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: Тимофеева И.А.

Преподаватель: Бахарев В.Д.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата: 25.12.2024

Москва, 2024

**Постановка задачи**

**Требуется создать две динамические библиотеки, реализующие два аллокатора, соответственно. Библиотеки загружаются в память с помощью интерфейса ОС (dlopen / LoadLibrary) для работы с динамическими библиотеками. Выбор библиотеки, реализующей аллокатор, осуществляется чтением первого аргумента при запуске программы (argv[1]). Этот аргумент должен содержать путь до динамической библиотеки (относительный или абсолютный).**

**Если аргумент не передан или по переданному пути библиотеки не оказалось, то указатели на функции, реализующие API аллокатора ниже, должны быть присвоены функциям, которые оборачивают системный аллокатор ОС (mmap / VirtualAlloc) в этот API. Эти аварийные оберточные функции должны быть реализованы внутри программы, которая загружает динамические библиотеки (см. пример на GitHub Gist).**

**Каждый аллокатор памяти должен иметь функции аналогичные стандартным функциям malloc и free (realloc, опционально). Перед работой каждый аллокатор инициализируется свободными страницами памяти, выделенными стандартными средствами ядра (mmap / VirtualAlloc). Необходимо самостоятельно разработать стратегию тестирования для определения ключевых характеристик аллокаторов памяти. При тестировании нужно свести к минимуму потери точности из-за накладных расходов при измерении ключевых характеристик, описанных выше.**

**Вариант 3 - Списки свободных блоков (первое подходящее) и алгоритм двойников**

**Общий метод и алгоритм решения**

Использованные системные вызовы:

● int shm\_open(const char \*name, int oflag, mode\_t mode); – функция устанавливает соединение между объектом разделяемой памяти и файловым дескриптором.

● int shm\_unlink(const char \*name); – функция удаляет объект разделяемой области памяти.

● void \*mmap (void \*address, size\_t length, int protect, int flags, int filedes, off\_t offset); - функция для сопоставления адресного пространства процесса с файлами или устройствами.

● int \*munmap (void \*addr, size\_t len); - функция удаляет сопоставление страниц памяти в заданном диапазоне адресов процесса.

● int ftruncate(int fd, off\_t length); – функция задает размер разделяемой памяти.

● void \* dlopen(const char \*filename, int flag); – функция загружает динамическую библиотеку, имя которой указано в строке filename, и возвращает прямой указатель на начало динамической библиотеки;

● void \*dlsym(void \*handle, char \*symbol); – функция ищет значение символа в открытой динамически загружаемой библиотеке. Она использует указатель на динамическую библиотеку, возвращаемую dlopen(), и оканчивающееся нулём символьное имя, а затем возвращает адрес;

● void srand(unsigned int seed); – функция инициализирует генератор псевдослучайных чисел;

● int fputs(const char \*s, FILE \*stream); – функция записывает в файл строку, то есть набор символов, который завершается символом '\0'

● int fprintf( FILE \* stream, const char \* format, ... ); – функция выполняет форматированный вывод в поток. Записывает в указанный поток последовательность символов в формате, указанном аргументом format;

● void perror(const char \*s) – функция выводит сообщение об ошибке в поток stderr.

Программа реализует алгоритм организации аллокатора памяти.

При инициализации аллокатора программа резервирует область памяти с помощью функции **mmap**. Адрес этой области памяти формируется с помощью функции **shm\_open**. Функция возвращает файловый дескриптор для ссылки на этот объект общей памяти. Этот дескриптор в дальнейшем используется функцией **mmap** для сопоставления адресного пространства процесса с файлами или устройствами. После создания общего объекта памяти задается размер разделяемой памяти вызовом **ftruncate**. На входе у функции передаётся файловый дескриптор объекта и необходимый размер. Эти действия реализованы в методе **alloc\_init**.

Затем адрес выделенной памяти передается в функцию **allocator\_create**. Указанная функция записывает в заголовок блока памяти переменную структуры **memory\_block**, которая хранит размер этого блока и признак занятости. Также инициализируется массив указателей **block\_list** для работы алгоритма двойников.

