Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

**Курсовой проект по курсу**

**«Операционные системы»**

Группа: М8О-215Б-23

Студент: Самарский Я.В.

Преподаватель: Миронов Е.С.

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

Дата: 26.12.24

Москва, 2024

**Постановка задачи**

**Вариант 13.**

Необходимо сравнить два алгоритма аллокации: списки свободных блоков (первое подходящее) и блоки размером 2^n:

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

Каждый аллокатор памяти должен иметь функции аналогичные стандартным функциям free и malloc (realloc, опционально).  Перед работой каждый аллокатор инициализируется свободными страницами памяти, выделенными стандартными средствами ядра. 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.

**Общий метод и алгоритм решения**

Использованные системные вызовы:

* void \*mmap(void \*addr, size\_t length, int prot, int flags, int fd, off\_t offset);– выделяет страницу памяти.
* int munmap(void \*addr, size\_t length); освобождения памяти, ранее отображенной с помощью mmap

Цель данной лабораторной работы сравнить две разные стратегии выделения памяти на примере написания двух аллокаторов: аллокатора, использующего список свободных блоков, и аллокатора, использующего метод двойников. Рассмотрим подробнее каждый из алгоритмов, а потом сравним их на различных тестах.

Разберем аллокатор, использующий списки свободных блоков. Он представляет собой метод управления памятью, который использует список свободных блоков памяти для выделения памяти по запросам.

Основные принципы работы

1. Инициализация и хранение внутренней структуры: при запуске программы мы выделяем системными средствами ядра некоторый объем памяти, который передаем аллокатору. Список блоков будет односвязным, так что в каждом блоке будем хранить: размер текущего блока, ссылку на следующий блок и флаг, указывающий на статус блока(занят/свободен).
2. Запрос памяти(alloc): когда программа запрашивает память определенного размера, аллокатор начинает проходить по списку блоков. Поиск подходящего блока осуществляется последовательно, начиная с первого элемента списка. Если найденный блок точно соответствует запрошенному размеру, он выделяется и помечается, как занятый, при этому пользователю возвращается адрес указателя на блок + размер блока. Если найденный блок больше запрошенного размера, он может быть разделен на два блока: один для удовлетворения запроса, а другой (оставшийся) остается в списке свободных блоков. Этот процесс называется разбиением блока.
3. Освобождение памяти и объединение блоков: когда память больше не нужна, она снова становится свободной. Для уменьшения внутренней фрагментации, аллокатор объединяет смежные свободные блоки. Этот процесс называется объединением блоков. Объединение блоков осуществляется путем проверки соседних блоков в списке свободных блоков. Если соседние блоки также свободны, они объединяются в один больший блок. Это позволяет уменьшить количество мелких блоков и увеличить вероятность нахождения подходящего блока при следующем запросе памяти.

Благодаря такому алгоритму аллокатор эффективно выделяет память и скорость работы зависит не от размера запрашиваемой памяти, а от длины списка

Теперь разберем работу аллокатора блоков длиной 2^n. Скорость выделения памяти можно существенно повысить, если хранить её в словаре длины и блоков. Скорость выделения памяти возрастает потому, что не требуется искать в списке подходящий по размеру блок, а доступ к словарю выполняется за О(1).

Сущность аллокатора блоков длинной 2^n. Размер блока выбирается кратным целой степени двойки. Положим, что размер блока равен 2N. В ответ на запрос о выделении памяти, может быть выделен блок памяти округленным размером в большую сторону до ближайшей степени двойки.

Занятые блоки также хранятся в словаре (адрес, блок), позволяя удалять их также за О(1).

