НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ”

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування та спеціалізованих комп‘ютерних систем

**Лабораторна робота №4  
*З дисципліни: «Операційні системи»***

«**Файлові системи**»

Студент групи КВ-71

Вернік Михайло

Перевірив(ла):

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Київ 2020

**Завдання**

1. Написати програму, що моделює процес управління пам’яттю (розподілу пам’яті для процесів), перетворення віртуальної адреси у фізичну, пошук у пам’яті за запитами процесів, вивільнення пам’яті) при заданому варіантом способі організації пам’яті (перелік варіантів представлений нижче). Вхідні дані – розмір пам’яті, що підлягає розподілу, розміри сторінок (розділів, сегментів тощо), розміри потрібної процесам пам’яті та ін. задаються самостійно та у відповідності до завдання.

2. Продемонструвати роботу моделі з виконанням основних операцій з пам’яттю: надання пам’яті потрібного розміру за запитом процесу, перетворення віртуальної адреси у ―фізичну‖ при зверненні до комірки пам’яті, здійнення запису або читання, вивільнення пам’яті при завершенні процесу. Завдання операцій можна реалізувати за допомогою меню. Якщо потрібно за алгоритмом, додатково продемонструвати процес вивантаження-завантаження сегментів або сторінок. Вихідні дані – наглядна інформація про поточний розподіл пам’яті (карта пам’яті), що містить адреси ділянок пам’яті, стан (вільно чи зайнято та ким) після кожної операції з пам’яттю. Окремо показати коректність перетворення віртуальної адреси звернення до комірки пам’яті за запитом у ―фізичну‖ адресу і здійнення запису до комірки та читання з неї.

Примітка. При моделюванні алгоритмів без використання зовнішньої пам’яті вважати, що вихідні адреси кожного незавантаженого процесу починаються з 0000..00, а розміри кожного із процесів – довільно задані. Кожній адресі незавантаженого процесу за запитом варто поставити у відповідність реальну адресу пам’яті, якщо процес завантажується. Завантаження та вивантаження (при необхідності) процесів виконувати згідно з заданою чергою. При моделюванні алгоритмів з використанням зовнішньої пам'яті слід задати віртуальний адресний простір кожного процесу. Звернення до пам'яті виконувати за віртуальною адресою, здійснювати при необхідності завантаження і вивантаження відповідних частин процесів, показуючи значення конкретної фізичної адреси, що відповідає заданій віртуальній. При моделюванні алгоритмів роботи кеш пам'яті, адресний простір кеш пам'яті і основної пам'яті може задаватися довільно при співвідношенні їх обсягів не менш ніж 1:10. Пошук, запис і заміщення інформації в кеш пам'яті повинно виконуватися шляхом завдання шуканих адрес основної пам'яті

**Варіант1:**

Фіксовані розділи (без використання зовнішньої пам’яті). Кількість розділів меньша, ніж кількість процесів. Процеси утворюють загальну чергу до розділів пам’яті. Використовується лінійний адресний простір. Розміри процесів задаються випадково.

**Текст програми**

**main.cpp**

#include "menu.h"

#include "msoftcon.h"

using namespace std;

void Simulation(){

init\_graphics();

OperationSystem\* os = new OperationSystem();

Menu m("Fixed Memory");

m.SetOperationSystem(os);

m.addItem("Set memory size");

m.addItem("Add segment");

m.addItem("Delete segment");

m.addItem("Add process");

m.addItem("Show all");

m.addItem("Run simulation");

m.addItem("Stop simulation");

m.addItem("Exit");

m.RunThreads();

}

int main()

{

Simulation();

return 0;

}

**menu.cpp**

#include "menu.h"

// ============ MenuItem =====================

MenuItem::MenuItem(string item, color col) {

this->item = item;

this->col = col;

}

// ============ Menu =========================

Menu::Menu(string name, color col) {

this->col = col;

this->name = name;

pos = 0;

items.clear();

}

void Menu::SetOperationSystem(OperationSystem \*operationSystem){

this->operationSystem = operationSystem;

}

void Menu::print(){

// print border

clear\_screen();

set\_cursor\_pos(10,5, std::string(160, '\*'));

for (int i=0;i<13;i++) {

set\_cursor\_pos(10,5+i, "\*");

set\_cursor\_pos(169,5+i, "\*");