Для выделения блоков памяти внутри зарезервированной области памяти используется функция **allocator\_alloc**.

При автономной работе программы (без подключения внешей библиотеки) выполняется выделение блоков памяти по алгоритму свободных блоков (первое подходящее).

Если программа запускается с аргументом argv[1], то используется алгоритм двойников (близнецов), который реализован внутри подгружаемой библиотеки с названием argv[1].

Программа в цикле с помощью метода allocator\_alloc выделяет блоки памяти случайного размера. При алгоритме свободных блоков программа последовательно перебирает и находит свободную область нужно размера и занимает его.

В случае алгоритма двойников все выделяемые блоки имеют размер в степень двойки - 4, 8, 16, 32, … байт;

* размер изначального единого свободного блока (при старте программы) тоже равен степени 2;
* свободные блоки одного размера объединены в список;
* когда требуется выделить блок размера S:

1. находим минимальную степень 2 больше S;
2. проверяем наличие свободных страниц такого размера, отдаём если есть;
3. если нет, находим минимальный блок большего размера и рекурсивно расщепляем его до нужного размера. Получившиеся “обрезки” пополняют списки свободных блоков.

* при освобождении блока:

1. находим размер блока;
2. проверяем, свободен ли второй блок (buddy) и если свободен, объединяем их и рекурсивно проверяем вышестоящий (уже объединённый) блок.

* размер и флаг занятости хранится в начале блока.

Выполнен анализ и сравнительная характеристика указанных алгоритмов.

1. Фактор использования

Алгоритм двойников потребляет в среднем больше памяти (т.н. внутренняя фрагментация за счет округления размера до степени двойки), но гораздо меньше подвержен внешней фрагментации.

1. Скорость выделения блоков.

В алгоритме свободных блоков скорость выделения памяти ниже, чем при методе двойников.

1. Скорость освобождения блоков.

В алгоритме свободных блоков скорость выделения памяти ниже, чем при методе двойников.

1. Простота использования аллокатора.

Алгоритм реализации метода двойников сложнее, чем простой перебор свободных блоков памяти, но за счет адресации блоков памяти в виде массива указателей увеличивается скорость и эффективность использования указнного метода

**Код программы**

**alloc\_test.c**

#include <unistd.h>

#include <sys/types.h>

#include <sys/mman.h>

#include <fcntl.h>

#include <stdio.h>

#include <string.h>

#include <stddef.h>

#include <stdint.h>

#include <stdlib.h>

#include <time.h>

#include <math.h>

#include <dlfcn.h> *// dlopen, dlsym, dlclose, RTLD\_\**

#include "lib.c"

#include "library.h"

static alloc\_func \*falloc;

*// инициализация аллокатора на памяти memory размера size*

Allocator \***allocator\_create**(void \*const memory, const size\_t size)

{

*//Allocator a = {memory, size, 0};*

    static Allocator alloc;

    alloc.mem = memory;

    alloc.totalSize = size;

    alloc.offset = 0;

    memory\_block\* current\_location\_mcb = (memory\_block\*)memory;

    current\_location\_mcb->is\_available = 1;

    current\_location\_mcb->size = size;

    current\_location\_mcb->block\_num = 0;

    current\_location\_mcb->next = NULL;

    int i;

    for (i = 0; i<POWR\_N-1; i++) {

        block\_list[i] = NULL;

    }

    block\_list[POWR\_N] = memory;

    return &alloc;

}

*// Инициализации памяти для Allocator-а*

void\* **alloc\_init**(const size\_t mem\_size)

{

    int shm;

    if ( (shm = **shm\_open**(SHARED\_MEMORY\_OBJECT\_NAME, O\_CREAT|O\_RDWR, 0777)) == -1 ) {

**perror**("shm\_open");

        return NULL;

    }

*// trunc file to Size bytes*

    if ( **ftruncate**(shm, mem\_size) == -1 ) {

**perror**("ftruncate");

        return NULL;