**Код программы**

**NaiveAllocator.cpp**

#pragma once

#include <vector>

#include <cstddef>

#include <map>

#include <unordered\_map>

#include <list>

#include "AllocatorBase.h"

const int alignment = alignof(std::max\_align\_t);

class NaiveAllocator : public AllocatorBase {

public:

    char\* memory;

    char\* top\_memory;

    struct Block {

        void\* ptr;

        size\_t size;

        bool is\_free;

    };

    std::vector<Block> blocks;

    NaiveAllocator(char\* memory, size\_t size) : AllocatorBase(size), memory(memory), top\_memory(memory) {}

    ~NaiveAllocator() override {

        std::free(memory);

    }

    void \* alloc(size\_t block\_size) override {

        block\_size = (block\_size / alignment + 1) \* alignment;

        for (auto& block : blocks) {

            if (block.is\_free && block.size >= block\_size) {

                block.is\_free = false;

                return block.ptr;

            }

        }

        return alloc\_new(block\_size);

    }

    void\* alloc\_new(size\_t block\_size) {

        if (top\_memory + block\_size > memory + size)

            return nullptr; // Not enough memory

        Block block{top\_memory, block\_size, false};

        blocks.push\_back(block);

        top\_memory += block\_size;

        return block.ptr;

    }

    void\* free(void\* block\_ptr) override {

        for (auto& block : blocks) {

            if (block.ptr == block\_ptr) {

                block.is\_free = true;

                break;

            }

        }

        return block\_ptr;

    }

};

class BinaryAllocator : public AllocatorBase {

public:

    char\* memory;

    char\* top\_memory;

    struct Block {

        void\* ptr;

        size\_t size;

    };

    std::unordered\_map<void\*, Block> used;

    std::unordered\_map<size\_t, std::list<Block>> free\_blocks;

    BinaryAllocator(char\* memory, size\_t size) : AllocatorBase(size), memory(memory), top\_memory(memory) {

        for (int i = 1; (1 << i) <= size; i++) {

            free\_blocks[1 << i] = std::list<Block>();

        }

    }

    ~BinaryAllocator() override {

        std::free(memory);

    }

    void \* alloc(size\_t block\_size) override {

        block\_size = (block\_size / alignment + 1) \* alignment;

        for (int i = 1 << bits(block\_size); i <= size; i <<= 1) {

            auto& blocks = free\_blocks[i];

            if (blocks.empty())

                continue;

            auto block = blocks.back();

            used[block.ptr] = block;

            blocks.pop\_back();

            return block.ptr;

        }

        return alloc\_new(block\_size);

    }

    void\* alloc\_new(size\_t block\_size) {

        size\_t binary\_size = 1 << bits(block\_size);

        if (top\_memory + binary\_size > memory + size)

            return nullptr; // Not enough memory

        Block new\_block{top\_memory + binary\_size, binary\_size};

        used[new\_block.ptr] = new\_block;

        top\_memory += binary\_size;

        return new\_block.ptr;

    }

    void\* free(void \*block\_ptr) override {

        if (!used.contains(block\_ptr))

            return nullptr;

        Block b = used[block\_ptr];

        used.erase(block\_ptr);

        free\_blocks[b.size].push\_back(b);

        return block\_ptr;

    }

    int bits(size\_t a) {

        int res = 0;

        while (a > 0) {

            a >>= 1;

            res++;

        }

        return res;

    }

};

**AllocatorBase.h**

#pragma once

#include <cstdlib>

class AllocatorBase {

public:

    size\_t size;

    explicit AllocatorBase(size\_t size) : size(size) {}

    virtual ~AllocatorBase() = default;

    virtual void\* alloc(size\_t block\_size) = 0;

    virtual void\* free(void\* block) = 0;

};

**main.cpp**

#include <iostream>

#include "NaiveAllocator.h"

#include "tests/random\_buffers\_tests.h"

#include "tests/iterative\_buffers\_tests.h"

#include "tests/concrete\_buffers\_tests.h"

void print\_result(Result result, AllocatorBase& allocator) {

    std::cout << result.time\_us << "\t" << result.allocated\_strings << "\t" << result.allocated\_chars << "\t" << (result.allocated\_chars / (float)allocator.size) << std::endl;

