size = allocator->memory_size;
initial_block->next = NULL;

return allocator;
}

void allocator_destroy(Allocator* const allocator) {
allocator->memory_start = NULL;
allocator->memory_size = 0;
allocator->free_list = NULL;
}
```

```
void* allocator_alloc(Allocator* const allocator, const size_t size) {
if (!allocator || size == 0) {
return NULL;
}

FreeBlock* prev = NULL;
FreeBlock* curr = (FreeBlock*)allocator->free_list;

while (curr) {
if (curr->size >= size + sizeof(FreeBlock)) {
if (curr->size > size + sizeof(FreeBlock)) {
FreeBlock* new_block = (FreeBlock*)((char*)curr + sizeof(FreeBlock) + size);
new_block->size = curr->size - size - sizeof(FreeBlock);
new_block->next = curr->next;

curr->size = size;
curr->next = new_block;
}

if (prev) {
prev->next = curr->next;
} else {
allocator->free_list = curr->next;
}

return (char*)curr + sizeof(FreeBlock);
}

prev = curr;
curr = curr->next;
}

return NULL;
}

void allocator_free(Allocator* const allocator, void* const memory) {
if (!allocator || !memory) {
return;
}

FreeBlock* block = (FreeBlock*)((char*)memory - sizeof(FreeBlock));
block->next = (FreeBlock*)allocator->free_list;
allocator->free_list = block;
}
```

mckusick_karels_allocator.c

```
#include "allocator.h"
```

```
typedef struct FreeBlock {
struct FreeBlock* next;
} FreeBlock;

#define ALIGN(size, alignment) (((size) + (alignment - 1)) & ~(alignment - 1))

Allocator* allocator_create(void* const memory, const size_t size) {
if (!memory || size < sizeof(Allocator)) {
return NULL;
}

Allocator* allocator = (Allocator*)memory;
allocator->memory_start = (char*)memory + sizeof(Allocator);
allocator->memory_size = size - sizeof(Allocator);
allocator->free_list = allocator->memory_start;

FreeBlock* block = (FreeBlock*)allocator->memory_start;
block->next = NULL;

return allocator;
}

void allocator_destroy(Allocator* const allocator) {
allocator->memory_start = NULL;
allocator->memory_size = 0;
allocator->free_list = NULL;
}

void* allocator_alloc(Allocator* const allocator, const size_t size) {
if (!allocator || size == 0) {
return NULL;
}

size_t aligned_size = ALIGN(size, sizeof(void*));
FreeBlock* prev = NULL;
FreeBlock* curr = (FreeBlock*)allocator->free_list;

while (curr) {
if (aligned_size <= allocator->memory_size) {
if (prev) {
prev->next = curr->next;
} else {
allocator->free_list = curr->next;
}
return curr;
}

prev = curr;
curr = curr->next;
}
```

```
return NULL;
}

void allocator_free(Allocator* const allocator, void* const memory) {
if (!allocator || !memory) {
return;
}

FreeBlock* block = (FreeBlock*)memory;
block->next = (FreeBlock*)allocator->free_list;
allocator->free_list = block;
}
```

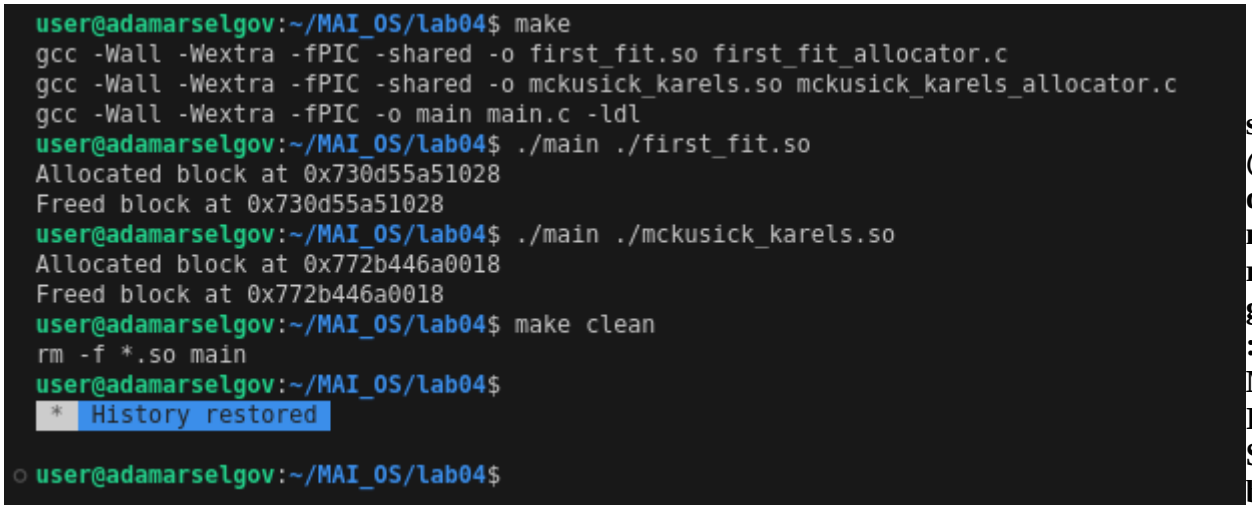
main.c