}

set\_cursor\_pos(10,18, std::string(160, '\*'));

string s = name;

transform(s.begin(), s.end(), s.begin(), ::toupper);

set\_cursor\_pos(40-name.length()/2-1, 5, " " + s + " ");

}

void Menu::RunThreads(){

std::thread t1(&Menu::show, this);

std::thread t2(&Menu::RunSimulation, this);

t1.join();

t2.join();

}

void Menu::printItems(){

for (int i=0; i<items.size(); i++) {

set\_color(DEFAULT\_COLOR);

set\_cursor\_pos(14,7+i, to\_string(i+1));

set\_color(items.at(i).col);

set\_cursor\_pos(17,7+i, items.at(i).item + "\n" );

}

printCarret();

printScroll();

char c = 0;

do {

c = getch();

if (c != 0 && c!= -32) {

switch(c) {

case 'w': case 'W': pos--; break;

case 's': case 'S': pos++; break;

}

} else {

c = getch();

switch(c) {

case 72: pos--; break;

case 80: pos++; break;

}

}

if (pos < 0) pos = items.size()-1;

if (pos >= items.size()) pos = 0;

printCarret();

printScroll();

} while (c != 13 && c != 27);

}

void Menu::show() {

while(true){

this->print();

//

this->printItems();

pos++;

bool state = this->isRunSimualtion;

switch (pos)

{

case 1:

this->isRunSimualtion = false;

this->print();

this->operationSystem->SetMemory();

//this->isRunSimualtion = true;

clear\_menu();

break;

case 2:

this->isRunSimualtion = false;

this->print();

this->operationSystem->AddSegment();

//this->isRunSimualtion = true;

clear\_menu();

break;

case 3:

this->isRunSimualtion = false;

this->print();

this->operationSystem->DeleteSegment(0);

//this->isRunSimualtion = true;

clear\_menu();

break;

case 4:

this->isRunSimualtion = false;

this->print();

this->operationSystem->AddProcess();

//this->isRunSimualtion = true;

clear\_menu();

break;

case 5:

this->isRunSimualtion = false;

this->print();

this->operationSystem->ShowAll();

cin.get();

//this->isRunSimualtion = true;

clear\_menu();

break;

case 6:

this->isRunSimualtion = true;

clear\_menu();

break;

case 7:

state = false;

this->isRunSimualtion = false;

clear\_menu();

break;

case 8:

default:

this->print();

exit(0);

break;

}

if (state){

this->isRunSimualtion = true;

}

pos--;

}

}

void Menu::RunSimulation(){

while (true){

if (this->isRunSimualtion){

this->operationSystem->TIMER++;

set\_cursor\_pos(80,5," TIME: " + to\_string(this->operationSystem->TIMER) + " \n");

// RUN SYSTEM

this->operationSystem->RunSimulation();

this->operationSystem->ShowAllInRealTime();

std::this\_thread::sleep\_for(std::chrono::milliseconds(500));

}

}

}

void Menu::addItem(string item, color col) {

MenuItem mi(item, col);

items.push\_back(mi);

}

void Menu::clearItems() {

items.clear();

}

int Menu::returnChoise() {

return pos;

}

void Menu::printScroll() {

for (int i=0;i<10; i++) {

set\_cursor\_pos(65,7+i, "");

printf("%c", 176);

}

if (items.size()>10)

set\_cursor\_pos(65,7, "█");

}

void Menu::printCarret() {

for (int i=0;i<10; i++) {

set\_cursor\_pos(12,7+i, " ");

}

set\_color(cWHITE);

set\_cursor\_pos(12, 7+pos, ">");

}

**Menu.h**

#ifndef MENU\_H\_INCLUDED

#define MENU\_H\_INCLUDED

#include <vector>

#include <string>

#include <iostream>

#include <cstdio>

#include <algorithm>

#include <thread>

#include <mutex> // std::mutex

#include <atomic>

#include "OperationSystem.h"

using namespace std;

struct MenuItem {

string item;

color col;

MenuItem(string, color);

};

class Menu {

private:

static const color DEFAULT\_COLOR = cDARK\_GRAY;

static const char CARRET = '>';

string name;

vector<MenuItem> items;

int pos;

color col;

OperationSystem\* operationSystem;

std::thread t1;

bool isRunSimualtion = false;

void printScroll();

void printCarret();

public:

void RunThreads();

void print();

void printItems();

void RunSimulation();

void SetOperationSystem(OperationSystem \*operationSystem);