}

template<class T>

AllocatorBase\* create(size\_t size) {

    char\* memory = static\_cast<char\*>(malloc(size));

    auto a = new T(memory, size);

    fulfill\_and\_release(\*a);

    return a;

}

void fulfill\_and\_release(AllocatorBase& allocator) {

    const int seed = 1;

    const int max\_size = 100'000;

    std::vector<void\*> blocks;

/\*

    srand(1);

    while (true) {

        auto b = allocator.alloc(rand() % max\_size);

        if (b == nullptr)

            break;

        blocks.push\_back(b);

    }

\*/

    for (int i = 0; i < max\_size; i++) {

        auto buf = allocator.alloc(i);

        if (buf == nullptr)

            break;

        blocks.push\_back(buf);

    }

    for (auto& b : blocks) {

        allocator.free(b);

    }

}

template<class T>

void runTests() {

    std::cout << "Random:" << std::endl;

    AllocatorBase\* allocator = create<T>(100 \* 1024 \* 1024);

    print\_result(random\_allocate\_test(\*allocator, 1, 10'000), \*allocator);

    delete allocator;

    allocator = create<T>(100 \* 1024 \* 1024);

    print\_result(random\_free\_test(\*allocator, 1, 10'000), \*allocator);

    delete allocator;

    std::cout << "\n\nIterative:" << std::endl;

    allocator = create<T>(100 \* 1024 \* 1024);

    print\_result(iterative\_allocate\_test(\*allocator, 100'000), \*allocator);

    delete allocator;

    allocator = create<T>(100 \* 1024 \* 1024);

    print\_result(iterative\_free\_test(\*allocator, 100'000), \*allocator);

    delete allocator;

    std::cout << "\n\nConcrete:" << std::endl;

    allocator = create<T>(100 \* 1024 \* 1024);

    print\_result(concrete\_allocate\_test(\*allocator, 10'000), \*allocator);

    delete allocator;

    allocator = create<T>(100 \* 1024 \* 1024);

    print\_result(concrete\_free\_test(\*allocator, 10'000), \*allocator);

    delete allocator;

}

int main() {

    std::cout << "[Naive]:" << std::endl;

    runTests<NaiveAllocator>();

    std::cout << "\n\n\n[Binary]:" << std::endl;

    runTests<BinaryAllocator>();

    return 0;

}

**Result.h**

#pragma once

#include <cstdint>

struct Result {

    uint64\_t time\_us;

    int allocated\_chars;

    int allocated\_strings;

};

**iterative\_buffers\_tests.h**

#pragma once

#include <cstdlib>

#include <vector>

#include <chrono>

#include "AllocatorBase.h"

#include "result.h"

Result concrete\_allocate\_test(AllocatorBase& allocator, int size) {

    std::vector<void\*> buffers;

    Result result{};

    auto t1 = std::chrono::high\_resolution\_clock::now();

    while (true) {

        auto buf = allocator.alloc(size);

        if (buf == nullptr)

            break;

        result.allocated\_chars += size;

        buffers.push\_back(buf);

    }

    auto t2 = std::chrono::high\_resolution\_clock::now();

    result.time\_us = std::chrono::duration\_cast<std::chrono::microseconds>(t2 - t1).count();

    result.allocated\_strings = buffers.size();

    return result;

}

Result concrete\_free\_test(AllocatorBase& allocator, int size) {

    std::vector<void\*> buffers;

    Result result{};

    while (true) {

        auto buf = allocator.alloc(size);

        if (buf == nullptr)

            break;

        result.allocated\_chars += size;

        buffers.push\_back(buf);

    }

    auto t1 = std::chrono::high\_resolution\_clock::now();

    for (auto& buf : buffers) {

        allocator.free(buf);

    }

    auto t2 = std::chrono::high\_resolution\_clock::now();

    result.time\_us = std::chrono::duration\_cast<std::chrono::microseconds>(t2 - t1).count();

    result.allocated\_strings = buffers.size();

    return result;

