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-210Б-23

Студент: Нугаев М. Э.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 07.01.25

Москва, 2025

Постановка задачи

Вариант 2.
Списки свободных блоков (первое подходящее) и алгоритм Мак-КьюзиКэрелса.

Общий метод и алгоритм решения

Использованные системные вызовы:

- ssize_t write(int __fd, const void *__buf, size_t __n); – записывает N байт из буфер(BUF) в файл (FD). Возвращает количество записанных байт или -1.
- void exit(int __status); – выполняет немедленное завершение программы. Все используемые программой потоки закрываются, и временные файлы удаляются, управление возвращается ОС или другой программе.
- void * mmap(void *start, size_t length, int prot , int flags, int fd, off_t offset); – отражает length байтов, начиная со смещения offset файла (или другого объекта), определенного файловым дескриптором fd, в память, начиная с адреса start.
- int munmap(void *start, size_t length); – удаляет все отражения из заданной области памяти, после чего все ссылки на данную область будут вызывать ошибку "неправильное обращение к памяти".
- void *dlopen(const char *filename, int flag); – загружает динамическую библиотеку, имя которой указано в строке filename, и возвращает прямой указатель на начало динамической библиотеки.
- void *dlsym(void *handle, char *symbol); – использует указатель на динамическую библиотеку, возвращаемую dlopen, и оканчивающееся нулем символьное имя, а затем возвращает адрес, указывающий, откуда загружается этот символ. Если символ не найден, то возвращаемым значением dlsym является NULL;
- int dlclose(void *handle); – уменьшает на единицу счетчик ссылок на указатель динамической библиотеки.

Код программы

main.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <dlfcn.h>
#include <sys/mman.h>
#include <time.h>

typedef struct AllocatorAPI {
    void* (*allocator_create)(void*, size_t);
    void (*allocator_destroy)(void*);
    void* (*allocator_alloc)(void*, size_t);
    void (*allocator_free)(void*, void*);
} AllocatorAPI;

void* default_allocator_create(void* memory, size_t size) {
    return memory;
}
```

```
void default_allocator_destroy(void* allocator) {
}

void* default_allocator_alloc(void* allocator, size_t size) {
    if (allocator) {
        return (void*)((char*)allocator + sizeof(size_t));
    }
    return NULL;
}

void default_allocator_free(void* allocator, void* memory) {
}

int main(int argc, char** argv) {
    AllocatorAPI api;
    void* library_handle = NULL;

    api.allocator_create = default_allocator_create;
    api.allocator_destroy = default_allocator_destroy;
    api.allocator_alloc = default_allocator_alloc;
    api.allocator_free = default_allocator_free;

    if (argc > 1) {
        library_handle = dlopen(argv[1], RTLD_LAZY);
        if (!library_handle) {
            fprintf(stderr, "Ошибка при загрузке библиотеки: %s\n", dlerror());
            return 1;
        }

        api.allocator_create = dlsym(library_handle, "allocator_create");
        api.allocator_destroy = dlsym(library_handle, "allocator_destroy");
        api.allocator_alloc = dlsym(library_handle, "allocator_alloc");
        api.allocator_free = dlsym(library_handle, "allocator_free");

        if (!api.allocator_create || !api.allocator_destroy || !api.allocator_alloc ||
!api.allocator_free) {
            fprintf(stderr, "Ошибка при загрузке функций из библиотеки: %s\n", dlerror());
            dlclose(library_handle);
            return 1;
        }
    } else {
        fprintf(stderr, "Библиотека не указана. Используется стандартный аллокатор.\n");
    }

    size_t pool_size = 1024 * 1024;
    void* memory = mmap(NULL, pool_size, PROT_READ | PROT_WRITE, MAP_ANONYMOUS |
MAP_PRIVATE, -1, 0);
    if (memory == MAP_FAILED) {
        perror("Ошибка mmap");
        return 1;
    }