```
#include <stdio.h>
#include <dlfcn.h>
#include <sys/mman.h>
#include "allocator.h"

#define MEMORY_SIZE 1024 * 1024

int main(int argc, char* argv[]) {
if (argc < 2) {
fprintf(stderr, "Usage: %s <path_to_allocator_library>\n", argv[0]);
return 1;
}

void* handle = dlopen(argv[1], RTLD_LAZY);
if (!handle) {
fprintf(stderr, "Failed to load library: %s\n", dlerror());
return 1;
}

Allocator* (*allocator_create)(void*, size_t) = dlsym(handle, "allocator_create");
void (*allocator_destroy)(Allocator*) = dlsym(handle, "allocator_destroy");
void* (*allocator_alloc)(Allocator*, size_t) = dlsym(handle, "allocator_alloc");
void (*allocator_free)(Allocator*, void*) = dlsym(handle, "allocator_free");

char* error;
if ((error = dlerror()) != NULL) {
fprintf(stderr, "Error resolving symbols: %s\n", error);
dlclose(handle);
return 1;
}

void* memory = mmap(NULL, MEMORY_SIZE, PROT_READ | PROT_WRITE, MAP_PRIVATE |
MAP_ANONYMOUS, -1, 0);
if (memory == MAP_FAILED) {
perror("mmap failed");
dlclose(handle);
return 1;
```

```
}

Allocator* allocator = allocator_create(memory, MEMORY_SIZE);
if (!allocator) {
fprintf(stderr, "Failed to create allocator\n");
munmap(memory, MEMORY_SIZE);
dlclose(handle);
return 1;
}

void* block = allocator_alloc(allocator, 128);
if (block) {
printf("Allocated block at %p\n", block);
} else {
printf("Failed to allocate block\n");
}

allocator_free(allocator, block);
printf("Freed block at %p\n", block);

allocator_destroy(allocator);
munmap(memory, MEMORY_SIZE);
dlclose(handle);

return 0;
}
```

```
makefile
CC = gcc
CFLAGS = -Wall -Wextra -fPIC
LDFLAGS = -shared

all: first_fit.so mckusick_karels.so main

first_fit.so: first_fit_allocator.c allocator.h
$(CC) $(CFLAGS) $(LDFLAGS) -o $@ $<

mckusick_karels.so: mckusick_karels_allocator.c allocator.h
$(CC) $(CFLAGS) $(LDFLAGS) -o $@ $<