}

**iterative\_buffers\_tests.h**

#pragma once

#include <cstdlib>

#include <vector>

#include <chrono>

#include "AllocatorBase.h"

#include "result.h"

Result iterative\_allocate\_test(AllocatorBase& allocator, int max\_size) {

    std::vector<void\*> buffers;

    Result result{};

    auto t1 = std::chrono::high\_resolution\_clock::now();

    for (int i = 0; i < max\_size; i++) {

        auto buf = allocator.alloc(i);

        if (buf == nullptr)

            break;

        result.allocated\_chars += i;

        buffers.push\_back(buf);

    }

    auto t2 = std::chrono::high\_resolution\_clock::now();

    result.time\_us = std::chrono::duration\_cast<std::chrono::microseconds>(t2 - t1).count();

    result.allocated\_strings = buffers.size();

    return result;

}

Result iterative\_free\_test(AllocatorBase& allocator, int max\_size) {

    std::vector<void\*> buffers;

    Result result{};

    for (int i = 0; i < max\_size; i++) {

        auto buf = allocator.alloc(i);

        if (buf == nullptr)

            break;

        result.allocated\_chars += i;

        buffers.push\_back(buf);

    }

    auto t1 = std::chrono::high\_resolution\_clock::now();

    for (auto& buf : buffers) {

        allocator.free(buf);

    }

    auto t2 = std::chrono::high\_resolution\_clock::now();

    result.time\_us = std::chrono::duration\_cast<std::chrono::microseconds>(t2 - t1).count();

    result.allocated\_strings = buffers.size();

    return result;

}

**concrete\_buffers\_tests.h**

#pragma once

#include <cstdlib>

#include <vector>

#include <chrono>

#include "AllocatorBase.h"

#include "result.h"

Result concrete\_allocate\_test(AllocatorBase& allocator, int size) {

    std::vector<void\*> buffers;

    Result result{};

    auto t1 = std::chrono::high\_resolution\_clock::now();

    while (true) {

        auto buf = allocator.alloc(size);

        if (buf == nullptr)

            break;

        result.allocated\_chars += size;

        buffers.push\_back(buf);

    }

    auto t2 = std::chrono::high\_resolution\_clock::now();

    result.time\_us = std::chrono::duration\_cast<std::chrono::microseconds>(t2 - t1).count();

    result.allocated\_strings = buffers.size();

    return result;

}

Result concrete\_free\_test(AllocatorBase& allocator, int size) {

    std::vector<void\*> buffers;

    Result result{};

    while (true) {

        auto buf = allocator.alloc(size);

        if (buf == nullptr)

            break;

        result.allocated\_chars += size;

        buffers.push\_back(buf);

    }

    auto t1 = std::chrono::high\_resolution\_clock::now();

    for (auto& buf : buffers) {

        allocator.free(buf);

    }

    auto t2 = std::chrono::high\_resolution\_clock::now();

    result.time\_us = std::chrono::duration\_cast<std::chrono::microseconds>(t2 - t1).count();

    result.allocated\_strings = buffers.size();

    return result;

}

**Протокол работы программы**

**Тестирование:**

user@DESKTOP-KC5QDB8:~/projects/mai\_os/kp/build$ ./lab

[Naive]:

Random:

796592 20883 104676537 0.998273

69663 20883 104676537 0.998273

Iterative:

381635 14473 104726628 0.998751

33082 14473 104726628 0.998751

Concrete:

200565 10469 104690000 0.998402

17422 10469 104690000 0.998402

[Binary]:

Random:

13505 14082 70374482 0.671143

5044 14082 70374482 0.671143

Iterative:

10748 11845 70146090 0.668965

3980 11845 70146090 0.668965

Concrete:

5126 6400 64000000 0.610352

2120 6400 64000000 0.610352

**Strace:**

execve("./lab", ["./lab"], 0x7ffdb99e1d30 /\* 38 vars \*/) = 0

brk(NULL) = 0x564648fd8000

arch\_prctl(0x3001 /\* ARCH\_??? \*/, 0x7fffdd2a7d00) = -1 EINVAL (Invalid argument)

