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: Мишин С.А.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 02.01.25

Москва, 2024

Постановка задачи

Вариант 9.

Исследовать два аллокатора памяти: необходимо реализовать два алгоритма (**Списки свободных блоков (наиболее подходящее)** и **алгоритм двойников**) аллокации памяти и сравнить их по следующим характеристикам:

- Фактор использования
- Скорость выделения блоков
- Скорость освобождения блоков
- Простота использования аллокатора

Требуется создать две динамические библиотеки, реализующие два аллокатора, соответственно.

Общий метод и алгоритм решения

Использованные системные вызовы:

- int write(int filedes, const void *buf, size_t nbyte); – записывает данные из буфера в файловый дескриптор.
- int read(int filedes, void *buf, size_t nbyte); – читает данные из файлового дескриптора в буфер.
- void* mmap(void *addr, size_t len, int prot, int flags, int fd, off_t offset); – выделяет память, размещает файл или устройство в память процесса.
- void* munmap(void *addr, size_t len); – удаляет размещение.

В работе реализованы две динамические библиотеки, экспортирующие функции аллокатора. Программа загружает указанную библиотеку и разрешает имена функций. Если библиотеку загрузить не удалось, или одна из функций не найдена, программа использует fallback-аллокатор, использующий mmap/munmap.

Библиотека «alloc-1» реализует алгоритм списка свободных блоков. При создании аллокатора выделяется один большой блок размером с доступную память. При выделении ищется блок с наиболее подходящим размером: ближайшим к выделяемому размеру. Блок удаляется из списка и разбивается, чтобы создать новый свободный. При освобождении памяти блок возвращается в список первым элементом, а блоки, находящиеся в памяти рядом, сливаются. Такой метод аллокации прост в реализации, но при длительной работе может возникнуть значительная фрагментация памяти.

В «alloc-2» реализован алгоритм двойников («Buddy allocator»). Аллокатор требует, чтобы доступная ему память была степенью двойки, иначе не получится делить блоки. Внутри хранится список свободных блоков для каждого порядка степеней двойки. При выделении памяти определяется минимальный порядок, достаточный для размещения запроса, и из списка свободных блоков извлекается первый подходящий. Если блок оказывается больше, чем требуется, он разделяется на два равных, один из которых возвращается в список меньшего порядка, а другой используется для размещения данных.

При освобождении памяти проверяется, находится ли соседний блок («buddy») в списке свободных блоков. Если соседний блок найден, они объединяются в блок более высокого порядка. Процесс повторяется, пока это возможно. Алгоритм обеспечивает быстрое выделение и

освобождение памяти. Однако, только блоки размерами, кратнымы степеням двойки, могут быть выделены, из-за чего остаётся неиспользованная память.

Подход тестирования

Для тестирования производительности аллокаторов программа реализует два подхода.

В первом программа выделяет 512 блоков со случайным размером до 1 КБ. Случайным образом освобождается половина блоков, затем вместо них выделяются новые. Такой подход позволяет выявить ошибки в реализации.

Второй подход заключается в неком fuzz-тестировании аллокаторов. Программа делает 200000 итераций, в которых она либо выделяет новый блок, либо освобождает старый. Такой подход позволяет сделать более точный замер производительности.

Результаты тестирования

	Free List	Buddy	Fallback
alloc	0.002933s	0.002492s	0.004760s
free	0.001186s	0.001031s	0.001851s
fuzz	0.524958s	0.250028s	0.476698s

Код программы

alloc-1/alloc.c

```
#define LIBRARY

#include "alloc.h"

#include <stdbool.h>
#include <string.h>

static const size_t MIN_SIZE = 16;

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

struct Allocator {
	/** Pointer to the first free block. */
	Block *head;
	/** Total memory available to the allocator. */
	size_t size;
};

Allocator *allocator_create(void *memory, size_t size) {
```

```
        if (!memory || size < sizeof(Allocator)) return NULL;

        Allocator *allocator = (Allocator *)memory;

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

        // Initialize head (first) block.
        allocator->head->size = size - sizeof(Allocator) - sizeof(Block);
        allocator->head->next = NULL;
        allocator->head->used = false;

        return allocator;
}

void allocator_destroy(Allocator *allocator) {
        if (!allocator) return;
        memset(allocator, 0, allocator->size);
}

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

        // Find the most suitable block (closest to |size|.)
        Block *current = allocator->head, *previous = NULL;
        Block *best = NULL, *prevBest = NULL;

        while (current) {
                if (!current->used && current->size >= size) {
                        if (!best || current->size < best->size) {
                                best = current;
                                prevBest = previous;
                        }
                }

                previous = current;
                current = current->next;
        }

        if (!best) {
                // No block found.
                return NULL;
        }

        // Remove 'best' from the free list.
        if (prevBest) {
                prevBest->next = best->next;
        } else {
                allocator->head = best->next;
        }
```

```
        // If it makes sense to insert a new block (i.e., if we have some space),
        // insert one.
        size_t leftover = best->size - size;

        if (leftover >= MIN_SIZE + sizeof(Block)) {
                Block *newBlock = (Block *)((char *)best + sizeof(Block) + size);
                newBlock->size = leftover - sizeof(Block);
                newBlock->used = false;

                // Insert the block at the top of the list.
                newBlock->next = allocator->head;
                allocator->head = newBlock;
        }

        best->size = size;
        best->used = true;

        return (char *)best + sizeof(Block);
}

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

        Block *block = (Block *)((char *)memory - sizeof(Block));

        // Insert the block at the top of the list.
        block->next = allocator->head;
        block->used = false;
        allocator->head = block;

        // Merge blocks that are right next to each other.
        Block *current = allocator->head;

        while (current && current->next) {
                Block *next = current->next;

                bool canMerge =
                    (Block *)((char *)current + sizeof(Block) + current->size) == next;

                if (canMerge) {
                        current->size += sizeof(Block) + next->size;
                        current->next = next->next;
                } else {
                        current = next;
                }
        }
}
```