    void* allocator = api.allocator_create(memory, pool_size);
```

```
    if (!allocator) {
        fprintf(stderr, "Ошибка при создании аллокатора.\n");
        munmap(memory, pool_size);
        return 1;
    }

    struct timespec start, end;
    double time_taken;


    for (int i = 0; i < 3; i++) {
        size_t block_size = 1024 * (i + 1);

        clock_gettime(CLOCK_MONOTONIC, &start);
        void* ptr = api.allocator_alloc(allocator, block_size);
        clock_gettime(CLOCK_MONOTONIC, &end);

        if (!ptr) {
            fprintf(stderr, "Ошибка при выделении памяти для блока размера %zu байт.\n",
block_size);
            break;
        }

        time_taken = (end.tv_sec - start.tv_sec) + (end.tv_nsec - start.tv_nsec) / 1e9;
        printf("Выделен блок: %p, размер: %zu байт, время: %.11f секунд\n", ptr,
block_size, time_taken);

        clock_gettime(CLOCK_MONOTONIC, &start);
        api.allocator_free(allocator, ptr);
        clock_gettime(CLOCK_MONOTONIC, &end);

        time_taken = (end.tv_sec - start.tv_sec) + (end.tv_nsec - start.tv_nsec) / 1e9;
        printf("Освобожден блок: %p, время: %.9f секунд\n", ptr, time_taken);
    }

    api.allocator_destroy(allocator);

    if (munmap(memory, pool_size) == -1) {
        perror("Ошибка при освобождении памяти с помощью munmap");
        return 1;
    }

    if (library_handle) {
        dlclose(library_handle);
    }

    return 0;
}
```

alloc1.c

```
#include <stdio.h>
#include <stdlib.h>
```

```c
#include <string.h>
#include <sys/mman.h>

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


typedef struct Allocator{
void* memory;     size_t
size;
    Block* free_list;
} Allocator;

Allocator* allocator_create(void* memory, size_t size) {
    Allocator* allocator = (Allocator*)mmap(NULL, sizeof(Allocator), PROT_READ |
PROT_WRITE, MAP_ANONYMOUS | MAP_PRIVATE, -1, 0);
allocator->memory = memory;     allocator->size
= size;
    allocator->free_list = (Block*)memory;
allocator->free_list->size = size;
allocator->free_list->next = NULL;     return
allocator;
}  void allocator_destroy(Allocator*
allocator) {     munmap(allocator,
sizeof(Allocator));
}
void* allocator_alloc(Allocator* allocator, size_t size) {
    Block* prev = NULL;
    Block* curr = allocator->free_list;


    while (curr != NULL) {
if (curr->size >= size) {
            if (curr->size > size + sizeof(Block)) {
                Block* new_block = (Block*)((char*)curr + sizeof(Block) + size);
new_block->size = curr->size - size - sizeof(Block);                 new_block-
>next = curr->next;                 curr->size = size;                   curr-
>next = new_block;
            }
            if (prev == NULL) {
                allocator->free_list = curr->next;
            } else {
                prev->next = curr->next;
            }
            return (void*)((char*)curr + sizeof(Block));
        }
        prev = curr;
curr = curr->next;
    }
    return NULL;
}  void allocator_free(Allocator* allocator, void* memory)
{     Block* block = (Block*)((char*)memory -
sizeof(Block));     block->next = allocator->free_list;
allocator->free_list = block; }
```

alloc2.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/mman.h>
#define ALIGN_SIZE(size, alignment) (((size) + (alignment - 1)) & ~(alignment - 1))
#define FREE_LIST_ALIGNMENT 8
 typedef struct Block {
size_t size;     struct
Block* next;
```

```
} Block;

typedef struct Allocator{
void* memory;     size_t
size;     Block*
free_list;
} Allocator;

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

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

    if (allocator->free_list != NULL) {
        allocator->free_list->size = allocator->size;         allocator-
>free_list->next = NULL;
    }
    return allocator;
}
void allocator_destroy(Allocator* allocator) {
if (allocator == NULL) {         return;
    }      allocator->memory
= NULL;     allocator->size =
0;
    allocator->free_list = NULL;
}
void* allocator_alloc(Allocator* allocator, size_t size) {
if (allocator == NULL || size == 0) {         return NULL;
    }
    size_t aligned_size = ALIGN_SIZE(size, FREE_LIST_ALIGNMENT);
    Block* prev = NULL;
    Block* curr = allocator->free_list;

    while (curr != NULL) {
        if (curr->size >= aligned_size) {
if (prev != NULL) {
                prev->next = curr->next;
            } else {
                allocator->free_list = curr->next;
            }
            return (void*)((char*)curr + sizeof(Block));
}
         prev = curr;
curr = curr->next;
    }
    return NULL;
}
void allocator_free(Allocator* allocator, void* memory) {
if (allocator == NULL || memory == NULL) {
return;
    }
```