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f80a09a6000

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

newfstatat(3, "", {st\_mode=S\_IFREG|0644, st\_size=22187, ...}, AT\_EMPTY\_PATH) = 0

mmap(NULL, 22187, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7f80a09a0000

close(3) = 0

openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libstdc++.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\0\0\0\0\0\0\0\0"..., 832) = 832

newfstatat(3, "", {st\_mode=S\_IFREG|0644, st\_size=2260296, ...}, AT\_EMPTY\_PATH) = 0

mmap(NULL, 2275520, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f80a0774000

mprotect(0x7f80a080e000, 1576960, PROT\_NONE) = 0

mmap(0x7f80a080e000, 1118208, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x9a000) = 0x7f80a080e000

mmap(0x7f80a091f000, 454656, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1ab000) = 0x7f80a091f000

mmap(0x7f80a098f000, 57344, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x21a000) = 0x7f80a098f000

mmap(0x7f80a099d000, 10432, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7f80a099d000

close(3) = 0

openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libgcc\_s.so.1", 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|0644, st\_size=125488, ...}, AT\_EMPTY\_PATH) = 0

mmap(NULL, 127720, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f80a0754000

mmap(0x7f80a0757000, 94208, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x3000) = 0x7f80a0757000

mmap(0x7f80a076e000, 16384, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1a000) = 0x7f80a076e000

mmap(0x7f80a0772000, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1d000) = 0x7f80a0772000

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) = 0x7f80a052b000

mprotect(0x7f80a0553000, 2023424, PROT\_NONE) = 0

mmap(0x7f80a0553000, 1658880, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x28000) = 0x7f80a0553000

mmap(0x7f80a06e8000, 360448, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1bd000) = 0x7f80a06e8000

mmap(0x7f80a0741000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x215000) = 0x7f80a0741000

mmap(0x7f80a0747000, 52816, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7f80a0747000

close(3) = 0

openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libm.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\0\0\0\0\0\0\0\0"..., 832) = 832

newfstatat(3, "", {st\_mode=S\_IFREG|0644, st\_size=940560, ...}, AT\_EMPTY\_PATH) = 0

mmap(NULL, 942344, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f80a0444000

mmap(0x7f80a0452000, 507904, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0xe000) = 0x7f80a0452000

mmap(0x7f80a04ce000, 372736, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x8a000) = 0x7f80a04ce000

mmap(0x7f80a0529000, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0xe4000) = 0x7f80a0529000

close(3) = 0

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f80a0442000

arch\_prctl(ARCH\_SET\_FS, 0x7f80a04433c0) = 0

set\_tid\_address(0x7f80a0443690) = 14936

set\_robust\_list(0x7f80a04436a0, 24) = 0

rseq(0x7f80a0443d60, 0x20, 0, 0x53053053) = 0

mprotect(0x7f80a0741000, 16384, PROT\_READ) = 0

mprotect(0x7f80a0529000, 4096, PROT\_READ) = 0

mprotect(0x7f80a0772000, 4096, PROT\_READ) = 0

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f80a0440000

mprotect(0x7f80a098f000, 45056, PROT\_READ) = 0

mprotect(0x564628720000, 4096, PROT\_READ) = 0

mprotect(0x7f80a09e0000, 8192, PROT\_READ) = 0

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

munmap(0x7f80a09a0000, 22187) = 0

getrandom("\x5c\x19\x5e\x46\xd4\xd1\xda\x5d", 8, GRND\_NONBLOCK) = 8

brk(NULL) = 0x564648fd8000

brk(0x564648ff9000) = 0x564648ff9000

futex(0x7f80a099d77c, FUTEX\_WAKE\_PRIVATE, 2147483647) = 0

newfstatat(1, "", {st\_mode=S\_IFCHR|0620, st\_rdev=makedev(0x88, 0x15), ...}, AT\_EMPTY\_PATH) = 0

