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-211Б-23

Студент: Соболин Т.С. Преподаватель: Бахарев В.Д.

Оценка:

Дата: 14.01.25

Москва, 2025

# Постановка задачи

**Вариант 1.**

Требуется создать две динамические библиотеки, реализующие два аллокатора: списки свободных блоков (первое подходящее) и блоки по 2^n.

# Общий метод и алгоритм решения

Использованные системные вызовы:

* **\*int munmap(void addr, size\_t length); -** Удаляет все отображения из заданной области памяти.
* **\*int dlclose(void handle);** - Закрывает динамическую библиотеку, открытую с помощью dlopen, и освобождает ресурсы, связанные с этим дескриптором.
* **\*\*void dlopen(const char filename, int flag);** - Открывает динамическую библиотеку и возвращает дескриптор для последующего использования.
* **\*\*void mmap(void addr, size\_t length, int prot, int flags, int fd, off\_t offset);** – создает новое отображение памяти или изменяет существующее**.**
* **int write(int \_Filehandle, const void \*\_Buf, unsigned int \_MaxCharCount) –** выводит информацию в файл с указанным дескриптором.

## Описание программы

1. main.c

Открывает динамические библиотеки и получает нужные функции. Если в библиотеке не нашлось нужных функций, то вместо них будут использоваться аварийные оберточные функции. Далее как пример функция выделяет и освобождает память массива.

1. 2n\_degree\_blocks.c

Файл в котором реализована логика работы аллокатора блоками по 2^n.

* 1. Вся память при инициализации разбивается на блоки которые равны степени двойки.
  2. Все блоки хранятся в списке свободных элементов.
  3. Каждый блок хранит указатель на следующий свободный блок.
  4. При освобождении нужно добавить этот блок в список свободных элементов в нужную позицию.
  5. Для выделения памяти выбираем блок N[log2(size)] и возвращаем указатель на первый элемент, помечая блок занятым.

1. free\_list\_blocks.c

Файл в котором реализована логика работы аллокатора на спискач свободных блоков. 1) Все свободные блоки организованы в список

* 1. В блоке хранится его размер и указатель на следующий свободный блок
  2. Для выделения памяти проходимся по всему списку свободных блоков и выбираем минимальный блок, который больше или равен по размеру нужного блока. Помечаем блок, как занятый и убираем из списка.
  3. При освобождении памяти возвращаем блок в список свободных элементов. И при возможности сливаем рядом стоящие блоки.

**Код программы**

## main.c

#include <dlfcn.h>

#include <stddef.h>

#include <stdint.h>

#include <stdlib.h>

#include <string.h>

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

typedef struct AllocatorAPI { void\* (\*create)(void\* addr, size\_t size);

void\* (\*alloc)(void\* allocator, size\_t size);

void (\*free)(void\* allocator, void\* ptr);

void (\*destroy)(void\* allocator);

} AllocatorAPI;

void\* default\_create(void\* memory, size\_t size) { (void)size; return memory; }

void\* default\_alloc(void\* allocator, size\_t size) { (void)allocator; uint32\_t\* block = mmap(NULL, size + sizeof(uint32\_t),

PROT\_READ | PROT\_WRITE,

MAP\_SHARED |

MAP\_ANONYMOUS, -1, 0); if (block == MAP\_FAILED) { return NULL;

}

\*block = (uint32\_t)(size + sizeof(uint32\_t));

return block + 1;

}

void default\_free(void\* allocator, void\* memory) {

(void)allocator; if (!memory) return; uint32\_t\* block =

(uint32\_t\*)memory - 1; munmap(block, \*block);

}

void default\_destroy(void\* allocator) {

(void)allocator;