alloc-2/alloc.c

```
#define LIBRARY
```

```
#include "alloc.h"

#include <stdbool.h>
#include <unistd.h>

#define MAX_ORDER 30

typedef struct Block {
	/** Linked list pointer for the free list. */
	struct Block *next;
	/** The power-of-two order of this block. */
	size_t order;
} Block;

struct Allocator {
	/** Keep a free list for each order from 0..MAX_ORDER. */
	Block *free_lists[MAX_ORDER + 1];
	/** Start (base) of the entire memory region managed. */
	void *start;
	/** Total size of the memory region. */
	size_t size;
};

/** Returns the smallest power-of-two order that can fit 'size' bytes. */
static size_t get_order(size_t size) {
	size_t order = 0;
	size_t blockSize = 1;

	while (blockSize < size) {
		blockSize <<= 1;
		++order;
	}

	return order;
}

/** Computes 2^order. */
static size_t order_to_block_size(size_t order) { return (size_t)1 << order; }

/** Returns an offset within the allocator's memory. */
static size_t get_offset(Allocator *allocator, void *ptr) {
	return (size_t)((char *)ptr - (char *)allocator->start);
}

/** Converts from offset to actual pointer in the memory region */
static void *offset_to_ptr(Allocator *allocator, size_t offset) {
	return (void *)((char *)allocator->start + offset);
}
```

```
/** Finds buddy of a given block. */
static void *find_buddy(Allocator *allocator, void *block_ptr, size_t order) {
	size_t offset = get_offset(allocator, block_ptr);
	size_t buddyOffset = offset ^ order_to_block_size(order);

	return offset_to_ptr(allocator, buddyOffset);
}

/** Insert a block into the free list for a given order. */
static void add_to_free_list(Allocator *allocator, Block *block, size_t order) {
	block->order = order;
	block->next = allocator->free_lists[order];

	allocator->free_lists[order] = block;
}

/** Remove a block from the free list. */
static void remove_from_free_list(Allocator *allocator, Block *block,
				  size_t order) {
	Block *previous = NULL;
	Block *current = allocator->free_lists[order];

	while (current) {
		if (current == block) {
			// Found it, remove it from the list
			if (previous) {
				previous->next = current->next;
			} else {
				allocator->free_lists[order] = current->next;
			}
			return;
		}
		previous = current;
		current = current->next;
	}
}

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

	Allocator *allocator = (Allocator *)memory;

	/* Initialize all free lists to NULL */
	for (int i = 0; i <= MAX_ORDER; i++) {
		allocator->free_lists[i] = NULL;
	}

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

	size_t order = get_order(allocator->size);
```

```
        size_t exactBlockSize = order_to_block_size(order);

        if (exactBlockSize != allocator->size) {
                const char msg[] =
                    "warn: size of memory region is not a power of two, might cause "
                    "issues\n";
                write(STDOUT_FILENO, msg, sizeof(msg) - 1);
        }

        // Create one free block covering the entire buddy region.
        Block *block = (Block *)allocator->start;

        block->order = order;
        block->next = NULL;

        allocator->free_lists[order] = block;

        return allocator;
}

void allocator_destroy(Allocator *allocator) { (void)allocator; }

void *allocator_alloc(Allocator *allocator, size_t size) {
        if (!size) return NULL;

        // Figure out which order can fit `size`.
        size_t neededOrder = get_order(size + sizeof(Block));

        // Find a free block at or above that order, then split down if needed.
        size_t currentOrder = neededOrder;
        while (currentOrder <= MAX_ORDER &&
               allocator->free_lists[currentOrder] == NULL) {
                currentOrder++;
        }
        if (currentOrder > MAX_ORDER) {
                // No block big enough.
                return NULL;
        }

        // Remove that block from the free list.
        Block *block = allocator->free_lists[currentOrder];
        remove_from_free_list(allocator, block, currentOrder);

        // Split down to the needed order
        while (currentOrder > neededOrder) {
                currentOrder--;

                size_t halfSize = order_to_block_size(currentOrder);
                // The buddy block starts half_size bytes after `block`.
                Block *buddy = (Block *)((char *)block + halfSize);
```

```
                // Put the buddy in the free list of the smaller order.
                add_to_free_list(allocator, buddy, currentOrder);
        }

        // |block| is now exactly big enough.
        block->order = neededOrder;
        block->next = NULL;

        return (void *)((char *)block + sizeof(Block));
}

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

        Block *block = (Block *)((char *)memory - sizeof(Block));
        size_t order = block->order;

        // Coalesce as far as possible.
        while (order < MAX_ORDER) {
                void *buddyAddr = find_buddy(allocator, block, order);

                // Look for the buddy in the free list of this 'order'.
                bool foundBuddy = false;

                Block *previous = NULL;
                Block *current = allocator->free_lists[order];

                while (current) {
                        if ((void *)current == buddyAddr) {
                                foundBuddy = true;
                                // Remove from free list.
                                if (previous) {
                                        previous->next = current->next;
                                } else {
                                        allocator->free_lists[order] = current->next;
                                }
                                break;
                        }
                        previous = current;
                        current = current->next;
                }

                if (!foundBuddy) {
                        break;  // can't coalesce
                }

                // Merge blocks.
                if (buddyAddr < (void *)block) {
                        block = (Block *)buddyAddr;
                }
```

```
                ++order;
        }

        // Add (possibly merged) block back.
        add_to_free_list(allocator, block, order);
}
```