write(1, "[Naive]:\n", 9) = 9

write(1, "Random:\n", 8) = 8

**mmap(NULL, 104857600, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f809a040000**

brk(0x564649023000) = 0x564649023000

mmap(NULL, 200704, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f809a00f000

brk(0x56464901b000) = 0x56464901b000

mmap(NULL, 397312, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f8099fae000

munmap(0x7f809a00f000, 200704) = 0

brk(0x56464904b000) = 0x56464904b000

mmap(NULL, 790528, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f8099eed000

munmap(0x7f8099fae000, 397312) = 0

brk(0x56464908b000) = 0x56464908b000

write(1, "1343066\t20883\t104676537\t0.998273"..., 33) = 33

**munmap(0x7f809a040000, 104857600) = 0**

munmap(0x7f8099eed000, 790528) = 0

**mmap(NULL, 104857600, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f809a040000**

brk(0x5646490cb000) = 0x5646490cb000

brk(0x56464918b000) = 0x56464918b000

write(1, "1289502\t20883\t104676537\t0.998273"..., 33) = 33

**munmap(0x7f809a040000, 104857600) = 0**

brk(0x56464900c000) = 0x56464900c000

write(1, "\n\n", 2) = 2

write(1, "Iterative:\n", 11) = 11

**mmap(NULL, 104857600, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f809a040000**

brk(0x56464903b000) = 0x56464903b000

brk(0x56464906b000) = 0x56464906b000

brk(0x5646490cb000) = 0x5646490cb000

write(1, "651414\t14473\t104726628\t0.998751\n", 32) = 32

**munmap(0x7f809a040000, 104857600) = 0**

**mmap(NULL, 104857600, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f809a040000**

write(1, "620694\t14473\t104726628\t0.998751\n", 32) = 32

**munmap(0x7f809a040000, 104857600) = 0**

write(1, "\n\n", 2) = 2

write(1, "Concrete:\n", 10) = 10

**mmap(NULL, 104857600, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f809a040000**

write(1, "337971\t10469\t104690000\t0.998402\n", 32) = 32

**munmap(0x7f809a040000, 104857600) = 0**

**mmap(NULL, 104857600, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f809a040000**

write(1, "323337\t10469\t104690000\t0.998402\n", 32) = 32

**munmap(0x7f809a040000, 104857600) = 0**

write(1, "\n\n\n", 3) = 3

write(1, "[Binary]:\n", 10) = 10

write(1, "Random:\n", 8) = 8

**mmap(NULL, 104857600, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f809a040000**

brk(0x5646490ec000) = 0x5646490ec000

write(1, "14224\t14082\t70374482\t0.671143\n", 30) = 30

**munmap(0x7f809a040000, 104857600) = 0**

**mmap(NULL, 104857600, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f809a040000**

write(1, "4705\t14082\t70374482\t0.671143\n", 29) = 29

**munmap(0x7f809a040000, 104857600) = 0**

write(1, "\n\n", 2) = 2

write(1, "Iterative:\n", 11) = 11

**mmap(NULL, 104857600, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f809a040000**

write(1, "11091\t11845\t70146090\t0.668965\n", 30) = 30

**munmap(0x7f809a040000, 104857600) = 0**

**mmap(NULL, 104857600, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f809a040000**

write(1, "3635\t11845\t70146090\t0.668965\n", 29) = 29

**munmap(0x7f809a040000, 104857600) = 0**

write(1, "\n\n", 2) = 2

write(1, "Concrete:\n", 10) = 10

**mmap(NULL, 104857600, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f809a040000**

write(1, "5516\t6400\t64000000\t0.610352\n", 28) = 28

**munmap(0x7f809a040000, 104857600) = 0**

mmap(NULL, 104857600, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f809a040000

write(1, "1902\t6400\t64000000\t0.610352\n", 28) = 28

**munmap(0x7f809a040000, 104857600) = 0**

exit\_group(0) = ?