Menu(string, color = DEFAULT\_COLOR);

void show();

void addItem(string, color = DEFAULT\_COLOR);

void clearItems();

int returnChoise();

};

#endif // MENU\_H\_INCLUDED

**Msoftcon.cpp**

//msoftcon.cpp

//provides routines to access Windows console functions

//compiler needs to be able to find this file

//in MCV++, /Tools/Options/Directories/Include/type path name

#include "msoftcon.h"

#include <conio.h>

#include <mutex>

HANDLE hConsole; //console handle

char fill\_char; //character used for fill

std::mutex mx\_set\_cursor\_pos; // mutex for synchronous printing;

std::mutex mx\_clear\_screen;

std::mutex mx\_save\_positions;

int saved\_x, saved\_y;

//--------------------------------------------------------------

void init\_graphics()

{

COORD console\_size = {80, 25};

//open i/o channel to console screen

hConsole = CreateFile("CONOUT$", GENERIC\_WRITE | GENERIC\_READ,

FILE\_SHARE\_READ | FILE\_SHARE\_WRITE,

0L, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, 0L);

//set to 80x25 screen size

SetConsoleScreenBufferSize(hConsole, console\_size);

//set text to white on black

SetConsoleTextAttribute( hConsole, (WORD)((0 << 4) | 15) );

fill\_char = '\xDB'; //default fill is solid block

clear\_screen();

}

//--------------------------------------------------------------

void set\_color(color foreground, color background)

{

SetConsoleTextAttribute( hConsole,

(WORD)((background << 4) | foreground) );

} //end setcolor()

/\* 0 Black 8 Dark gray

1 Dark blue 9 Blue

2 Dark green 10 Green

3 Dark cyan 11 Cyan

4 Dark red 12 Red

5 Dark magenta 13 Magenta

6 Brown 14 Yellow

7 Light gray 15 White

\*/

void save\_positions(int x, int y){

mx\_save\_positions.lock();

saved\_x = x;

saved\_y = y;

mx\_save\_positions.unlock();

}

//--------------------------------------------------------------

void set\_cursor\_pos(int x, int y, std::string text)

{

mx\_set\_cursor\_pos.lock();

COORD cursor\_pos; //origin in upper left corner

cursor\_pos.X = x - 1; //Windows starts at (0, 0)

cursor\_pos.Y = y - 1; //we start at (1, 1)

SetConsoleCursorPosition(hConsole, cursor\_pos);

std::cout << text;

mx\_set\_cursor\_pos.unlock();

}

//--------------------------------------------------------------

void clear\_menu(){

set\_cursor\_pos(1, 25, "");

for(int j=0; j<25; j++)

putch('\n');

set\_cursor\_pos(1, 1, "");

}

void clear\_real\_time\_table(){

}

void clear\_screen()

{

mx\_clear\_screen.lock();

#if defined \_WIN32

system("cls");

#elif defined (\_\_LINUX\_\_) || defined(\_\_gnu\_linux\_\_) || defined(\_\_linux\_\_)

system("clear");

#elif defined (\_\_APPLE\_\_)

system("clear");

#endif

mx\_clear\_screen.unlock();

}

//--------------------------------------------------------------

void wait(int milliseconds)

{

Sleep(milliseconds);

}

//--------------------------------------------------------------

void clear\_line() //clear to end of line

{ //80 spaces

//.....1234567890123456789012345678901234567890

//.....0........1.........2.........3.........4

\_cputs(" ");

\_cputs(" ");

}

//--------------------------------------------------------------

void draw\_rectangle(int left, int top, int right, int bottom)

{

char temp[80];

int width = right - left + 1;

int j;

for(j=0; j<width; j++) //string of squares

temp[j] = fill\_char;

temp[j] = 0; //null

for(int y=top; y<=bottom; y++) //stack of strings

{

set\_cursor\_pos(left, y, std::string(temp));

//\_cputs(temp);

}

}

//--------------------------------------------------------------

void draw\_circle(int xC, int yC, int radius)