example/main.c

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

#include "alloc.h"
#include "bootleg_stdio.h"
#include "fallback.h"

static const size_t MEMORY_SIZE =
    1024 * 1024 + 264 /* sizeof(BuddyAllocator) */;

static pfnAllocatorCreate allocatorCreate;
static pfnAllocatorDestroy allocatorDestroy;
static pfnAllocatorAlloc allocatorAlloc;
static pfnAllocatorFree allocatorFree;

static bool load_symbols(void* library) {
        allocatorCreate = (pfnAllocatorCreate)dlsym(library, "allocator_create");
        if (!allocatorCreate) {
                blg_perrorf("can't find `allocator_create`\n");
                return false;
        }

        allocatorDestroy = (pfnAllocatorDestroy)dlsym(library, "allocator_destroy");
        if (!allocatorDestroy) {
                blg_perrorf("can't find `allocator_destroy`\n");
                return false;
        }

        allocatorAlloc = (pfnAllocatorAlloc)dlsym(library, "allocator_alloc");
        if (!allocatorAlloc) {
                blg_perrorf("can't find `allocator_alloc`\n");
                return false;
        }

        allocatorFree = (pfnAllocatorFree)dlsym(library, "allocator_free");
        if (!allocatorFree) {
                blg_perrorf("can't find `allocator_free`\n");
                return false;
        }

        return true;
```

```
}

static size_t rand_size() {
	static const size_t BLOCK_SIZE = 1024;

	return rand() % BLOCK_SIZE + 1;
}

static void test_allocator(Allocator* allocator) {
	static const size_t BLOCK_COUNT = 512;

	void* blocks[BLOCK_COUNT];

	clock_t startAlloc = clock();

	for (size_t i = 0; i != BLOCK_COUNT; ++i) {
		size_t size = rand_size();

		blocks[i] = allocatorAlloc(allocator, size);
		blg_printf("alloc(%zu) = %p\n", size, blocks[i]);

		if (!blocks[i]) {
			blg_perrorf("can't allocate block\n");
		}
	}

	clock_t endAlloc = clock();
	clock_t startFree = clock();

	// Free half of the blocks randomly.
	for (size_t i = 0; i != BLOCK_COUNT / 2; ++i) {
		size_t idx = rand() % BLOCK_COUNT;

		allocatorFree(allocator, blocks[idx]);
		blg_printf("free(%p)\n", blocks[idx]);

		blocks[idx] = NULL;
	}

	clock_t endFree = clock();

	// Try to allocate on the freed blocks.
	for (size_t i = 0; i != BLOCK_COUNT; ++i) {
		if (blocks[i]) continue;

		size_t size = rand_size();

		blocks[i] = allocatorAlloc(allocator, size);
		blg_printf("alloc(%zu) = %p\n", size, blocks[i]);
```

```
                if (!blocks[i]) {
                        blg_perrorf("can't allocate block\n");
                }
        }

        blg_printf("alloc took %fs\n",
                   (double)(endAlloc - startAlloc) / CLOCKS_PER_SEC);
        blg_printf("free took %fs\n",
                   (double)(endFree - startFree) / CLOCKS_PER_SEC);

        for (size_t i = 0; i != BLOCK_COUNT; ++i) {
                allocatorFree(allocator, blocks[i]);
        }
}

static void stress_allocator(Allocator* allocator) {
        static const size_t ITERATIONS = 200000;
        static const size_t BLOCK_COUNT = 20000;

        void* blocks[BLOCK_COUNT];

        for (size_t i = 0; i != BLOCK_COUNT; ++i) {
                blocks[i] = NULL;
        }

        clock_t start = clock();

        // Keep track of how many blocks are in use and randomly decide,
        // whether to allocate or free.
        size_t used = 0;

        for (size_t op = 0; op < ITERATIONS; op++) {
                bool alloc = true;

                if (used == 0) {
                        alloc = true;
                } else if (used == BLOCK_COUNT) {
                        alloc = false;
                } else {
                        alloc = rand() % 2;
                }

                if (alloc) {
                        size_t idx;
                        do {
                                idx = rand() % BLOCK_COUNT;
                        } while (blocks[idx] != NULL);

                        size_t size = rand_size();
```

```
                        blocks[idx] = allocatorAlloc(allocator, size);

                        if (blocks[idx]) {
                                ++used;
                        } else {
                                // Allocation failed
                        }
                } else {
                        size_t idx;
                        do {
                                idx = rand() % BLOCK_COUNT;
                        } while (blocks[idx] == NULL);

                        allocatorFree(allocator, blocks[idx]);
                        blocks[idx] = NULL;

