Міністерство освіти і науки, молоді та спорту України

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Кафедра АСОІУ

ЗВІТ

про виконання лабораторної роботи № 1

з дисципліни

«Сучасні операційні системи»

Варіант 4

|  |  |
| --- | --- |
| Перевірив:  Дифучин А. Ю. | Виконала:  Студентка групи ІС-71  Вознюк Олександра |
|  |  |

Київ 2020

1. **Опис розробленого алгоритму**

* Виділення пам’яті
* Якщо розмір блоку менше або дорівнює загальному розміру вільної пам’яті, то вставити блок ()
* Інакше повернути NULL
* Перевиділення пам’яті
* Якщо адреса NULL, то просто виділити блок пам’яті з даним розміром
* Якщо потрібно збільшити блок і пам’яті достатньо, то створюємо новий блок с новим розміром
* Якщо потрібно зменшити блок, то створюємо новій блок з меншим розміром
* Повертаємо адресу нового зміненого блоку
* Інакше повертаємо NULL
* Вивільнення пам’яті
* По адресі блоку звільнити його і додати цю адресу пам’яті та з цим розміром до вільної пам’яті.

1. **Оцінка часу пошуку вільного блоку пам’яті, оцінка часу вивільнення зайнятого блоку**

Пошук вільного блоку: O(n) в найгіршому випадку де n – кількість блоків

Вивільнення зайнятого блоку: O(1)

1. **Опис переваг та недоліків розробленого алокатора**

Перевагою є швидке вивільнення пам’яті за рахунок того, як організоване зберігання блоків.

Недоліком є великий час пошуку вільного блоку так як нам доводиться перебирати покроково усі блоки, поки не знайдемо вільну частину пам’яті.

1. **Лістинг алокатора пам’яті загального призначення**

Файл Allocator.h

#include <iostream>

#include "Memory.h"

using namespace std;

int main() {

Memory m = Memory(64);

cout << "Create 64 bytes of memory \n";

m.mem\_dump();

void\* ref0 = m.mem\_alloc(16);

cout << "create reference 1 on allocating 16 bytes \n";

m.mem\_dump();

void\* ref1 = m.mem\_alloc(2);

cout<<"Create reference 2 on allocating 2 bytes \n";

m.mem\_dump();

m.mem\_alloc(3);

cout << "Allocate 3 bytes \n";

m.mem\_dump();

m.mem\_realloc(ref0, 8);

cout << "Reallocate reference 1 from 16 bytes to 8 \n";

m.mem\_dump();

m.mem\_realloc(ref0, 24);

cout << "Reallocate reference 1 from 8 bytes to 24 \n";

m.mem\_dump();

m.mem\_free(ref1);

cout << "Free reference 2 \n";

m.mem\_dump();

return 0;

}

Файл Memory.h

#ifndef LAB1\_MEMORY\_H

#define LAB1\_MEMORY\_H

#include <vector>

#include <iostream>

using namespace std;

class Memory {

struct Memory\_unit\_info {

int addr;

size\_t size;

Memory\_unit\_info(int addr, size\_t size) : size(size), addr(addr) { }

};

class Memory\_unit {

int memory;

public:

void operator=(const Memory\_unit& unit) {

this->memory = unit.memory;

}

};

// Information about memory

vector<Memory\_unit\_info> info\_free;

vector<Memory\_unit\_info> info\_in\_use;

// Memory

vector<Memory\_unit> memory\_block;

// Returns index of memory start from addr

long what\_number\_am\_i(void\* addr);

// Returns size aligned to 4B

size\_t size\_with\_align\_4B(size\_t size);

public:

Memory(size\_t size);

// Allocates memory. If cannot returns NULL

void\* mem\_alloc(size\_t size);

// Reallocates memory. Firstly tries to extend current block and if unsuccessful to find new

void\* mem\_realloc(void\* addr, size\_t size);

// Make memory free

void mem\_free(void\* addr);

// Prints memory info. '\_' - free byte, '\*' - byte in use

void mem\_dump();

};

#endif //LAB1\_MEMORY\_H

Файл Memory.cpp

#include "Memory.h"

Memory::Memory(size\_t size) {

size = size\_with\_align\_4B(size);

info\_free = vector<Memory\_unit\_info>();

info\_free.push\_back(Memory\_unit\_info(0, size)); // All memory is free

info\_in\_use = vector<Memory\_unit\_info>();//memory in use

memory\_block = vector<Memory\_unit>((unsigned long)size);//memory blocks

}

void\* Memory::mem\_alloc(size\_t size) {

size = size\_with\_align\_4B(size);//count how many memory to take with 4bytes alignment

for (int i = 0; i < info\_free.size(); i++)//all free memory

if (size <= info\_free[i].size) {//if size is lesser or equal then enable free memory

info\_in\_use.push\_back(Memory\_unit\_info(info\_free[i].addr, size));//take free memory with this size

if (info\_free[i].size == size) // If allocating the whole block

info\_free.erase(info\_free.begin() + i);

else { // PArt of block

info\_free[i].addr += size;

info\_free[i].size -= size;