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

Strace:

Allocated block: 0x7f1a926e9010, size: 1024 bytes, time: 0.000000200 seconds

Freed block: 0x7f1a926e9010 (id=1, name=Object 1, value=123.45), time: 0.000000100 seconds

Allocated block: 0x7f1a926e9420, size: 2048 bytes, time: 0.000000200 seconds

Freed block: 0x7f1a926e9420 (id=2, name=Object 2, value=246.90), time: 0.000000100 seconds

Allocated block: 0x7f1a926e9c30, size: 3072 bytes, time: 0.000003600 seconds

Freed block: 0x7f1a926e9c30 (id=3, name=Object 3, value=370.35), time: 0.000000000 seconds

miron@DESKTOP-GD72A05:~/LABS/lab 4$ strace -f ./main ./liballoc2.so

execve("./main", ["./main", "./liballoc2.so"], 0x7ffd10f27ff0 /* 26 vars */) = 0

brk(NULL) = 0x56529f81f000

arch_prctl(0x3001 /* ARCH_??? */, 0x7ffdb0716d50) = -1 EINVAL (Invalid argument)

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=33773, ...}) = 0

mmap(NULL, 33773, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f73c3cfb000

close(3) = 0

openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libdl.so.2", 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 \22\0\0\0\0\0"..., 832) = 832

fstat(3, {st_mode=S_IFREG|0644, st_size=18848, ...}) = 0

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f73c3cf9000

mmap(NULL, 20752, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f73c3cf3000

mmap(0x7f73c3cf4000, 8192, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1000) = 0x7f73c3cf4000

mmap(0x7f73c3cf6000, 4096, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x3000) = 0x7f73c3cf6000

mmap(0x7f73c3cf7000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x3000) = 0x7f73c3cf7000

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\300A\2\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

pread64(3, "\4\0\0\0\20\0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0", 32, 848) = 32

pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\7\2C\n\357_\243\335\2449\206V>\237\374\304"..., 68, 880) = 68

fstat(3, {st_mode=S_IFREG|0755, st_size=2029592, ...}) = 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

pread64(3, "\4\0\0\0\20\0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0", 32, 848) = 32

pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\7\2C\n\357_\243\335\2449\206V>\237\374\304"..., 68, 880) = 68

mmap(NULL, 2037344, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f73c3b01000

mmap(0x7f73c3b23000, 1540096, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x22000) = 0x7f73c3b23000

mmap(0x7f73c3c9b000, 319488, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x19a000) = 0x7f73c3c9b000

mmap(0x7f73c3ce9000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1e7000) = 0x7f73c3ce9000

mmap(0x7f73c3cef000, 13920, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f73c3cef000

close(3) = 0

mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f73c3afe000

arch_prctl(ARCH_SET_FS, 0x7f73c3afe740) = 0

mprotect(0x7f73c3ce9000, 16384, PROT_READ) = 0

mprotect(0x7f73c3cf7000, 4096, PROT_READ) = 0

mprotect(0x56529edc6000, 4096, PROT_READ) = 0

mprotect(0x7f73c3d31000, 4096, PROT_READ) = 0

munmap(0x7f73c3cfb000, 33773) = 0

brk(NULL) = 0x56529f81f000

brk(0x56529f840000) = 0x56529f840000

openat(AT_FDCWD, "./liballoc2.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@\20\0\0\0\0\0\0"..., 832) = 832