}

void load\_allocator(const char\* lib\_path, AllocatorAPI\* api) {

void\* lib\_handle = dlopen(lib\_path,

### RTLD\_LOCAL | RTLD\_NOW);

if (!lib\_path || !lib\_path[0] ||

!lib\_handle) { write(STDERR\_FILENO, "WARNING: Using default

allocator\n", 34);

api->create = default\_create; api->alloc = default\_alloc; api->free = default\_free; api->destroy = default\_destroy; return; }

api->create = dlsym(lib\_handle,

"create\_allocator"); api->alloc = dlsym(lib\_handle,

"allocate\_memory"); api->free = dlsym(lib\_handle,

"free\_memory"); api->destroy = dlsym(lib\_handle, "destroy\_allocator");

if (!api->create || !api->alloc || !api-

>free || !api->destroy) { write(STDERR\_FILENO, "ERROR: Failed loading allocator functions\n", 43);

dlclose(lib\_handle); api->create = default\_create; api->alloc = default\_alloc; api->free = default\_free; api->destroy = default\_destroy;

}

}

void print\_message(const char\* msg) {

write(STDOUT\_FILENO, msg, strlen(msg));

}

void print\_address(const char\* label,

int index, void\* address) { char buffer[64];

int len = 0;

while (\*label) { buffer[len++] = \*label++;

}

if (index < 10) { buffer[len++] = '0' + index; } else { buffer[len++] = '0' + (index / 10); buffer[len++] = '0' + (index % 10);

}

char\* ad = " address: "; while (\*ad) { buffer[len++] = \*ad++;

}

uintptr\_t addr = (uintptr\_t)address; for (int i = (sizeof(uintptr\_t) \* 2) - 1; i

>= 0; --i) { int nibble = (addr >> (i \* 4)) &

0xF;

buffer[len++] = (nibble < 10) ? ('0' + nibble) : ('a' + (nibble - 10));

}

buffer[len++] = '\n';

write(STDOUT\_FILENO, buffer, len);

}

int main(int argc, char\*\* argv) { const char\* lib\_path = (argc > 1) ? argv[1] : NULL; AllocatorAPI allocator\_api; load\_allocator(lib\_path, &allocator\_api);

size\_t pool\_size = 4096; void\* pool = mmap(NULL,

pool\_size, PROT\_READ |

PROT\_WRITE,

MAP\_PRIVATE |

MAP\_ANONYMOUS, -1, 0); if (pool == MAP\_FAILED) { print\_message("ERROR: Memory

pool allocation failed\n");

return EXIT\_FAILURE;

}

void\* allocator = allocator\_api.create(pool, pool\_size);

if (!allocator) { print\_message("ERROR:

Allocator initialization failed\n"); munmap(pool, pool\_size); return EXIT\_FAILURE;

}

size\_t block\_sizes[] = {12, 13, 24,

40, 56, 100, 120, 400}; void\* blocks[8];

for (int i = 0; i < 8; ++i) { blocks[i] = allocator\_api.alloc(allocator, block\_sizes[i]); if (!blocks[i]) { print\_message("ERROR:

Memory allocation failed\n");

break;

}

print\_address("block №", i + 1,

blocks[i]);

}

print\_message("INFO: Memory

allocation - SUCCESS\n");

for (int i = 0; i < 8; ++i) { if (blocks[i]) { allocator\_api.free(allocator,

blocks[i]);

}

}

print\_message("INFO: Memory

freed\n");

allocator\_api.destroy(allocator); print\_message("INFO: Allocator

destroyed\n");

return EXIT\_SUCCESS;

}

## free\_list\_blocks.c

#include <stddef.h>

typedef struct Allocator { void\* start; size\_t total\_size; void\* free\_blocks;

} Allocator;

typedef struct FreeBlock { size\_t block\_size;

struct FreeBlock\* next\_block;

} FreeBlock;

Allocator\* create\_allocator(void\* memory\_pool, size\_t pool\_size) {

if (memory\_pool == NULL || pool\_size < sizeof(FreeBlock)) {

return NULL;

}

Allocator\* allocator = (Allocator\*)memory\_pool; allocator->start = (char\*)memory\_pool + sizeof(Allocator); allocator->total\_size = pool\_size - sizeof(Allocator); allocator->free\_blocks = allocator->start;