                        --used;
                }
        }

        clock_t end = clock();

        double elapsed = (double)(end - start) / CLOCKS_PER_SEC;
        blg_printf("stress test took %fs\n", elapsed);

        for (size_t i = 0; i != BLOCK_COUNT; ++i) {
                allocatorFree(allocator, blocks[i]);
        }
}

static int cleanup(int retVal, void* library, void* memory,
                   Allocator* allocator) {
        if (allocator) allocatorDestroy(allocator);
        if (memory) munmap(memory, MEMORY_SIZE);
        if (library) dlclose(library);

        return retVal;
}

int main(int argc, char* argv[]) {
        void* library = NULL;
        void* memory = NULL;
        Allocator* allocator = NULL;

        library = dlopen(argv[1], RTLD_LOCAL | RTLD_NOW);
        bool libraryLoaded = false;

        // Try to load the allocator library from argv[1].
        if (argc > 1 && library) {
                libraryLoaded = load_symbols(library);
        }
```

```
        // If we can't load one of the symbols, use fallbacks.
        if (!libraryLoaded) {
                blg_printf("can't load allocator library, using fallbacks");

                allocatorCreate = fb_allocator_create;
                allocatorDestroy = fb_allocator_destroy;
                allocatorAlloc = fb_allocator_alloc;
                allocatorFree = fb_allocator_free;
        }

        // Create a 1 MiB memory region.
        memory = mmap(NULL, MEMORY_SIZE, PROT_READ | PROT_WRITE,
                      MAP_PRIVATE | MAP_ANONYMOUS, -1, 0);

        if (memory == MAP_FAILED) {
                blg_perrorf("can't create memory region\n");
                return cleanup(1, library, memory, allocator);
        }

        // Create an allocator.
        allocator = allocatorCreate(memory, MEMORY_SIZE);

        if (!allocator) {
                blg_perrorf("can't create allocator\n");
                return cleanup(2, library, memory, allocator);
        }

        // Test the allocator.
        srand(123456);

        test_allocator(allocator);
        stress_allocator(allocator);

        return cleanup(0, library, memory, allocator);
}
```

example/fallback.c

```
#include "fallback.h"

#include <sys/mman.h>

struct Allocator {
        /** Some data to silence the compiler's empty
         * struct warning. */
        char marker;
};

static Allocator DUMMY_ALLOCATOR = (Allocator){'\0'};
```

```
typedef struct Block {
        size_t size;
} Block;

Allocator* fb_allocator_create(void* memory, size_t size) {
        return &DUMMY_ALLOCATOR;
}

void fb_allocator_destroy(Allocator* allocator) {
        // no-op
}

void* fb_allocator_alloc(Allocator* allocator, size_t size) {
        Block* block =
            (Block*)mmap(NULL, sizeof(Block) + size, PROT_READ | PROT_WRITE,
                         MAP_ANONYMOUS | MAP_PRIVATE, -1, 0);
        if (block == MAP_FAILED) return NULL;

        block->size = size;

        return (char*)block + sizeof(Block);
}

void fb_allocator_free(Allocator* allocator, void* memory) {
        if (!memory) return;

        Block* block = (Block*)((char*)memory - sizeof(Block));

        // Free allocated memory for the block.
        munmap(block, block->size + sizeof(Block));
}
```

Протокол работы программы

Тестирование:

```
skidunion@apollo example % ./lab_4_example ../alloc-1/liblab_4_alloc_1.dylib
alloc(961) = 0x104dd0028
alloc(432) = 0x104dd0401
alloc(165) = 0x104dd05c9
-- snip --
free(0x104dd82e7)
free(0x104df10e4)
free(0x104de2028)
-- snip --
alloc(1024) = 0x104e13bb5
alloc(259) = 0x104dfc33c
alloc(273) = 0x104e0dcd1
-- snip --
```

```
alloc took 0.002933s

free took 0.001186s

stress test took 0.524958s

skidunion@apollo example % ./lab_4_example ../alloc-2/liblab_4_alloc_2.dylib

alloc(961) = 0x100fbc118

alloc(432) = 0x100fbc518

alloc(165) = 0x100fbc718

-- snip --

free(0x100fbcf98)

free(0x100fe8118)

free(0x100fd3318)

-- snip --

alloc(1024) = 0x100ffe918

alloc(259) = 0x100fc2518

alloc(273) = 0x100fc8b18

-- snip --

alloc took 0.002492s

free took 0.001031s

stress test took 0.250028s

skidunion@apollo example % ./lab_4_example

can't load allocator library, using fallbacks

alloc(961) = 0x102888008

alloc(432) = 0x10288c008

alloc(165) = 0x102890008

-- snip --

free(0x102980008)

free(0x102d18008)

free(0x102aac008)

-- snip --

alloc(1024) = 0x102890008

alloc(259) = 0x102898008

alloc(273) = 0x10289c008

-- snip --

alloc took 0.004760s

free took 0.001851s

stress test took 0.476698s
```

Dtrace:

```
skidunion@apollo example % sudo dtruss -a ./lab_4_example ../alloc-1/liblab_4_alloc_1.dylib

Password:

        PID/THRD  RELATIVE  ELAPSD    CPU SYSCALL(args)                 = return
```

```
       1886/0x3c88:       146:        0:       0 fork()                = 0 0

       1886/0x3c88:       153      31      6 munmap(0x102BDC000, 0x8C000)              = 0 0