+++ exited with 0 +++

**Результат сравнения**

Для оценки эффективности аллокаторов были выбраны тесты, которые моделируют реальные сценарии использования памяти. Это включает в себя различные паттерны выделения и освобождения памяти, такие как частое выделение и освобождение небольших блоков, выделение и освобождение больших блоков, а также смешанные сценарии. Такой подход позволяет получить представление о производительности аллокаторов в различных условиях и выявить их сильные и слабые стороны.

* **Фактор использования**

|  |  |  |
| --- | --- | --- |
| Задача | Аллокатор на списке свободных блоков | Аллокатор блоков длиной 2^n |
| Размер страницы памяти, байты | 1e8 | 1e8 |
| Полезная нагрузка (случайная длина блоков), байты | 104676537 | 70374482 |
| Утилизация (случайная длина блоков), байты | 0.998273 | 0.671143 |
| Полезная нагрузка  (линейный рост длины блоков), байты | 104726628 | 70146090 |
| Утилизация (линейный рост длины блоков), байты | 0.998751 | 0.668965 |
| Полезная нагрузка  (блоки длины 1000 байт), байты | 104025000 | 102400000 |
| Утилизация (блоки длины 1000 байт), байты | 0.99206 | 0.976562 |
| Полезная нагрузка  (блоки длины 10000 байт), байты | 104690000 | 64000000 |
| Утилизация (блоки длины 10000 байт), байты | 0.998402 | 0.610352 |

Аллокатор на списке свободных блоков показал себя с лучшей стороны, потому что ему не требуется подгонять длину под степень двойки.

* **Скорость выделения и освобождения блоков**

|  |  |  |  |
| --- | --- | --- | --- |
| Задача | случайная длина блоков | линейный рост длины блоков | блоки длины 10000 байт |
| Выделение (свободные блоки) | 796592 | 381635 | 200565 |
| Освобждение (свободные блоки) | 69663 | 33082 | 17422 |
| Выделение (блоки длиной 2^n) | 13570 | 10730 | 5244 |
| Освобждение (блоки длиной 2^n) | 4907 | 3911 | 2000 |

Скорость выделения и освобождения блоков памяти является также ключевым показателем производительности аллокатора. Эффективный аллокатор должен минимизировать время, необходимое для выделения памяти, чтобы обеспечить быстрое выполнение программ.

Для того, чтобы при измерении времени работы разных аллокаторов учесть особенности каждого из алгоритмов тестирование включает в себя несколько этапов. Мы должны измерить время с разными размерами, и тд. Рассмотрим результаты тестов в таблице выше.

* **Простота использования аллокатора**

Аллокатор с использованием списков блоков является одним из самых простых аллокаторов, но тем не менее является довольно эффективным в некоторых ситуациях, он не требует сложной архитектуры и прост в написании и использовании. В отличии от него аллокатор на блоках длины 2^n немного более сложен. Если использовать аллокаторы в качестве библиотек, то они оба одинаковы легки в использовании, так как имеют функции с одинаковыми сигнатурами.

**Вывод**

В данной лабораторной работе мной были рассмотрены два ключевых метода аллокации памяти: с использованием списков свободных блоков и с использованием блоков длины 2^n. Были написаны программы, реализующие оба алгоритма и проведено всестороннее тестирование для определения слабых и сильных сторон алгоритмов. Каждый из алгоритмов имеет свои особенности, позволяющие побеждать или проигрывать в тестах друг другу, так же не стоит забывать, что каждый из алгоритмов можно продолжать модифицировать и улучшать, для возрастания скорости работы. По результатам тестов лучшим выбором для большинства задач является аллокатор, использующий блоки длиной 2^n. Благодаря своей архитектуре, метод близнецов обеспечивает очень высокую скорость динамического выделения и освобождения памяти, и используется в составе многих современных операционных систем для динамического распределения памяти в ядре системы, драйверах или в других ответственных компонентах системы, критичных к скорости работы.