FreeBlock\* first\_block = (FreeBlock\*)allocator->start; first\_block->block\_size = allocator->total\_size; first\_block->next\_block = NULL;

return allocator;

}

void destroy\_allocator(Allocator\* allocator) { if (allocator == NULL) { return;

}

allocator->start = NULL; allocator->total\_size = 0;

allocator->free\_blocks = NULL;

}

void\* allocate\_memory(Allocator\* allocator, size\_t request\_size) {

if (allocator == NULL || request\_size == 0) {

return NULL;

}

FreeBlock\* previous = NULL;

FreeBlock\* current = (FreeBlock\*)allocator->free\_blocks;

while (current != NULL) {

if (current->block\_size >= request\_size + sizeof(FreeBlock)) { if (current->block\_size > request\_size + sizeof(FreeBlock)) {

FreeBlock\* remaining\_block = (FreeBlock\*)((char\*)current + sizeof(FreeBlock) + request\_size);

remaining\_block->block\_size = current->block\_size - request\_size - sizeof(FreeBlock); remaining\_block->next\_block = current->next\_block;

current->block\_size = request\_size;

current->next\_block = remaining\_block;

}

if (previous != NULL) {

previous->next\_block = current->next\_block;

} else {

allocator->free\_blocks = current->next\_block;

}

return (char\*)current + sizeof(FreeBlock);

}

previous = current;

current = current->next\_block;

}

return NULL;

}

void free\_memory(Allocator\* allocator, void\* memory) {

if (allocator == NULL || memory == NULL) {

return;

}

FreeBlock\* block\_to\_free = (FreeBlock\*)((char\*)memory - sizeof(FreeBlock)); block\_to\_free->next\_block = (FreeBlock\*)allocator->free\_blocks; allocator->free\_blocks = block\_to\_free;

}

## 2n\_degree\_blocks.c

#include <math.h>

#include <stdbool.h>

#include <stdint.h>

#include <stdlib.h>

#include <string.h>

#include <sys/mman.h>

#include <unistd.h>

#define MIN\_BLOCK\_SIZE 16

typedef struct BlockHeader {

struct BlockHeader \*next;

} BlockHeader;

typedef struct Allocator {

BlockHeader \*\*free\_lists; size\_t num\_lists; void \*base\_address;

size\_t total\_size;

} Allocator;

static int calculate\_log2(int value) { int result = -1;

while (value > 0) {

value >>= 1;

result++;

}

return result;

}

Allocator\* create\_allocator(void \*memory, size\_t size) {

if (memory == NULL || size < sizeof(Allocator)) {

return NULL;

}

Allocator \*allocator = (Allocator \*)memory; allocator->base\_address = memory; allocator->total\_size = size;

size\_t min\_usable\_size = sizeof(BlockHeader) + MIN\_BLOCK\_SIZE; size\_t max\_block\_size = (size < 32) ? 32 : size;

allocator->num\_lists = (size\_t)(calculate\_log2(max\_block\_size) / 2) + 3; allocator->free\_lists = (BlockHeader \*\*)((char \*)memory + sizeof(Allocator));

for (size\_t i = 0; i < allocator->num\_lists; i++) { allocator->free\_lists[i] = NULL;

}

void \*current\_block = (char \*)memory + sizeof(Allocator) + allocator->num\_lists \* sizeof(BlockHeader \*); size\_t remaining\_size = size - sizeof(Allocator) - allocator->num\_lists \* sizeof(BlockHeader \*);

size\_t block\_size = MIN\_BLOCK\_SIZE; while (remaining\_size >= min\_usable\_size) {

if (block\_size > remaining\_size) {

break;