{

double theta, increment, xF, pi=3.14159;

int x, xN, yN;

increment = 0.8 / static\_cast<double>(radius);

for(theta=0; theta<=pi/2; theta+=increment) //quarter circle

{

xF = radius \* cos(theta);

xN = static\_cast<int>(xF \* 2 / 1); //pixels not square

yN = static\_cast<int>(radius \* sin(theta) + 0.5);

x = xC-xN;

while(x <= xC+xN) //fill two horizontal lines

{ //one for each half circle

set\_cursor\_pos(x, yC-yN, std::to\_string(fill\_char));// putch(fill\_char); //top

set\_cursor\_pos(x++, yC+yN, std::to\_string(fill\_char));// putch(fill\_char); //bottom

}

} //end for

}

//--------------------------------------------------------------

void draw\_line(int x1, int y1, int x2, int y2)

{

int w, z, t, w1, w2, z1, z2;

double xDelta=x1-x2, yDelta=y1-y2, slope;

bool isMoreHoriz;

if( fabs(xDelta) > fabs(yDelta) ) //more horizontal

{

isMoreHoriz = true;

slope = yDelta / xDelta;

w1=x1; z1=y1; w2=x2, z2=y2; //w=x, z=y

}

else //more vertical

{

isMoreHoriz = false;

slope = xDelta / yDelta;

w1=y1; z1=x1; w2=y2, z2=x2; //w=y, z=x

}

if(w1 > w2) //if backwards w

{

t=w1; w1=w2; w2=t; // swap (w1,z1)

t=z1; z1=z2; z2=t; // with (w2,z2)

}

for(w=w1; w<=w2; w++)

{

z = static\_cast<int>(z1 + slope \* (w-w1));

if( !(w==80 && z==25) ) //avoid scroll at 80,25

{

if(isMoreHoriz)

set\_cursor\_pos(w, z, "");

else

set\_cursor\_pos(z, w, "");

putch(fill\_char);

}

}

}

//--------------------------------------------------------------

void draw\_pyramid(int x1, int y1, int height)

{

int x, y;

for(y=y1; y<y1+height; y++)

{

int incr = y - y1;

for(x=x1-incr; x<=x1+incr; x++)

{

set\_cursor\_pos(x, y, "");

putch(fill\_char);

}

}

}

//--------------------------------------------------------------

void set\_fill\_style(fstyle fs)

{

switch(fs)

{

case SOLID\_FILL: fill\_char = '\xDB'; break;

case DARK\_FILL: fill\_char = '\xB0'; break;

case MEDIUM\_FILL: fill\_char = '\xB1'; break;

case LIGHT\_FILL: fill\_char = '\xB2'; break;

case X\_FILL: fill\_char = 'X'; break;

case O\_FILL: fill\_char = 'O'; break;

}

}

//--------------------------------------------------------------

**OperationSystem.cpp**

#include "OperationSystem.h"

OperationSystem::OperationSystem(){

}

OperationSystem::~OperationSystem(){

}

std::string OperationSystem::GetProgressBar(float progress){

int barWidth = 20;

std::string progressBar = "";

progressBar += this->item + "[";

int pos = barWidth \* progress;

for (int i = 0; i < barWidth; ++i) {

if (i < pos) progressBar += "=";

else if (i == pos) progressBar += ">";

else progressBar += " ";

}

progressBar +="] " + to\_string((int)(progress \* 100.0)) + " %\n";

//std::cout.flush();

return progressBar;

}

void OperationSystem::RunSimulation(){

// If the Disk speed much bigger than MEMORY SIZE

// then we can say that loading process into memory takes 0 seconds!

for (auto& process: this->processes){

if (process.endTime < this->TIMER && process.state != ProcessState::NOT\_STARTED){

process.state = ProcessState::FINISHED;

process.segment->state = State::FREE;

clear\_real\_time\_table();

}else if (process.startTime > this->TIMER){

process.state = ProcessState::NOT\_STARTED;

}else if (process.startTime == this->TIMER){

for (auto& segment : this->segments){

if (segment.state == State::FREE && segment.size >= process.size){

segment.process = &process;

segment.state = State::OCCUPIED;

process.segment = &segment;

process.state = ProcessState::STARTED;

break;

}

}

}else if (process.startTime > this->TIMER && process.endTime < this->TIMER && process.state == ProcessState::STARTED){

process.state = ProcessState::IN\_PROGRESS;