}

return &memory\_block[info\_in\_use[info\_in\_use.size() - 1].addr];//return an adress of memory block

}

return NULL;//if a block of memory wasn't taken successfully

}

void\* Memory::mem\_realloc(void\* addr, size\_t size) {//what block to realloc and how (increase or decrease a block size memory)

size = size\_with\_align\_4B(size);//count size with 4 bytes allignment

if (addr == NULL)

return mem\_alloc(size);

// Trying enhance current block

long idx = what\_number\_am\_i(addr); // Index in memory

size\_t my\_size = 0; // Size of current block

int used\_idx = 0; // Current block index in info vector

for (int i = 0; i < info\_in\_use.size(); i++) // Finding this index

if (info\_in\_use[i].addr == idx) {

my\_size = (int)info\_in\_use[i].size;

used\_idx = i;

break;

}

int need\_size = (int)(size - my\_size); // How many more we need

if (need\_size > 0) { // If we need to extend

for (int i = 0; i < info\_free.size(); i++)

if (info\_free[i].addr == idx + my\_size &&

need\_size <= info\_free[i].size) { // If have on right free block

info\_in\_use[used\_idx].size += need\_size;

if (info\_free[i].size == need\_size)

info\_free.erase(info\_free.begin() + i);

else {

info\_free[i].addr += need\_size;

info\_free[i].size -= need\_size;

}

return addr;

}

}

else { // If we need to reduce

info\_in\_use[used\_idx].size += need\_size;

for (int i = 0; i < info\_free.size(); i++)

if (info\_free[i].addr == idx + info\_in\_use[used\_idx].size - need\_size) {

info\_free[i].addr += need\_size;

info\_free[i].size -= need\_size;

return addr;

}

info\_free.push\_back(Memory\_unit\_info((int)(info\_in\_use[used\_idx].addr + info\_in\_use[used\_idx].size),

(size\_t)-need\_size));

}

// If cannot enhance current block

for (int i = 0; i < info\_free.size(); i++)

if (size <= info\_free[i].size) {

info\_in\_use.push\_back(Memory\_unit\_info(info\_free[i].addr, size));

if (info\_free[i].size == size)

info\_free.erase(info\_free.begin() + i);

else {

info\_free[i].addr += size;

info\_free[i].size -= size;

}

// Moving data to new place

for (int j = 0; j < info\_in\_use[used\_idx].size; j++)

memory\_block[info\_in\_use[used\_idx].addr + j] = memory\_block[info\_free[i].addr + j];

mem\_free(addr);

return &memory\_block[info\_in\_use[info\_in\_use.size() - 1].addr];

}

return NULL;

}

void Memory::mem\_free(void\* addr) {

long idx = what\_number\_am\_i(addr);

for (int i = 0; i < info\_in\_use.size(); i++)

if (idx == info\_in\_use[i].addr) {

info\_free.push\_back(Memory\_unit\_info(info\_in\_use[i].addr, info\_in\_use[i].size));

info\_in\_use.erase(info\_in\_use.begin() + i);

break;

}

}

void Memory::mem\_dump() {

int i = 0;

while (i < memory\_block.size()) {

for (int j = 0; j < info\_free.size(); j++)

if (info\_free[j].addr == i) {

cout << string(info\_free[j].size, '\_');

i += info\_free[j].size;

}

for (int j = 0; j < info\_in\_use.size(); j++)

if (info\_in\_use[j].addr == i) {

cout << string(info\_in\_use[j].size, '\*');

i += info\_in\_use[j].size;

}

}

cout << endl;

}

long Memory::what\_number\_am\_i(void\* addr) {

if (addr == NULL)

return -1;

return ((long)addr - (long)&memory\_block[0]) / sizeof(Memory\_unit);

}

size\_t Memory::size\_with\_align\_4B(size\_t size) {

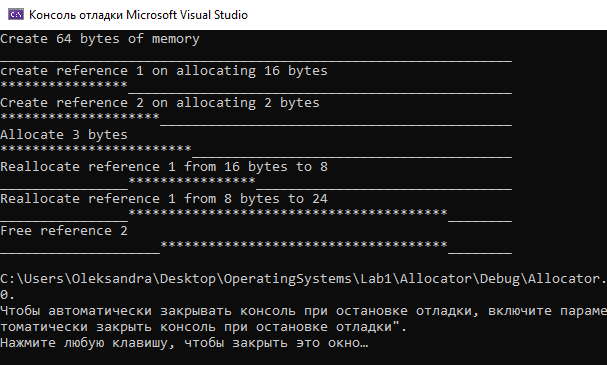
if (size % 4 == 0)

return size;

return size - size % 4 + 4;

}

1. **Приклад роботи алокатора**



<https://github.com/oleksandravozniuk/OperatingSystems>