fstat(3, {st_mode=S_IFREG|0755, st_size=15776, ...}) = 0

getcwd("/home/miron/LABS/lab 4", 128) = 23

mmap(NULL, 16424, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f73c3cff000

mmap(0x7f73c3d00000, 4096, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1000) = 0x7f73c3d00000

mmap(0x7f73c3d01000, 4096, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2000) = 0x7f73c3d01000

mmap(0x7f73c3d02000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2000) = 0x7f73c3d02000

close(3) = 0

mprotect(0x7f73c3d02000, 4096, PROT_READ) = 0

mmap(NULL, 1048576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f73c39fe000

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

write(1, "Allocated block: 0x7f73c39fe028,"..., 77Allocated block: 0x7f73c39fe028, size: 1024 bytes, time: 0.000000200 seconds

) = 77

write(1, "Freed block: 0x7f73c39fe028 (id="..., 91Freed block: 0x7f73c39fe028 (id=1, name=Object 1, value=123.45), time: 0.000000300 seconds

) = 91

write(1, "Allocated block: 0x7f73c39fe028,"..., 77Allocated block: 0x7f73c39fe028, size: 2048 bytes, time: 0.000000300 seconds

) = 77

write(1, "Freed block: 0x7f73c39fe028 (id="..., 91Freed block: 0x7f73c39fe028 (id=2, name=Object 2, value=246.90), time: 0.000000300 seconds

) = 91

write(1, "Allocated block: 0x7f73c39fe028,"..., 77Allocated block: 0x7f73c39fe028, size: 3072 bytes, time: 0.000000200 seconds

) = 77

write(1, "Freed block: 0x7f73c39fe028 (id="..., 91Freed block: 0x7f73c39fe028 (id=3, name=Object 3, value=370.35), time: 0.000000300 seconds

) = 91

munmap(0x7f73c39fe000, 1048576) = 0

munmap(0x7f73c3cff000, 16424) = 0

exit_group(0) = ?

+++ exited with 0 +++

Сравнение алгоритмов

Аллокатор с обычным списком свободных блоков:

Выделен блок: 0x7f740795a010, размер: 1024 байт, время: 0.00000010000 секунд

Освобожден блок: 0x7f740795a010, время: 0.000000100 секунд

Выделен блок: 0x7f740795a420, размер: 2048 байт, время: 0.00000010000 секунд

Освобожден блок: 0x7f740795a420, время: 0.000000100 секунд

Выделен блок: 0x7f740795ac30, размер: 3072 байт, время: 0.00000240000 секунд

Освобожден блок: 0x7f740795ac30, время: 0.000000000 секунд

Аллокатор на основе алгоритма Мак-Кьюзика-Кэрелса:

Выделен блок: 0x7fd39939d028, размер: 1024 байт, время: 0.00000010000 секунд

Освобожден блок: 0x7fd39939d028, время: 0.000000000 секунд

Выделен блок: 0x7fd39939d028, размер: 2048 байт, время: 0.00000010000 секунд

Освобожден блок: 0x7fd39939d028, время: 0.000000100 секунд

Выделен блок: 0x7fd39939d028, размер: 3072 байт, время: 0.00000000000 секунд

Освобожден блок: 0x7fd39939d028, время: 0.000000000 секунд

Аллокатор с обычным списком свободных блоков: эффективно работает для небольших блоков данных, но неэффективен при фрагментации.

Аллокатор на основе алгоритма Мак-Кьюзика-Кэрелса: лучше управляется с памятью, предотвращаю фрагментацию. Работает быстрее для больших блоков данных. Использование памяти с учетом размера и выравнивания, что снижает вероятность ошибок при работе с памятью.

Вывод

В ходе написания данной лабораторной работы я узнал об устройстве аллокаторов а также научился использовать динамические библиотеки.