}

}

}

void OperationSystem::ShowAllInRealTime(){

this->item = "";

int column = 7;

int row = 70;

cout << "\n";

set\_cursor\_pos(row,column++, this->item + "MEMORY SIZE: " + to\_string(this->MEMORY\_SIZE) + "\n");

set\_cursor\_pos(row,column++, this->item + "Segment ID\tStart address\tEnd address\tSize\tProcess Id\tState\n");

for (auto segment : this->segments){

float progress = (segment.process == nullptr || segment.state == State::FREE) ? 0 : (segment.process->size) / (float)segment.size;

set\_cursor\_pos(row,column, string(100, ' '));

set\_cursor\_pos(row,column++, this->item + to\_string(segment.segmentId) + "\t\t" + to\_string(segment.startAddress) + "\t\t" +

to\_string(segment.endAddress) + "\t\t" + to\_string(segment.size) + "\t" + ((segment.process == nullptr || segment.state == State::FREE) ? "-" : to\_string(segment.process->processId)) + "\t" +

((segment.state == State::FREE) ?"Free" : "Occupied") + " " + this->GetProgressBar(progress));

}

column++;

set\_cursor\_pos(row,column++, this->item + "Process queue number: " + to\_string(this->processes.size()) + "\n");

set\_cursor\_pos(row,column++, this->item + "Process ID\tStart time\tEnd time\tStart address\tEnd address\tSize\tState\n");

for (auto process : this->processes){

string state = "";

switch (process.state)

{

case ProcessState::NOT\_STARTED :

state = "NOT\_STARTED";

break;

case ProcessState::STARTED :

state = "STARTED";

break;

case ProcessState::IN\_PROGRESS :

state = "IN\_PROGRESS";

break;

case ProcessState::STOPPED :

state = "STOPPED";

break;

case ProcessState::FINISHED :

state = "FINISHED";

break;

default:

break;

}

std::string s = this->item +

to\_string(process.processId) +

"\t\t" +

to\_string(process.startTime) + "\t\t" + to\_string(process.endTime) + "\t\t" + to\_string(process.startAddress) +

"\t\t" + to\_string(process.endAddress) + "\t\t" + to\_string(process.size) + "\t";

set\_cursor\_pos(s.size() + row,column, string(state.size() + 120, ' '));

set\_cursor\_pos(row,column, s + state + "\n");

column++;

}

this->item = " \t \*\t";

}

void OperationSystem::ShowAll(){

cout << "\n"<< this->item << "MEMORY SIZE: " << this->MEMORY\_SIZE << "\n";

cout << this->item << "Segment ID\tStart address\tEnd address\tSize\tState\n";

for (auto segment : this->segments){

float progress = (segment.process == nullptr || segment.state == State::FREE) ? 0 : (segment.process->size) / (float)segment.size;

cout << this->item << segment.segmentId << "\t\t" << segment.startAddress << "\t\t" << segment.endAddress << "\t\t" << segment.size << "\t" << ((segment.state == State::FREE) ?"Free" : "Occupied") << " " + this->GetProgressBar(progress);

}

cout << "\n";

cout << this->item << "Process queue number: " << this->processes.size() << "\n";

cout << this->item << "Process ID\tStart time\tEnd time\tStart address\tEnd address\tSize\tState\n";

for (auto process : this->processes){

string state = "";

switch (process.state)

{

case ProcessState::NOT\_STARTED :

state = "NOT\_STARTED";

break;

case ProcessState::STARTED :

state = "STARTED";

break;

case ProcessState::IN\_PROGRESS :

state = "IN\_PROGRESS";

break;

case ProcessState::STOPPED :

state = "STOPPED";

break;

case ProcessState::FINISHED :

state = "FINISHED";

break;

default:

break;

}

cout << this->item <<

process.processId <<

"\t\t" <<

process.startTime << "\t\t" << process.endTime << "\t\t" << process.startAddress << "\t\t" << process.endAddress << "\t\t" << process.size << "\t" << state << "\n";

}

}

void OperationSystem::DeleteSegment(int segmentId){

}

void OperationSystem::SetMemory(){

set\_cursor\_pos(47, 6, "\n" + this->item + "Please enter the memory size: ");

std::cin >> this->MEMORY\_SIZE;

}

void OperationSystem::AddSegment(){

int startPos = 0;

int endPos = -1;

do {

if (startPos == endPos){

cout << "\n" << this->item << "The start segment position must be less than end position.\n";

}

cout << "\n"<< this->item <<"Start position: ";

std::cin >> startPos;

cout << "\n"<< this->item <<"End position: ";

std::cin >> endPos;

if (endPos > this->MEMORY\_SIZE){

cout << "\n"<< this->item <<"The end segment position must be less or equal than MEMORY SIZE.\n";