}

size\_t num\_blocks = (remaining\_size >= (block\_size + sizeof(BlockHeader)) \* 2) ? 2 : 1;

for (size\_t i = 0; i < num\_blocks; i++) {

BlockHeader \*header = (BlockHeader \*)current\_block; size\_t index = (block\_size == 0) ? 0 : calculate\_log2(block\_size);

header->next = allocator->free\_lists[index]; allocator->free\_lists[index] = header;

current\_block = (char \*)current\_block + block\_size;

remaining\_size -= block\_size;

}

block\_size <<= 1;

}

return allocator;

}

void destroy\_allocator(Allocator \*allocator) { if (allocator != NULL) {

munmap(allocator->base\_address, allocator->total\_size);

}

}

void\* allocate\_memory(Allocator \*allocator, size\_t size) { if (allocator == NULL || size == 0) {

return NULL;

}

size\_t index = (size == 0) ? 0 : calculate\_log2(size) + 1;

if (index >= allocator->num\_lists) {

index = allocator->num\_lists - 1;

}

while (index < allocator->num\_lists && allocator->free\_lists[index] == NULL) {

index++;

}

if (index >= allocator->num\_lists) {

return NULL;

}

BlockHeader \*block = allocator->free\_lists[index]; allocator->free\_lists[index] = block->next;

return (void \*)((char \*)block + sizeof(BlockHeader));

}

void free\_memory(Allocator \*allocator, void \*ptr) {

if (allocator == NULL || ptr == NULL) {

return;

}

BlockHeader \*block = (BlockHeader \*)((char \*)ptr - sizeof(BlockHeader)); size\_t block\_offset = (char \*)block - (char \*)allocator->base\_address;

size\_t block\_size = MIN\_BLOCK\_SIZE;

while ((block\_size << 1) <= block\_offset) {

block\_size <<= 1;

}

size\_t index = calculate\_log2(block\_size);

if (index >= allocator->num\_lists) {

index = allocator->num\_lists - 1;

}

block->next = allocator->free\_lists[index];

allocator->free\_lists[index] = block;

}

Протокол работы программы

kotlasboy@kotlasboy-Modern-15-B12M:~/Programming/Projects/OS/lab\_4$ ./main

WARNING: Using default allocator

block №1 address: 00007dca5badb004

block №2 address: 00007dca5bada004

block №3 address: 00007dca5bad9004

block №4 address: 00007dca5bad8004

block №5 address: 00007dca5bad7004

block №6 address: 00007dca5bad6004

block №7 address: 00007dca5bad5004

block №8 address: 00007dca5bad4004

INFO: Memory allocation - SUCCESS

INFO: Memory freed

INFO: Allocator destroyed

kotlasboy@kotlasboy-Modern-15-B12M:~/Programming/Projects/OS/lab\_4$ ./main ./2ndegree.so

block №1 address: 00007883b52f6080

block №2 address: 00007883b52f6070

block №3 address: 00007883b52f60b0

block №4 address: 00007883b52f6110

block №5 address: 00007883b52f60d0

block №6 address: 00007883b52f61d0

block №7 address: 00007883b52f6150

block №8 address: 00007883b52f6350

INFO: Memory allocation - SUCCESS

INFO: Memory freed

INFO: Allocator destroyed

kotlasboy@kotlasboy-Modern-15-B12M:~/Programming/Projects/OS/lab\_4$ ./main ./flist.so

block №1 address: 000076b8e7ce0028

block №2 address: 000076b8e7ce0044

block №3 address: 000076b8e7ce0061

block №4 address: 000076b8e7ce0089

block №5 address: 000076b8e7ce00c1

block №6 address: 000076b8e7ce0109

block №7 address: 000076b8e7ce017d

block №8 address: 000076b8e7ce0205

INFO: Memory allocation - SUCCESS

INFO: Memory freed

INFO: Allocator destroyed

**Strace:**

kotlasboy@kotlasboy-Modern-15-B12M:~/Programming/Projects/OS/lab\_4$ strace -f ./main ./flist.so

execve("./main", ["./main", "./flist.so"], 0x7fffc397b7f0 /\* 60 vars \*/) = 0

brk(NULL) = 0x653fca54f000

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x76b48048d000

access("/etc/ld.so.preload", R\_OK) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "/etc/ld.so.cache", O\_RDONLY|O\_CLOEXEC) = 3

fstat(3, {st\_mode=S\_IFREG|0644, st\_size=68847, ...}) = 0

mmap(NULL, 68847, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x76b48047c000