       1886/0x3c88:       160       7      6 munmap(0x102C68000, 0x8000)               = 0 0

       1886/0x3c88:       170      11     10 munmap(0x102C70000, 0x4000)               = 0 0

       1886/0x3c88:       185      43     14 munmap(0x102C74000, 0x4000)               = 0 0

       1886/0x3c88:       186       1      1 munmap(0x102C78000, 0x50000)              = 0 0

       1886/0x3c88:       202       1      0 crossarch_trap(0x0, 0x0, 0x0)             = -1 Err#45

       1886/0x3c88:       227      24     23 open(".\0", 0x100000, 0x0)                = 3 0

       1886/0x3c88:       229       2      2 fcntl(0x3, 0x32, 0x16D2B3278)             = 0 0

       1886/0x3c88:       231       2      1 close(0x3)                = 0 0

       1886/0x3c88:       237       5      4 fsgetpath(0x16D2B3288, 0x400, 0x16D2B3268)                 =
102 0

       1886/0x3c88:       253      16     15 fsgetpath(0x16D2B3298, 0x400, 0x16D2B3278)                 =
14 0

       1886/0x3c88:       258       1      0 csrctl(0x0, 0x16D2B369C, 0x4)             = -1 Err#1

       1886/0x3c88:       262       4      3 __mac_syscall(0x199531ACF, 0x2, 0x16D2B35E0)               =
0 0

       1886/0x3c88:       263       0      0 csrctl(0x0, 0x16D2B368C, 0x4)             = -1 Err#1

       1886/0x3c88:       267       3      2 __mac_syscall(0x19952E902, 0x5A, 0x16D2B3620)              =
0 0

      1886/0x3c88:       280       7      6 sysctl([unknown, 3, 0, 0, 0, 0] (2), 0x16D2B2BA8, 0x16D2B2BA0,
0x199530553, 0xD)          = 0 0

      1886/0x3c88:      281      1     1 sysctl([CTL_KERN, 140, 0, 0, 0, 0] (2), 0x16D2B2C58, 0x16D2B2C50,
0x0, 0x0)                = 0 0

       1886/0x3c88:       288       5      4 open("/\0", 0x20100000, 0x0)              = 3 0

       1886/0x3c88:          304         16       15  openat(0x3,  "System/Cryptexes/OS\0",  0x100000,  0x0)
= 4 0

       1886/0x3c88:       305       1      0 dup(0x4, 0x0, 0x0)                = 5 0

       1886/0x3c88:       310       5      4 fstatat64(0x4, 0x16D2B2731, 0x16D2B26A0)          = 0 0

       1886/0x3c88:          315          5        4  openat(0x4,  "System/Library/dyld/\0",  0x100000,  0x0)
= 6 0

       1886/0x3c88:       316       1      0 fcntl(0x6, 0x32, 0x16D2B2730)             = 0 0

       1886/0x3c88:       317       0      0 dup(0x6, 0x0, 0x0)                = 7 0

       1886/0x3c88:       317       0      0 dup(0x5, 0x0, 0x0)                = 8 0

       1886/0x3c88:       318       1      0 close(0x3)                = 0 0

       1886/0x3c88:       318       0      0 close(0x5)                = 0 0

       1886/0x3c88:       318       0      0 close(0x4)                = 0 0

       1886/0x3c88:       319       0      0 close(0x6)                = 0 0

       1886/0x3c88:       320       2      1 __mac_syscall(0x199531ACF, 0x2, 0x16D2B3120)               =
0 0

       1886/0x3c88:       322       2      1 shared_region_check_np(0x16D2B2D40, 0x0, 0x0)              =
0 0

       1886/0x3c88:       326       4      3 fsgetpath(0x16D2B32A0, 0x400, 0x16D2B31E8)                 =
82 0

      alloc took 0.000027s

      free took 0.000039s
```

```
      stress test took 0.497871s

       1886/0x3c88:       328       1      0 fcntl(0x8, 0x32, 0x16D2B32A0)             = 0 0

       1886/0x3c88:       329       0      0 close(0x8)                = 0 0

       1886/0x3c88:       329       0      0 close(0x7)                = 0 0

       1886/0x3c88:       336       2      1 getfsstat64(0x0, 0x0, 0x2)                = 10 0

       1886/0x3c88:       343       8      7 getfsstat64(0x102F66110, 0x54B0, 0x2)             = 10 0

       1886/0x3c88:       348       5      4 getattrlist("/\0", 0x16D2B31D0, 0x16D2B3140)              =
0 0

       1886/0x3c88:                              358                              7                          7
stat64("/System/Volumes/Preboot/Cryptexes/OS/System/Library/dyld/dyld_shared_cache_arm64e\0",
0x16D2B3530, 0x0)   = 0 0

      dtrace: error on enabled probe ID 1690 (ID 845: syscall::stat64:return): invalid address (0x0) in
action #12 at DIF offset 12

       1886/0x3c88:      399     28     2 stat64("/Users/skidunion/Work/Depot/Personal/mai-os-3-sem/cmake-
build-release/src/lab-4/example/lab_4_example\0", 0x16D2B29E0, 0x0)                = 0 0