}

bool isCrossed = false;

for (auto segment : this->segments){

if (segment.startAddress < endPos && segment.endAddress > startPos){

isCrossed = true;

cout << "\n"<< this->item <<"There is a cross in memory.\n";

// Show current memory table.

this->ShowAll();

break;

}

}

if (isCrossed){

endPos = startPos - 1;

}

} while (startPos > endPos);

this->segments.push\_back(Segment(this->segments.size() + 1, startPos, endPos));

sort(this->segments.begin(), this->segments.end(),[](Segment& left, Segment& right){

return left.startAddress < right.startAddress;

});

}

void OperationSystem::AddProcess(){

int startPos = 0;

int endPos = 0;

int startTime = 0;

int endTime = 0;

int times = 0;

do {

if (startPos >= endPos){

cout << "\n"<< this->item <<"The start process position must be less than end position.\n";

}else

if (startTime >= endTime){

cout << "\n"<< this->item <<"The start time must be less than end time.\n";

}

cout << "\n"<< this->item <<"Start position: ";

std::cin >> startPos;

cout << "\n"<< this->item <<"End position: ";

std::cin >> endPos;

cout << "\n"<< this->item <<"Start time: ";

std::cin >> startTime;

cout << "\n"<< this->item <<"End time: ";

std::cin >> endTime;

times++;

} while ((startPos > endPos || startTime > endTime) || times > 2);

if (startPos > endPos || startTime > endTime){

// Error message.

// The proccess hasn't been added to the Queue.

}else{

this->processes.push\_back(Process(this->processes.size() + 1, startTime, endTime, startPos, endPos));

}

sort(this->processes.begin(), this->processes.end(),[](Process& left, Process& right){

return left.startTime > right.startTime;

});

}

**OperationSystem.h**

#pragma once

#include <vector>

#include <iostream>

#include "Segment.h"

#include "Process.h"

#include "msoftcon.h"

using namespace std;

#include <string>

#include <stdlib.h> /\* srand, rand \*/

#include <time.h> /\* time \*/

#include <algorithm>

#include <thread>

#include <chrono>

class OperationSystem

{

private:

vector<Segment> segments;

vector<Process> processes;

int MEMORY\_SIZE;

std::string item = " \t \*\t";

public:

OperationSystem();

~OperationSystem();

void RunSimulation();

std::string GetProgressBar(float progress);

void SetMemory();

void AddSegment();

void DeleteSegment(int segmentId);

void AddProcess();

void ShowAll();

void ShowAllInRealTime();

void IncreaseTimer();

std::thread t1;

int TIMER = 0;

};

#pragma once

class Segment;

enum ProcessState{

NOT\_STARTED,

STARTED,

IN\_PROGRESS,

STOPPED,

FINISHED

};

Process.cpp

class Process {

public:

Process(int processId, int startTime, int endTime, int startAddress, int endAddress)

: processId(processId), startTime(startTime), endTime(endTime), startAddress(startAddress), endAddress(endAddress) {

size = endAddress - startAddress;

state = ProcessState::NOT\_STARTED;

segment = nullptr;

};

Process(const Process &obj){

this->state = obj.state;

this->startTime = obj.startTime;

this->startAddress = obj.startAddress;

this->size = obj.size;

this->segment = obj.segment;

this->processId = obj.processId;

this->endTime = obj.endTime;

this->endAddress = obj.endAddress;

}

int processId;

int startTime;

int endTime;

int startAddress;

int endAddress;

int size;

Segment\* segment;

ProcessState state;

};

**Segment.h**

#pragma once

#include "Process.h"

enum State{

FREE,

OCCUPIED

};

class Segment {

public:

Segment(int segmentId, int startAddress, int endAddress) : segmentId(segmentId), startAddress(startAddress), endAddress(endAddress){

state = State::FREE;

size = endAddress - startAddress;

process = nullptr;

}

Segment(const Segment &obj){

this->segmentId = obj.segmentId;

this->startAddress = obj.startAddress;

this->size = obj.size;

this->endAddress = obj.endAddress;

this->state = obj.state;

this->process = obj.process;

}

int segmentId;

int startAddress;

int endAddress;

int size;

Process\* process;

State state;

};

**Тест**

