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.
2. 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.
   1) Все блоки хранятся в списке свободных элементов.
   2) Каждый блок хранит указатель на следующий свободный блок.
   3) При освобождении нужно добавить этот блок в список свободных элементов в нужную позицию.
   4) Для выделения памяти выбираем блок N[log2(size)] и возвращаем указатель на первый элемент, помечая блок занятым.
3. free_list_blocks.c

   Файл в котором реализована логика работы аллокатора на спискач свободных блоков. 1) Все свободные блоки организованы в список
   2) В блоке хранится его размер и указатель на следующий свободный блок
   3) 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.
   4) 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.