       1886/0x3c88:       416      16     15 open("/Users/skidunion/Work/Depot/Personal/mai-os-3-sem/cmake-
build-release/src/lab-4/example/lab_4_example\0", 0x0, 0x0)          = 3 0

       1886/0x3c88:       424       9      7 mmap(0x0, 0x95A0, 0x1, 0x40002, 0x3, 0x0)                 =
0x102FA8000 0

       1886/0x3c88:       426       2      1 fcntl(0x3, 0x32, 0x16D2B2AF8)             = 0 0

       1886/0x3c88:       426       1      0 close(0x3)                = 0 0

       1886/0x3c88:       429       2      1 munmap(0x102FA8000, 0x95A0)               = 0 0

       1886/0x3c88:      431      2     1 stat64("/Users/skidunion/Work/Depot/Personal/mai-os-3-sem/cmake-
build-release/src/lab-4/example/lab_4_example\0", 0x16D2B2F50, 0x0)                = 0 0

       1886/0x3c88:          458          4        3 stat64("/usr/lib/libc++.1.dylib\0", 0x16D2B1E90, 0x0)
= -1 Err#2

       1886/0x3c88:                              464                              2                          2
stat64("/System/Volumes/Preboot/Cryptexes/OS/usr/lib/libc++.1.dylib\0",          0x16D2B1E40,          0x0)
= -1 Err#2

       1886/0x3c88:      508      2     2 stat64("/usr/lib/system/libdispatch.dylib\0", 0x16D2AFAB0, 0x0)
= -1 Err#2

       1886/0x3c88:                              510                              2                          1
stat64("/System/Volumes/Preboot/Cryptexes/OS/usr/lib/system/libdispatch.dylib\0",     0x16D2AFA60,     0x0)
= -1 Err#2

       1886/0x3c88:      511      1     0 stat64("/usr/lib/system/libdispatch.dylib\0", 0x16D2AFAB0, 0x0)
= -1 Err#2

       1886/0x3c88:         520         1       1 stat64("/usr/lib/libSystem.B.dylib\0", 0x16D2B1E90, 0x0)
= -1 Err#2

       1886/0x3c88:                              523                              1                          0
stat64("/System/Volumes/Preboot/Cryptexes/OS/usr/lib/libSystem.B.dylib\0",         0x16D2B1E40,         0x0)
= -1 Err#2

       1886/0x3c88:       746      41     38 open("/dev/dtracehelper\0", 0x2, 0x0)             = 3 0

       1886/0x3c88:      1062     316    315 ioctl(0x3, 0x80086804, 0x16D2B1A88)              = 0 0

       1886/0x3c88:      1079      72      7 close(0x3)                = 0 0

       1886/0x3c88:     1251      51     22 open("/Users/skidunion/Work/Depot/Personal/mai-os-3-sem/cmake-
build-release/src/lab-4/example/lab_4_example\0", 0x0, 0x0)          = 3 0

       1886/0x3c88:      1256       5      3 __mac_syscall(0x199531ACF, 0x2, 0x16D2B1190)              =
0 0

       1886/0x3c88:         1273         16        14  map_with_linking_np(0x16D2B0F60,  0x1,  0x16D2B0F90)
= 0 0

       1886/0x3c88:      1275       2      1 close(0x3)                = 0 0

       1886/0x3c88:      1279       4      3 mprotect(0x102B50000, 0x4000, 0x1)                = 0 0
```

```
      1886/0x3c88:        1546        30        0 shared_region_check_np(0xFFFFFFFFFFFFFFFF, 0x0, 0x0)
= 0 0

      1886/0x3c88:      1550       3      2 mprotect(0x102F64000, 0x40000, 0x1)              = 0 0

      1886/0x3c88:        1720         3        2 access("/AppleInternal/XBS/.isChrooted\0", 0x0, 0x0)
= -1 Err#2

      1886/0x3c88:        1932         4        1 bsdthread_register(0x199834D2C, 0x199834D20, 0x4000)
= 1073746399 0

      1886/0x3c88:      2118       3      1 getpid(0x0, 0x0, 0x0)            = 1886 0

      1886/0x3c88:      2124       5      4 shm_open(0x1996CFF51, 0x0, 0x65747379)           = 3 0

      1886/0x3c88:      2125       2      1 fstat64(0x3, 0x16D2B1E20, 0x0)           = 0 0

      1886/0x3c88:       2129       5       3 mmap(0x0, 0x4000, 0x1, 0x40001, 0x3, 0x0)                =
0x102FB0000 0

      1886/0x3c88:      2132       1      0 close(0x3)               = 0 0

      1886/0x3c88:      2146       1      0 ioctl(0x2, 0x4004667A, 0x16D2B1ECC)              = 0 0

      1886/0x3c88:      2153       1      0 mprotect(0x102FBC000, 0x4000, 0x0)               = 0 0

      1886/0x3c88:      2154       1      0 mprotect(0x102FC8000, 0x4000, 0x0)               = 0 0

      1886/0x3c88:      2158       1      0 mprotect(0x102FCC000, 0x4000, 0x0)               = 0 0

      1886/0x3c88:      2159       1      0 mprotect(0x102FD8000, 0x4000, 0x0)               = 0 0

      1886/0x3c88:      2163       1      0 mprotect(0x102FDC000, 0x4000, 0x0)               = 0 0

      1886/0x3c88:      2163       1      0 mprotect(0x102FE8000, 0x4000, 0x0)               = 0 0