    }

    void \*shm\_address = **mmap**(NULL, mem\_size, PROT\_READ | PROT\_WRITE, MAP\_SHARED, shm, 0);

    if (shm\_address == MAP\_FAILED) {

**perror**("Mapping Failed");

        return NULL;

    }

**close**(shm);

    return shm\_address;

}

*// деинициализация структуры аллокатора*

void **allocator\_destroy**(Allocator \*const allocator)

{

    int err = **munmap**(allocator->mem, allocator->totalSize);

**shm\_unlink**(SHARED\_MEMORY\_OBJECT\_NAME);

    if (err) {

**fprintf**(stdout, "Unmapping Failed, %d\n", err);

        return;

    }

    allocator->mem = NULL;

    allocator->totalSize = 0;

    allocator->offset = 0;

}

*// выделение памяти аллокатором памяти размера size (non-aligned allocation from linear buffer)*

void\* **allocator\_alloc**(Allocator \*const allocator, const size\_t size)

{

    if(!allocator || !size || allocator->totalSize-allocator->offset < size)

        return NULL;

    uint32\_t newOffset = allocator->offset + size + sizeof(memory\_block);

    if(newOffset <= allocator->totalSize) {

        void\* ptr = allocator->mem + allocator->offset;

        allocator->offset = newOffset;

        memory\_block\* current\_location\_mcb = (memory\_block\*)ptr;

        current\_location\_mcb->is\_available = 0;

        current\_location\_mcb->size = size;

        current\_location\_mcb->block\_num = blocknum;

        blocknum++;

        return ptr;

    }

    return NULL; */\* out of memory \*/*

}

*// возвращает выделенную память аллокатору*

void **allocator\_free**(Allocator \*const allocator, void \*const memory)

{

    void \*current\_location = allocator->mem;

    memory\_block\* current\_location\_mcb = (memory\_block\*)current\_location;

    while (current\_location < (void\*)(allocator->mem + allocator->offset)) {

        current\_location\_mcb->is\_available = 1;

        current\_location += current\_location\_mcb->size + sizeof(memory\_block);

        current\_location\_mcb = (memory\_block\*)current\_location;

    }

}

*// NOTE: Functions stubs will be used, if library failed to load*

*// NOTE: Stubs are better than NULL function pointers,*

*//       you don't need to check for NULL before calling a function*

static float **func\_impl\_stub**(float x) {

    (void)x; *// NOTE: Compiler will warn about unused parameter otherwise*

    return 0.0f;

}

int **main**(int argc, char \*\*argv) {

    char \*error;

    void \*library = **dlopen**(argv[1], RTLD\_LOCAL | RTLD\_NOW);

    if (library != NULL)

    {

        falloc = **dlsym**(library, "twins\_allocator\_alloc");

        if (falloc == NULL) {

            const char msg[] = "warning: failed to find alloc function implementation\n";

**write**(STDERR\_FILENO, msg, sizeof(msg));

            falloc = allocator\_alloc;

        }

    }

    else

    {

**fputs**(**dlerror**(), stderr);

        const char msg[] = "\nwarning: library failed to load, trying standard implemntations\n";

**write**(STDERR\_FILENO, msg, sizeof(msg));

        falloc = allocator\_alloc;

    }

*// allocate a 1 MB region (1\*1024\*1024)*

    size\_t size\_of\_mem = **Pow2**(POWR\_N);

    void \*memory = **alloc\_init**(size\_of\_mem);

    Allocator \*alloc = **allocator\_create**(memory, size\_of\_mem);

**fprintf**(stdout, "Size of memory %ld\n", size\_of\_mem);

**fprintf**(stdout, "Address of memory %p\n", alloc->mem);

    int mem\_part;

    size\_t sum\_mem = 0;

    size\_t free\_mem = 0;

    int block\_count = 0;

    void \*next = (void\*)1;

