Министерство образования и науки Российской Федерации

|  |  |
| --- | --- |
|  | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования | |
| «Новосибирский государственный технический университет» | |
| Факультет автоматики и вычислительной техники  Кафедра АСУ | |

ОТЧЁТ

по лабораторной работе №1

«Моделирование виртуальной памяти»

по дисциплине: Технология программирования

Проверил:

к.т.н., доцент

Астапчук В.А.

Выполнили:

Студенты гр. АВТ-612, АВТФ

Савинцев А.С,

Ларионова Д.Н,

Новосибирск

2018

1. Задание

Разработать программу моделирования виртуальной памяти для хранения блоков целых чисел (по 512 байт). Реализовать функции создания файла, чтения элемента из файла, записи элемента в файл и вывода содержимого файла.

2 Код программы

#include <fstream>

#include <iostream>

#include <string>

#include <iomanip>

#include <sstream>

#include <windows.h>

using namespace std;

#define MAX RAND\_MAX/2 // Максимальное значение принимаемое элементами по модулю

#define pageSize 512 // Размер одной страницы

#define pageCountNum pageSize / sizeof(int) // Количество элементов типа int на одной странице

#define commandsCount 6 // Количество команд консоли

#define RAMCounter 2 // Количество блоков памяти

#define indent 6 // Отступ в сообщениях

#define nullString ""

#define helpString "/?"

#define indentString cout << endl;

#define negativeValue -1

#define dotChar '.'

#define minusChar '-'

#define showMessage\_podskazka cout << "> ";

#define showMessage\_helpCreate cout << "Описание:" << endl << "Создает файл." << endl << endl << "Использование:" << endl << std::setw(indent) << "" << "create " << "<имя> <размер>" << endl << endl << "Параметры" << endl << std::setw(indent) << "" << std::setw(indent\*3) << std::left << "<имя>" << "Имя создаваемого файла." << endl << std::setw(indent) << "" << std::setw(indent\*3) << "<размер>" << "Размер создаваемого файла в КБайтах." << endl << endl;

#define showMessage\_helpRead cout << "Описание:" << endl << "Считывает элемент из файла если он существует." << endl << endl << "Использование:" << endl << std::setw(indent) << "" << "read " << "<имя> <элемент>" << endl << endl << "Параметры" << endl << std::setw(indent) << "" << std::setw(indent\*3) << std::left << "<имя>" << "Имя читаемого файла." << endl << std::setw(indent) << "" << std::setw(indent\*3) << "<элемент>" << "Номер элемента для чтения." << endl << endl;

#define showMessage\_helpWrite cout << "Описание:" << endl << "Записывает элемент в файл." << endl << endl << "Использование:" << endl << std::setw(indent) << "" << "write " << "<имя> <элемент> <значение>" << endl << endl << "Параметры" << endl << std::setw(6) << "" << std::setw(indent\*3) << std::left << "<имя>" << "Имя файла для записи элемента." << endl << std::setw(indent) << "" << std::setw(indent\*3) << "<элемент>" << "Номер элемента для записи." << endl << std::setw(indent) << "" << std::setw(indent\*3) << "<значение>" << "Новое значение элемента."<< endl << endl;

#define showMessage\_helpOutput cout << "Описание:" << endl << "Вывод файла на экран." << endl << endl << "Использование:" << endl << std::setw(indent) << "" << "output " << "<имя>" << endl << endl << "Параметры" << endl << std::setw(indent) << "" << std::setw(indent\*3) << std::left << "<имя>" << "Имя файла для вывода его содержимого." << endl << endl;

#define showMessage\_helpHelp cout << "Описание:" << endl << "Вывод справочной информации о командах." << endl << endl << "Использование:" << endl << std::setw(indent) << "" << "help " << endl << endl;

#define showMessage\_helpExit cout << "Описание:" << endl << "Завершает работу программы." << endl << endl << "Использование:" << endl << std::setw(indent) << "" << "exit " << endl << endl;

#define showMessage\_helpHelpParameter cout << "Данная команда не поддержиается. Воспользуйтесь параметром " << '"' << str << " /?" << '"' << '.' << endl << endl;

#define showMessage\_helpHelpCommand cout << "Для получения сведений об определенной команде наберите HELP <имя команды> " << endl;