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) = 0x76b480200000

mmap(0x76b480228000, 1605632, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x28000) = 0x76b480228000

mmap(0x76b4803b0000, 323584, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1b0000) = 0x76b4803b0000

mmap(0x76b4803ff000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1fe000) = 0x76b4803ff000

mmap(0x76b480405000, 52624, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x76b480405000

close(3) = 0

mmap(NULL, 12288, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x76b480479000

arch\_prctl(ARCH\_SET\_FS, 0x76b480479740) = 0

set\_tid\_address(0x76b480479a10) = 66893

set\_robust\_list(0x76b480479a20, 24) = 0

rseq(0x76b48047a060, 0x20, 0, 0x53053053) = 0

mprotect(0x76b4803ff000, 16384, PROT\_READ) = 0

mprotect(0x653fb29da000, 4096, PROT\_READ) = 0

mprotect(0x76b4804c5000, 8192, PROT\_READ) = 0

prlimit64(0, RLIMIT\_STACK, NULL, {rlim\_cur=8192\*1024, rlim\_max=RLIM64\_INFINITY}) = 0

munmap(0x76b48047c000, 68847) = 0

getrandom("\x8b\xa3\x6f\x7d\x3a\xa0\x53\x7d", 8, GRND\_NONBLOCK) = 8

brk(NULL) = 0x653fca54f000

brk(0x653fca570000) = 0x653fca570000

openat(AT\_FDCWD, "./flist.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=15248, ...}) = 0

getcwd("/home/kotlasboy/Programming/Projects/OS/lab\_4", 128) = 46

mmap(NULL, 16400, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x76b480488000

mmap(0x76b480489000, 4096, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1000) = 0x76b480489000

mmap(0x76b48048a000, 4096, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x2000) = 0x76b48048a000

mmap(0x76b48048b000, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x2000) = 0x76b48048b000

close(3) = 0

mprotect(0x76b48048b000, 4096, PROT\_READ) = 0

mmap(NULL, 4096, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x76b480487000

write(1, "block \342\204\2261 address: 000076b48048"..., 37block №1 address: 000076b480487028

) = 37

write(1, "block \342\204\2262 address: 000076b48048"..., 37block №2 address: 000076b480487044

) = 37

write(1, "block \342\204\2263 address: 000076b48048"..., 37block №3 address: 000076b480487061

) = 37

write(1, "block \342\204\2264 address: 000076b48048"..., 37block №4 address: 000076b480487089

) = 37

write(1, "block \342\204\2265 address: 000076b48048"..., 37block №5 address: 000076b4804870c1

) = 37

write(1, "block \342\204\2266 address: 000076b48048"..., 37block №6 address: 000076b480487109

) = 37

write(1, "block \342\204\2267 address: 000076b48048"..., 37block №7 address: 000076b48048717d

) = 37

write(1, "block \342\204\2268 address: 000076b48048"..., 37block №8 address: 000076b480487205

) = 37

write(1, "INFO: Memory allocation - SUCCES"..., 34INFO: Memory allocation - SUCCESS

) = 34

write(1, "INFO: Memory freed\n", 19INFO: Memory freed

) = 19

write(1, "INFO: Allocator destroyed\n", 26INFO: Allocator destroyed

) = 26

exit\_group(0) = ?

+++ exited with 0 +++

# Вывод

В рамках лабораторной работы была разработана программа, демонстрирующая работу аллокатора передаваемого в качестве аргумента при вызове программы. Было реализовано 2 аллокатора и проведена работа по сравнению их работоспособности.