**srand**(**time**(NULL));

    while (next != NULL) {

        mem\_part = (**rand**() % 100 + 1)\*32;

        next = **falloc**(alloc, mem\_part);

        if (next != NULL) {

            memory\_block\* current\_location\_mcb = (memory\_block\*)next;

            sum\_mem += current\_location\_mcb->size;

            block\_count++;

        }

    }

*// calc free memory*

    int i;

    for (i = 0; i<=POWR\_N; i++) {

        if (block\_list[i] != NULL) {

            void\* cur\_ptr = block\_list[i];

            while (cur\_ptr != NULL) {

                memory\_block\* current\_location\_mcb = (memory\_block\*)cur\_ptr;

                free\_mem += current\_location\_mcb->size;

                current\_location\_mcb = current\_location\_mcb->next;

                if (cur\_ptr == (void\*)current\_location\_mcb) break;

                cur\_ptr = (void\*)current\_location\_mcb;

            }

        }

    }

**fprintf**(stdout, "Used memory: %ld\n", sum\_mem);

**fprintf**(stdout, "Free memory: %ld\n", free\_mem);

**fprintf**(stdout, "Total memory: %ld\n", sum\_mem+free\_mem);

**fprintf**(stdout, "Count of blocks: %d\n", block\_count);

**allocator\_free**(alloc, memory);

**allocator\_destroy**(alloc);

    return 0;

}

**lib.c**

#include <unistd.h>

#include <stdint.h>

#include <stdio.h>

#include "library.h"

*// NOTE: MSVC compiler does not export symbols unless annotated*

#ifdef **\_MSC\_VER**

#define **EXPORT** **\_\_declspec**(dllexport)

#else

#define **EXPORT**

#endif

*// массив для хранения очередей указателей на свободные блоки*

int\* block\_list[POWR\_N+1];

*// выделение памяти аллокатором памяти размера size методом близнецов (non-aligned allocation from linear buffer)*

EXPORT void\* **twins\_allocator\_alloc**(Allocator\* const allocator, const size\_t size)

{

    if(!allocator || !size)

        return NULL;

    int i;

    static int start = 0;

    if (start==0)

    {

        for (i = 0; i<POWR\_N-1; i++) {

            block\_list[i] = NULL;

        }

        block\_list[POWR\_N] = (void\*)allocator->mem;

        start = 1;

    }

*// определим размер выделяемого блока с учетом заголовка*

    int power = **NearestPow2**(size + sizeof(memory\_block));

    int blockLength = **Pow2**(power);

    int tmp\_power = power;

    if (power >= POWR\_N) {

**fprintf**(stdout, "Max power: %d\n", POWR\_N);

**perror**("Out of memory");

        return NULL;

    }

    void\* ptr = block\_list[power];

**fprintf**(stdout, "Power for block: %d\n", power);

**fprintf**(stdout, "Power block size: %d\n", blockLength);

**fprintf**(stdout, "Power block address: %p\n", block\_list[power]);

    if (block\_list[power] == NULL) {

        do {

            tmp\_power++;

            ptr = block\_list[tmp\_power];

*//fprintf(stdout, "Tmp\_power block address: %p\n", ptr);*

        } while (tmp\_power < POWR\_N && block\_list[tmp\_power] == NULL);

        memory\_block\* current\_location\_mcb = (memory\_block\*)ptr;

        void\* cur\_ptr = ptr;

        while (current\_location\_mcb != NULL)

        {

            if (current\_location\_mcb->is\_available > 0) break;

            current\_location\_mcb = current\_location\_mcb->next;

        }

        cur\_ptr = (void\*)current\_location\_mcb;

        if (tmp\_power==POWR\_N && cur\_ptr == NULL) return NULL;

        if (current\_location\_mcb != NULL) {

*// отметим найденный блок как занятый и разделим его пополам*

*//current\_location\_mcb = (memory\_block\*)cur\_ptr;*

*//current\_location\_mcb->is\_available = 1;*

            int tmp\_size = current\_location\_mcb->size;