#define showMessage\_Create cout << std::setw(indent\*2) << std::left << "CREATE" << "Создание файла." << endl ;

#define showMessage\_Read cout << std::setw(indent\*2) << std::left << "READ" << "Чтение элемента из файла." << endl ;

#define showMessage\_Write cout << std::setw(indent\*2) << std::left << "WRITE" << "Запись элемента в файл." << endl ;

#define showMessage\_Output cout << std::setw(indent\*2) << std::left << "OUTPUT" << "Вывод файла на экран." << endl ;

#define showMessage\_Help cout << std::setw(indent\*2) << std::left << "HELP" << "Вывод справочной информации о командах." << endl ;

#define showMessage\_Exit cout << std::setw(indent\*2) << std::left << "EXIT" << "Завершение работы программы." << endl ;

#define showMessage\_errorParam cout << "Ошибка в синтаксисе команды." << endl << endl;

#define showMessage\_errorCommand cout << '"' << command << '"' << " не является внутренней или внешней" << endl << "командой, исполняемой программой или пакетным фалом." << endl << "Для просмотра всех доступных команд используйте " << '"' << "help" << '"' << '.' << endl << endl;

#define showMessage\_errorCreateFile cout << "Произошла ошибка при создании файла." << endl << endl;

#define showMessage\_errorOpenFile cout << "Произошла ошибка при открытии файла. Файл не найден." << endl << endl;

#define showMessage\_errorReadElement cout << "Произошла ошибка при чтении файла. Элемент не найден." << endl << endl;

#define showMessage\_errorWriteValue cout << "Произошла ошибка при записи в файл. Введеное значение находится вне диапазона допустимых." << endl << endl;

#define showMessage\_errorWriteRAM cout << "Произошла ошибка при записи в память." << endl << endl;

enum commandsEnum { Create, Read, Write, Output, Help, Exit }; // Команды

string commandString[commandsCount] = { "create", "read", "write", "output", "help", "exit" };

string delimiterString = " ,:;"; // Разделители в строке

string lastUseFile = nullString; // Имя последнего используемого файла

int lastUseRAM = -1; // Номер последней использованной ячейки памяти

int readValue; // Считаное значение

struct structRAM

{

int pageNumber = -1;

int pageArray[pageSize / sizeof(int)];

} RAM[RAMCounter]; // Оперативная память

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// \_\_\_\_\_\_\_\_\_\_\_\_\_ Функции проверки \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// проверка является ли "num" числом

int isNum(string num)

{

bool dotInNum = false;

bool minusInNum = false;

for (int i = 0; i < num.length(); i++)

{

if (!isdigit(num[i]))

{

if (num[i] == dotChar)

{

if (!dotInNum) dotInNum = true;

else return negativeValue;

}

else if (num[i] == minusChar && !i) minusInNum = true;

else return negativeValue;

}

}

return 1;

}

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// \_\_\_\_\_\_\_\_\_\_ Функции обработки строк \_\_\_\_\_\_\_\_\_\_\_

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// перевод строки "userString" в строчную

string toLower(string userString)

{

string tempString;

char\* tempChar = new char[userString.length() + 1];

for (int i = 0; i < userString.length(); i++)

tempChar[i] = (char)tolower(userString[i]);

tempChar[userString.length()] = '\0';

tempString = tempChar;

delete[] tempChar;

return tempString;

}

// получить из строки "fullString" слово на "posWord" позиции при использовании разделителей записанных в строке "delimiterString"

string getWord(string fullString, int posWord, string delimiterString)

{

int wordCounter = 0, fullStringLength = fullString.length(), wordFirstLetter = -1, i = 0;

bool currentLetterIsDelimiter = false, previousLetterIsDelimiter = false;

do

{

// проверка принадлежности символа буквам/разделителям

for (int j = 0; j < delimiterString.length(); j++)

if (fullString[i] == delimiterString[j])

{

currentLetterIsDelimiter = true;

break;

}

// получение индекса первой буквы заданного слова

if (!currentLetterIsDelimiter)

{

if (i == 0 || previousLetterIsDelimiter)

wordCounter++;

if (wordCounter == posWord)

if (wordFirstLetter == -1)

wordFirstLetter = i;