      1886/0x3c88:      2169       2      1 mprotect(0x102FB4000, 0xA0, 0x1)          = 0 0

      1886/0x3c88:      2170       1      0 mprotect(0x102FB4000, 0xA0, 0x3)          = 0 0

      1886/0x3c88:      2174       1      1 mprotect(0x102FB4000, 0xA0, 0x1)          = 0 0

      1886/0x3c88:      2177       1      0 mprotect(0x102FEC000, 0x4000, 0x1)               = 0 0

      1886/0x3c88:      2182       2      1 mprotect(0x102FF0000, 0xA0, 0x1)          = 0 0

      1886/0x3c88:      2215       3      1 mprotect(0x102FF0000, 0xA0, 0x3)          = 0 0

      1886/0x3c88:      2219       1      0 mprotect(0x102FF0000, 0xA0, 0x1)          = 0 0

      1886/0x3c88:      2220       1      1 mprotect(0x102FB4000, 0xA0, 0x3)          = 0 0

      1886/0x3c88:      2223       1      0 mprotect(0x102FB4000, 0xA0, 0x1)          = 0 0

      1886/0x3c88:      2224       1      1 mprotect(0x102FEC000, 0x4000, 0x3)               = 0 0

      1886/0x3c88:      2226       1      0 mprotect(0x102FEC000, 0x4000, 0x1)               = 0 0

      1886/0x3c88:      2228       2      1 mprotect(0x102F64000, 0x40000, 0x3)              = 0 0

      1886/0x3c88:      2232       1      0 mprotect(0x102F64000, 0x40000, 0x1)              = 0 0

      1886/0x3c88:       2243        1       0 objc_bp_assist_cfg_np(0x199461000, 0x80000018001C1048, 0x0)
= -1 Err#5

      1886/0x3c88:      2432       1      0 issetugid(0x0, 0x0, 0x0)          = 0 0

      1886/0x3c88:      2463       2      1 mprotect(0x102F64000, 0x40000, 0x3)              = 0 0

      1886/0x3c88:      2508       6      4 getentropy(0x16D2B1538, 0x20, 0x0)               = 0 0

      1886/0x3c88:      2543       1      0 mprotect(0x102F64000, 0x40000, 0x1)              = 0 0

      1886/0x3c88:      2568       1      0 mprotect(0x102F64000, 0x40000, 0x3)              = 0 0

      1886/0x3c88:      2573       1      0 mprotect(0x102F64000, 0x40000, 0x1)              = 0 0

      1886/0x3c88:      2590      14     13 getattrlist("/Users/skidunion/Work/Depot/Personal/mai-os-3-
sem/cmake-build-release/src/lab-4/example/lab_4_example\0", 0x16D2B1DB0, 0x16D2B1DC8)          = 0 0
```

```
      1886/0x3c88:     2604     12    12 access("/Users/skidunion/Work/Depot/Personal/mai-os-3-sem/cmake-build-release/src/lab-4/example\0", 0x4, 0x0)     = 0 0

      1886/0x3c88:     2613     10     9 open("/Users/skidunion/Work/Depot/Personal/mai-os-3-sem/cmake-build-release/src/lab-4/example\0", 0x0, 0x0)       = 3 0

      1886/0x3c88:     2615      2     1 fstat64(0x3, 0x122E04450, 0x0)            = 0 0

      1886/0x3c88:     2616      1     0 csrctl(0x0, 0x16D2B1FDC, 0x4)             = 0 0

      1886/0x3c88:     2619      2     1 fcntl(0x3, 0x32, 0x16D2B1C98)             = 0 0

      1886/0x3c88:     2620      1     1 close(0x3)                = 0 0

      1886/0x3c88:     2631      9     9 open("/Users/skidunion/Work/Depot/Personal/mai-os-3-sem/cmake-build-release/src/lab-4/example/Info.plist\0", 0x0, 0x0)             = -1 Err#2

      1886/0x3c88:     2646      2     1 proc_info(0x2, 0x75E, 0xD)                = 64 0

      1886/0x3c88:     2661      3     2 csops_audittoken(0x75E, 0x10, 0x16D2B2020)                = 0 0

      1886/0x3c88:     2677     10     8 sysctl([unknown, 3, 0, 0, 0, 0] (2), 0x16D2B2378, 0x16D2B2370, 0x19CC66D3D, 0x15)                 = 0 0

      1886/0x3c88:     2678      2     1 sysctl([CTL_KERN, 138, 0, 0, 0, 0] (2), 0x16D2B2408, 0x16D2B2400, 0x0, 0x0)               = 0 0

      1886/0x3c88:     2680      1     0 csops(0x75E, 0x0, 0x16D2B24AC)            = 0 0

      1886/0x3c88:     2685      3     2 mprotect(0x102F64000, 0x40000, 0x3)               = 0 0

      1886/0x3c88:     2835     22    21 open("../alloc-1/liblab_4_alloc_1.dylib\0", 0x0, 0x0)     = 3 0

      1886/0x3c88:     2837      2     1 fcntl(0x3, 0x32, 0x16D28B4F8)             = 0 0

      1886/0x3c88:     2839      2     1 close(0x3)                = 0 0

      1886/0x3c88:     2845      4     3 stat64("../alloc-1/liblab_4_alloc_1.dylib\0", 0x16D28B050, 0x0)     = 0 0