*// надо убрать текущий блок из списка свободных блоков этого размера*

            void\* ppp = block\_list[tmp\_power];

            memory\_block\* ccc = (memory\_block\*)ppp;

            if (ppp = cur\_ptr)

                block\_list[tmp\_power] = (void\*)ccc->next;

            else

                ccc->next = NULL;

            do {

                tmp\_power--;

                tmp\_size /= 2;

                current\_location\_mcb = (memory\_block\*)cur\_ptr;

                current\_location\_mcb->size = tmp\_size;

                current\_location\_mcb->is\_available = 0;

                current\_location\_mcb->next = NULL;

                current\_location\_mcb = (memory\_block\*)(cur\_ptr+tmp\_size);

                current\_location\_mcb->size = tmp\_size;

                current\_location\_mcb->is\_available = 1;

                current\_location\_mcb->next = (void\*)block\_list[tmp\_power];

                block\_list[tmp\_power] = cur\_ptr+tmp\_size;

            } while (tmp\_power > power);

            return cur\_ptr;

        }

    }

    else

    {

        memory\_block\* current\_location\_mcb = (memory\_block\*)ptr;

        if (current\_location\_mcb != NULL) {

            current\_location\_mcb->is\_available = 0;

            current\_location\_mcb->size = blockLength;

            current\_location\_mcb->block\_num = blocknum;

*// надо убрать текущий блок из списка свободных блоков этого размера*

            void\* ppp = block\_list[power];

            memory\_block\* ccc = (memory\_block\*)ppp;

            if (ppp = ptr)

                block\_list[power] = (void\*)ccc->next;

            else

                ccc->next = NULL;

        }

        return ptr;

    }

    return NULL; */\* out of memory \*/*

}

**library.h**

#ifndef **\_\_LIBRARY\_H**

#define **\_\_LIBRARY\_H**

#define **SHARED\_MEMORY\_OBJECT\_NAME** "my\_shared\_memory"

#define **POWR\_N** 20 *// размерность выделяемой памяти 2^POWR\_N*

*// Структура Memory Block*

typedef struct **mem\_block** {

    int is\_available;

    int size;

    int block\_num;

    struct **mem\_block**\* next;

} **memory\_block**;

*// Header structure for linear buffer*

typedef struct **\_Allocator** {

    uint8\_t \*mem; */\*!< Pointer to buffer memory. \*/*

    uint32\_t totalSize; */\*!< Total size in bytes. \*/*

    uint32\_t offset; */\*!< Offset. \*/*

} **Allocator**;

int **Pow2**(int n) {

    return 1<<n;

}

int **NearestPow2**(int x) {

    int y = 1;

    int pwr = 0;

    while (y < x) {

        y \*= 2;

        pwr++;

    }

    return pwr;

}

int blocknum = 0;

typedef void\* **alloc\_func**(Allocator\* const allocator, const size\_t size);

#endif *// \_\_LIBRARY\_H*

**Протокол работы программы**

**Тестирование:**

irina@Irina-VivoBook:~/Prog/Prog\_C/Kurs2/LabOS/Lab04$ gcc -o lib.so -fPIC -shared lib.c

irina@Irina-VivoBook:~/Prog/Prog\_C/Kurs2/LabOS/Lab04$ gcc -o prog -ldl alloc\_test.c

irina@Irina-VivoBook:~/Prog/Prog\_C/Kurs2/LabOS/Lab04$ ./prog

warning: failed to find alloc function implementation

Size of memory 1048576

Address of memory 0x7a697283c000

Used memory: 1032416

Free memory: 1440

Total memory: 1033856

Count of blocks: 670

irina@Irina-VivoBook:~/Prog/Prog\_C/Kurs2/LabOS/Lab04$ ./prog ./lib.so

Size of memory 1048576

Address of memory 0x70fd9cd00000

Power for block: 10

Power block size: 1024

Power block address: (nil)

Power for block: 12

Power block size: 4096

Power block address: 0x70fd9cd01000

Power for block: 11

Power block size: 2048

Power block address: 0x70fd9cd00800

Power for block: 12

Power block size: 4096

Power block address: (nil)