main: main.c
$(CC) $(CFLAGS) -o $@ $< -ldl

clean:
rm -f *.so main
```

Протокол работы программы



user@adamarselgov:~/MAI_OS/lab04/src$ strace ./main ./first_fit.so

execve("./main", ["./main", "./first_fit.so"], 0x7ffdaf9c5a18 /* 78 vars */) = 0

brk(NULL) = 0x6308705cd000

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7a9b9cc88000

access("/etc/ld.so.preload", R_OK) = -1 ENOENT (Нет такого файла или каталога)

openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3

fstat(3, {st_mode=S_IFREG|0644, st_size=83935, ...}) = 0

mmap(NULL, 83935, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7a9b9cc73000

close(3) = 0

openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3

read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\220\243\2\0\0\0\0\0"..., 832) = 832

pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784

fstat(3, {st_mode=S_IFREG|0755, st_size=2125328, ...}) = 0

pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784

mmap(NULL, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7a9b9ca00000

mmap(0x7a9b9ca28000, 1605632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7a9b9ca28000

mmap(0x7a9b9cbb0000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b0000) = 0x7a9b9cbb0000

mmap(0x7a9b9cbff000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000) = 0x7a9b9cbff000

mmap(0x7a9b9cc05000, 52624, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7a9b9cc05000

close(3) = 0

mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7a9b9cc70000

arch_prctl(ARCH_SET_FS, 0x7a9b9cc70740) = 0

set_tid_address(0x7a9b9cc70a10) = 11134

set_robust_list(0x7a9b9cc70a20, 24) = 0

rseq(0x7a9b9cc71060, 0x20, 0, 0x53053053) = 0

mprotect(0x7a9b9cbff000, 16384, PROT_READ) = 0

mprotect(0x6308705bb000, 4096, PROT_READ) = 0

mprotect(0x7a9b9ccc0000, 8192, PROT_READ) = 0

prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0

munmap(0x7a9b9cc73000, 83935) = 0

getrandom("\x04\x57\xf9\xe6\xe2\x20\x32\x03", 8, GRND_NONBLOCK) = 8

brk(NULL) = 0x6308705cd000

brk(0x6308705ee000) = 0x6308705ee000

openat(AT_FDCWD, "./first_fit.so", O_RDONLY|O_CLOEXEC) = 3

read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0"..., 832) = 832

fstat(3, {st_mode=S_IFREG|0775, st_size=15248, ...}) = 0

getcwd("/home/user/MAI_OS/lab04/src", 128) = 28

mmap(NULL, 16400, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7a9b9cc83000

mmap(0x7a9b9cc84000, 4096, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1000) = 0x7a9b9cc84000

mmap(0x7a9b9cc85000, 4096, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2000) = 0x7a9b9cc85000

mmap(0x7a9b9cc86000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2000) = 0x7a9b9cc86000

close(3) = 0

mprotect(0x7a9b9cc86000, 4096, PROT_READ) = 0

mmap(NULL, 1048576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7a9b9c900000

fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}) = 0

write(1, "Allocated block at 0x7a9b9c90002"..., 34Allocated block at 0x7a9b9c900028

) = 34

write(1, "Freed block at 0x7a9b9c900028\n", 30Freed block at 0x7a9b9c900028

) = 30

munmap(0x7a9b9c900000, 1048576) = 0

munmap(0x7a9b9cc83000, 16400) = 0

user@adamarselgov:~/MAI_OS/lab04/src$

user@adamarselgov:~/MAI_OS/lab04/src$ strace ./main ./mckusick_karels.so

execve("./main", ["./main", "./mckusick_karels.so"], 0x7ffe5f2f3ef8 /* 78 vars */) = 0

brk(NULL) = 0x5e03e7a60000

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x714d5c11e000

access("/etc/ld.so.preload", R_OK) = -1 ENOENT (Нет такого файла или каталога)

openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3

fstat(3, {st_mode=S_IFREG|0644, st_size=83935, ...}) = 0

mmap(NULL, 83935, PROT_READ, MAP_PRIVATE, 3, 0) = 0x714d5c109000

close(3) = 0

openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3

read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\220\243\2\0\0\0\0\0"..., 832) = 832

pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784

fstat(3, {st_mode=S_IFREG|0755, st_size=2125328, ...}) = 0

pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784

mmap(NULL, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x714d5be00000

mmap(0x714d5be28000, 1605632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x714d5be28000

mmap(0x714d5bfb0000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b0000) = 0x714d5bfb0000

mmap(0x714d5bfff000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000) = 0x714d5bfff000

mmap(0x714d5c005000, 52624, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x714d5c005000

close(3) = 0

mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x714d5c106000

arch_prctl(ARCH_SET_FS, 0x714d5c106740) = 0

set_tid_address(0x714d5c106a10) = 11207

set_robust_list(0x714d5c106a20, 24) = 0

rseq(0x714d5c107060, 0x20, 0, 0x53053053) = 0

mprotect(0x714d5bfff000, 16384, PROT_READ) = 0

mprotect(0x5e03e5c2d000, 4096, PROT_READ) = 0

mprotect(0x714d5c156000, 8192, PROT_READ) = 0

prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0

munmap(0x714d5c109000, 83935) = 0

getrandom("\xc6\x74\x73\x12\x42\x54\x72\x17", 8, GRND_NONBLOCK) = 8

brk(NULL) = 0x5e03e7a60000

brk(0x5e03e7a81000) = 0x5e03e7a81000

openat(AT_FDCWD, "./mckusick_karels.so", O_RDONLY|O_CLOEXEC) = 3

read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0\0"..., 832) = 832

fstat(3, {st_mode=S_IFREG|0775, st_size=15256, ...}) = 0

getcwd("/home/user/MAI_OS/lab04/src", 128) = 28

mmap(NULL, 16400, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x714d5c119000

mmap(0x714d5c11a000, 4096, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1000) = 0x714d5c11a000

mmap(0x714d5c11b000, 4096, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2000) = 0x714d5c11b000

mmap(0x714d5c11c000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2000) = 0x714d5c11c000

close(3) = 0

mprotect(0x714d5c11c000, 4096, PROT_READ) = 0

mmap(NULL, 1048576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x714d5bd00000

fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}) = 0

write(1, "Allocated block at 0x714d5bd0001"..., 34Allocated block at 0x714d5bd00018

) = 34

write(1, "Freed block at 0x714d5bd00018\n", 30Freed block at 0x714d5bd00018

) = 30

munmap(0x714d5bd00000, 1048576) = 0

munmap(0x714d5c119000, 16400) = 0

exit_group(0) = ?

+++ exited with 0 +++

user@adamarselgov:~/MAI_OS/lab04/src$

Вывод

В ходе выполнения лабораторной работы была реализована система выделения и освобождения памяти с использованием собственного аллокатора, а также взаимодействие с динамическими библиотеками через `dlopen` и `dlsym`. Основной трудностью оказалось корректное использование системных вызовов `mmap` и `munmap` для выделения памяти, а также правильная настройка динамической загрузки библиотек. Также возникли проблемы с настройкой Makefile для корректной сборки библиотек и основной программы. В будущем хотелось бы улучшить обработку ошибок и сделать код более устойчивым к некорректным данным.