      1886/0x3c88:     2847      1     1 stat64("../alloc-1/liblab_4_alloc_1.dylib\0", 0x16D28AA80, 0x0)     = 0 0

      1886/0x3c88:     2854      7     6 open("../alloc-1/liblab_4_alloc_1.dylib\0", 0x0, 0x0)     = 3 0

      1886/0x3c88:     2862      9     7 mmap(0x0, 0x82F8, 0x1, 0x40002, 0x3, 0x0)                 = 0x102B60000 0

      1886/0x3c88:     2863      1     0 fcntl(0x3, 0x32, 0x16D28AB98)             = 0 0

      1886/0x3c88:     2864      0     0 close(0x3)                = 0 0

      1886/0x3c88:     2891      9     8 open("/Users/skidunion/Work/Depot/Personal/mai-os-3-sem/cmake-build-release/src/lab-4/alloc-1/liblab_4_alloc_1.dylib\0", 0x0, 0x0)                = 3 0

      1886/0x3c88:     3003    235   111 fcntl(0x3, 0x61, 0x16D28A848)             = 0 0

      1886/0x3c88:     3039     37    34 fcntl(0x3, 0x62, 0x16D28A848)             = 0 0

      1886/0x3c88:     3105     63    58 mmap(0x102B6C000, 0x4000, 0x5, 0x40012, 0x3, 0x0)     = 0x102B6C000 0

      1886/0x3c88:     3137      8     5 mmap(0x102B70000, 0x4000, 0x3, 0x40012, 0x3, 0x4000)     = 0x102B70000 0

      1886/0x3c88:     3141      5     3 mmap(0x102B74000, 0x4000, 0x1, 0x40012, 0x3, 0x8000)     = 0x102B74000 0

      1886/0x3c88:     3145      4     3 close(0x3)                = 0 0

      1886/0x3c88:     3163      6     3 munmap(0x102B60000, 0x82F8)               = 0 0

      1886/0x3c88:     3261     12    10 open("/Users/skidunion/Work/Depot/Personal/mai-os-3-sem/cmake-build-release/src/lab-4/alloc-1/liblab_4_alloc_1.dylib\0", 0x0, 0x0)                = 3 0

      1886/0x3c88:     3318      4     2 close(0x3)                = 0 0

      1886/0x3c88:     3324      5     3 mprotect(0x102B70000, 0x4000, 0x1)                = 0 0
```

```
      1886/0x3c88:       3352        3       1 mmap(0x0, 0x100108, 0x3, 0x41002, 0xFFFFFFFFFFFFFFFF, 0x0)
= 0x102B78000 0

      1886/0x3c88:       3355        2       1 getrusage(0x0, 0x16D28C520, 0x0)          = 0 0

      1886/0x3c88:       3377        1       0 getrusage(0x0, 0x16D28C520, 0x0)          = 0 0

      1886/0x3c88:       3377        0       0 getrusage(0x0, 0x16D28C520, 0x0)          = 0 0

      1886/0x3c88:       3414        1       0 getrusage(0x0, 0x16D28C520, 0x0)          = 0 0

      1886/0x3c88:       3500        7       6 write(0x1, "alloc took 0.000027s\n\0", 0x15)              =
21 0

      1886/0x3c88:       3501        1       0 write(0x1, "free took 0.000039s\n\0", 0x14)               =
20 0

      1886/0x3c88:       3792        0       0 getrusage(0x0, 0x16D28C520, 0x0)          = 0 0

      1886/0x3c88:     501662        4       1 getrusage(0x0, 0x16D28C520, 0x0)          = 0 0

      1886/0x3c88:     501676       12      11 write(0x1, "stress test took 0.497871s\n\0", 0x1B)
= 27 0

      1886/0x3c88:     503005       16      15 munmap(0x102B78000, 0x100108)            = 0 0

      1886/0x3c88:     503075        9       9 munmap(0x102B6C000, 0xC000)              = 0 0


      CALL                                        COUNT
      bsdthread_register                              1
      crossarch_trap                                  1
      csops                                           1
      csops_audittoken                                1
      exit                                            1
      fstatat64                                       1
      getentropy                                      1
      getpid                                          1
      issetugid                                       1
      map_with_linking_np                             1
      objc_bp_assist_cfg_np                           1
      proc_info                                       1
      shm_open                                        1
      access                                          2
      fstat64                                         2
      getattrlist                                     2
      getfsstat64                                     2
      ioctl                                           2
      openat                                          2
      shared_region_check_np                          2
      csrctl                                          3
      dup                                             3
      fsgetpath                                       3
      write                                           3
      __mac_syscall                                   4
```

```
sysctl                                  4
getrusage                               6
mmap                                    7
fcntl                                   9
munmap                                  9
open                                   11
stat64                                 13
close                                  16
mprotect                               27
```

Вывод

В ходе выполнения этой лабораторной работы я познакомился с алгоритмами выделения памяти: список свободных элементов и алгоритм двойников, научился работать с динамическиими библиотеками в среде POSIX. Основные трудности возникли при реализации алгоритма двойников: неясно, как хранить блоки. Сам код часто не делал то, чего от него хотелось, потребовалось много времени на отладку. В процессе тестирования первый бенчмарк не показал особой разницы между алгоритмами, пришлось дорабатывать.