Power for block: 12

Power block size: 4096

Power block address: 0x70fd9cd03000

Power for block: 10

Power block size: 1024

Power block address: 0x70fd9cd00400

Power for block: 12

Power block size: 4096

Power block address: 0x70fd9cdff000

Power for block: 10

Power block size: 1024

Power block address: (nil)

Power for block: 11

Power block size: 2048

Power block address: (nil)

Used memory: 1047232

Free memory: 1024

Total memory: 1048256

Count of blocks: 416

**Strace**

irina@Irina-VivoBook:~/Prog/Prog\_C/Kurs2/LabOS/Lab04$ strace -f ./prog

**execve**("./prog", ["./prog"], 0x7ffdce81e8a8 */\* 46 vars \*/*) = 0

**brk**(NULL)                               = 0x5f8441f48000

**arch\_prctl**(0x3001 */\* ARCH\_??? \*/*, 0x7ffe8afeca20) = -1 **EINVAL** (Недопустимый аргумент)

**mmap**(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x785415a54000

**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

**newfstatat**(3, "", {st\_mode=S\_IFREG|0644, st\_size=64071, ...}, AT\_EMPTY\_PATH) = 0

**mmap**(NULL, 64071, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x785415a44000

**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\0P\237\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

**pread64**(3, "\4\0\0\0 \0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0"..., 48, 848) = 48

**pread64**(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0I\17\357\204\3$\f\221\2039x\324\224\323\236S"..., 68, 896) = 68

**newfstatat**(3, "", {st\_mode=S\_IFREG|0755, st\_size=2220400, ...}, AT\_EMPTY\_PATH) = 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, 2264656, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x785415800000

**mprotect**(0x785415828000, 2023424, PROT\_NONE) = 0

**mmap**(0x785415828000, 1658880, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x28000) = 0x785415828000

**mmap**(0x7854159bd000, 360448, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1bd000) = 0x7854159bd000

**mmap**(0x785415a16000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x215000) = 0x785415a16000

**mmap**(0x785415a1c000, 52816, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x785415a1c000

**close**(3)                                = 0

**mmap**(NULL, 12288, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x785415a41000

**arch\_prctl**(ARCH\_SET\_FS, 0x785415a41740) = 0

**set\_tid\_address**(0x785415a41a10)         = 3392

**set\_robust\_list**(0x785415a41a20, 24)     = 0

**rseq**(0x785415a420e0, 0x20, 0, 0x53053053) = 0

**mprotect**(0x785415a16000, 16384, PROT\_READ) = 0

**mprotect**(0x5f8441049000, 4096, PROT\_READ) = 0

**mprotect**(0x785415a8e000, 8192, PROT\_READ) = 0

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

**munmap**(0x785415a44000, 64071)           = 0

**getrandom**("\xd5\x43\x28\xf5\x8a\x20\xaa\xd6", 8, GRND\_NONBLOCK) = 8

**brk**(NULL)                               = 0x5f8441f48000

**brk**(0x5f8441f69000)                     = 0x5f8441f69000

**write**(2, "warning: failed to find alloc fu"..., 55warning: failed to find alloc function implementation

) = 55

**openat**(AT\_FDCWD, "/dev/shm/my\_shared\_memory", O\_RDWR|O\_CREAT|O\_NOFOLLOW|O\_CLOEXEC, 0777) = 3

**ftruncate**(3, 1048576)                   = 0

**mmap**(NULL, 1048576, PROT\_READ|PROT\_WRITE, MAP\_SHARED, 3, 0) = 0x785415700000

**close**(3)                                = 0

**newfstatat**(1, "", {st\_mode=S\_IFCHR|0620, st\_rdev=**makedev**(0x88, 0), ...}, AT\_EMPTY\_PATH) = 0

**write**(1, "Size of memory 1048576\n", 23Size of memory 1048576

) = 23

**write**(1, "Address of memory 0x785415700000"..., 33Address of memory 0x785415700000

) = 33

**write**(1, "Used memory: 1031936\n", 21Used memory: 1031936

)  = 21

**write**(1, "Free memory: 256\n", 17Free memory: 256

)      = 17

**write**(1, "Total memory: 1032192\n", 22Total memory: 1032192

) = 22

**write**(1, "Count of blocks: 649\n", 21Count of blocks: 649

)  = 21

**munmap**(0x785415700000, 1048576)         = 0

**unlink**("/dev/shm/my\_shared\_memory")     = 0

**exit\_group**(0)                           = ?

+++ exited with 0 +++

irina@Irina-VivoBook:~/Prog/Prog\_C/Kurs2/LabOS/Lab04$ strace -f ./prog ./lib.so

**execve**("./prog", ["./prog", "./lib.so"], 0x7ffcf621d390 */\* 46 vars \*/*) = 0

**brk**(NULL)                               = 0x556dc8839000

**arch\_prctl**(0x3001 */\* ARCH\_??? \*/*, 0x7ffdb737a630) = -1 **EINVAL** (Недопустимый аргумент)

**mmap**(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x781fa5be5000

**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

**newfstatat**(3, "", {st\_mode=S\_IFREG|0644, st\_size=64071, ...}, AT\_EMPTY\_PATH) = 0

**mmap**(NULL, 64071, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x781fa5bd5000

**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\0P\237\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

**pread64**(3, "\4\0\0\0 \0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0"..., 48, 848) = 48

**pread64**(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0I\17\357\204\3$\f\221\2039x\324\224\323\236S"..., 68, 896) = 68

**newfstatat**(3, "", {st\_mode=S\_IFREG|0755, st\_size=2220400, ...}, AT\_EMPTY\_PATH) = 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, 2264656, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x781fa5800000

**mprotect**(0x781fa5828000, 2023424, PROT\_NONE) = 0

**mmap**(0x781fa5828000, 1658880, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x28000) = 0x781fa5828000

**mmap**(0x781fa59bd000, 360448, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1bd000) = 0x781fa59bd000

**mmap**(0x781fa5a16000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x215000) = 0x781fa5a16000

**mmap**(0x781fa5a1c000, 52816, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x781fa5a1c000

**close**(3)                                = 0

**mmap**(NULL, 12288, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x781fa5bd2000

**arch\_prctl**(ARCH\_SET\_FS, 0x781fa5bd2740) = 0

**set\_tid\_address**(0x781fa5bd2a10)         = 3461

**set\_robust\_list**(0x781fa5bd2a20, 24)     = 0

**rseq**(0x781fa5bd30e0, 0x20, 0, 0x53053053) = 0

**mprotect**(0x781fa5a16000, 16384, PROT\_READ) = 0

**mprotect**(0x556dc8560000, 4096, PROT\_READ) = 0

**mprotect**(0x781fa5c1f000, 8192, PROT\_READ) = 0

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

**munmap**(0x781fa5bd5000, 64071)           = 0

**getrandom**("\xa9\xd3\x9e\x91\x63\xc2\x15\xb0", 8, GRND\_NONBLOCK) = 8

**brk**(NULL)                               = 0x556dc8839000

**brk**(0x556dc885a000)                     = 0x556dc885a000

**openat**(AT\_FDCWD, "./lib.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

**newfstatat**(3, "", {st\_mode=S\_IFREG|0775, st\_size=15856, ...}, AT\_EMPTY\_PATH) = 0

**getcwd**("/home/irina/Prog/Prog\_C/Kurs2/LabOS/Lab04", 128) = 42

**mmap**(NULL, 16688, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x781fa5be0000

**mmap**(0x781fa5be1000, 4096, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1000) = 0x781fa5be1000

**mmap**(0x781fa5be2000, 4096, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x2000) = 0x781fa5be2000

**mmap**(0x781fa5be3000, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x2000) = 0x781fa5be3000

**close**(3)                                = 0

**mprotect**(0x781fa5be3000, 4096, PROT\_READ) = 0

**openat**(AT\_FDCWD, "/dev/shm/my\_shared\_memory", O\_RDWR|O\_CREAT|O\_NOFOLLOW|O\_CLOEXEC, 0777) = 3

**ftruncate**(3, 1048576)                   = 0

**mmap**(NULL, 1048576, PROT\_READ|PROT\_WRITE, MAP\_SHARED, 3, 0) = 0x781fa5ad2000

**close**(3)                                = 0

**newfstatat**(1, "", {st\_mode=S\_IFCHR|0620, st\_rdev=**makedev**(0x88, 0), ...}, AT\_EMPTY\_PATH) = 0

**write**(1, "Size of memory 1048576\n", 23Size of memory 1048576

) = 23

**write**(1, "Address of memory 0x781fa5ad2000"..., 33Address of memory 0x781fa5ad2000

) = 33

**write**(1, "Power for block: 12\n", 20Power for block: 12

)   = 20

**write**(1, "Power block size: 4096\n", 23Power block size: 4096

) = 23

**write**(1, "Power block address: (nil)\n", 27Power block address: (nil)

) = 27

**write**(1, "Power for block: 11\n", 20Power for block: 11

)   = 20

**write**(1, "Power block size: 2048\n", 23Power block size: 2048

) = 23

**write**(1, "Power block address: (nil)\n", 27Power block address: (nil)

) = 27

**write**(1, "Power for block: 10\n", 20Power for block: 10

)   = 20

**write**(1, "Power block size: 1024\n", 23Power block size: 1024

) = 23

**write**(1, "Power block address: (nil)\n", 27Power block address: (nil)

) = 27

**write**(1, "Power for block: 12\n", 20Power for block: 12

)   = 20

**write**(1, "Power block size: 4096\n", 23Power block size: 4096

) = 23

**write**(1, "Power block address: 0x781fa5bbb"..., 36Power block address: 0x781fa5bbb000

) = 36

**write**(1, "Power for block: 12\n", 20Power for block: 12

)   = 20

**write**(1, "Power block size: 256\n", 22Power block size: 256

) = 22

**write**(1, "Power block address: 0x781fa5bba"..., 36Power block address: 0x781fa5bbaf00

) = 36

**write**(1, "Power for block: 12\n", 20Power for block: 12

)   = 20

**write**(1, "Power block size: 4096\n", 23Power block size: 4096

) = 23

**write**(1, "Power block address: (nil)\n", 27Power block address: (nil)

) = 27

**write**(1, "Power for block: 12\n", 20Power for block: 12

)   = 20

**write**(1, "Power block size: 4096\n", 23Power block size: 4096

) = 23

**write**(1, "Power block address: 0x781fa5bd1"..., 36Power block address: 0x781fa5bd1000

) = 36

**write**(1, "Power for block: 11\n", 20Power for block: 11

)   = 20

**write**(1, "Power block size: 2048\n", 23Power block size: 2048

) = 23

**write**(1, "Power block address: 0x781fa5bcf"..., 36Power block address: 0x781fa5bcf800

) = 36

**write**(1, "Power for block: 12\n", 20Power for block: 12

)   = 20

**write**(1, "Power block size: 4096\n", 23Power block size: 4096

) = 23

**write**(1, "Power block address: (nil)\n", 27Power block address: (nil)

) = 27

**write**(1, "Used memory: 1047872\n", 21Used memory: 1047872

)  = 21

**write**(1, "Free memory: 4096\n", 18Free memory: 4096

)     = 18

**write**(1, "Total memory: 1051968\n", 22Total memory: 1051968

) = 22

**write**(1, "Count of blocks: 440\n", 21Count of blocks: 440

)  = 21

**munmap**(0x781fa5ad2000, 1048576)         = 0

**unlink**("/dev/shm/my\_shared\_memory")     = 0

**exit\_group**(0)                           = ?

+++ exited with 0 +++

**Вывод**

**В ходе лабораторной работы я научилась создавать аллокаторы памяти и динамические библиотеки, а также подключать и использовать их. Таким образом, программа иллюстрирует использование базовых системных вызовов для аллокации блоков памяти и их использования.**