}

// получение слова с помощью индексов

if (wordCounter == posWord && (currentLetterIsDelimiter || i == fullStringLength))

{

int newSize = i - wordFirstLetter;

char\* tempChar = new char[newSize + 1];

for (int j = wordFirstLetter, k = 0; j < i; j++, k++)

tempChar[k] = fullString[j];

tempChar[newSize] = '\0';

string tempString = tempChar;

return tempString;

}

previousLetterIsDelimiter = currentLetterIsDelimiter;

currentLetterIsDelimiter = false;

} while (i++ < fullStringLength);

return nullString;

}

// проверить строку "userString" на корректность ввода команды

int checkString(string userString)

{

// получение и проверка наличия команды в строке

string command = toLower(getWord(userString, 1, delimiterString));

if (command == nullString)

return negativeValue;

// проверка наличия команды в базе данных

int commandIndex = -1;

for (int i = 0; i < commandsCount; i++)

if (command == commandString[i])

commandIndex = i;

if (commandIndex == negativeValue)

{

showMessage\_errorCommand

return negativeValue;

}

else return commandIndex;

}

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// \_\_\_\_\_\_\_\_\_\_ Функции работы с памятью \_\_\_\_\_\_\_\_\_\_

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// Записать в ячейку памяти "RAM" из файла "name" страницу под номером "elementPage"

int writeRAM(string name, int elementPage, structRAM &RAM)

{

ifstream fileR(name, ios::binary | ios::in);

if (!fileR.is\_open()) return -1;

fileR.seekg((elementPage - 1) \* pageSize, std::ios::beg);

for (int i = 0; i < (pageCountNum); i++)

fileR.read((char\*)&RAM.pageArray[i], sizeof(int));

RAM.pageNumber = elementPage;

fileR.close();

return 1;

}

// Считать из ячейки памяти "RAM" в файл "name"

int readRAM(string name, structRAM &RAM)

{

ofstream file(name, ios::in | ios::out | ios::binary);

if (!file.is\_open()) return -1;

file.seekp((RAM.pageNumber - 1) \* pageSize, std::ios::beg);

for (int i = 0; i < (pageCountNum); i++)

file.write((char\*)&RAM.pageArray[i], sizeof(int));

file.close();

return 1;

}

// Очистить память

void clearRAM(structRAM RAM[RAMCounter])

{

lastUseRAM = -1;

for (int i = 0; i < RAMCounter; i++)

{

RAM[i].pageNumber = -1;

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

RAM[i].pageArray[j] = 0;

}

}

// Найти номер ячейки памяти, которую можно перезаписать

int findRAMForWrite()

{

int numberRAMForWrite;

for (int i = 0; i < RAMCounter; i++)

{

if (RAM[i].pageNumber == negativeValue)

{

numberRAMForWrite = i;

break;

}

if (i != lastUseRAM)

numberRAMForWrite = i;

}

return numberRAMForWrite;

}

// Проверка на наличие элемента "element" в памяти

bool isElementInRAM(int element)

{

int elementPage = (element - 1) / (pageCountNum) + 1;

for (int i = 0; i < RAMCounter; i++)

if (RAM[i].pageNumber == elementPage)

return true;

return false;

}

// Сохранение страниц находящихся в памяти после закрытия программы

void saveRAM()

{

for (int i = 0; i < RAMCounter; i++)

{

if (RAM[i].pageNumber != negativeValue)

{

readRAM(lastUseFile, RAM[i]);

}

}

}

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// \_\_\_\_\_\_\_\_\_ Функции работы с командами \_\_\_\_\_\_\_\_\_

// \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

// создание файла с именем "name" размером "size" КБайт

int createFile(string name, int size)

{

int typeSize = sizeof(int), num;

ofstream file(name, ios::binary | ios::out);

if (!file) return -1;

for (int i = 0; i < size\*2; i++)

for (int j = 0; j < pageSize / typeSize; j++)

{

num = 0;

file.write((char\*)&num, sizeof(num));

}

file.close();

return 1;

}

// чтение элемента "element" из файла с именем "name"

int readFile(string name, int element)

{

ifstream file(name, ios::binary | ios::in);

if (!file.is\_open()) return -1;

int sizeReadValue = sizeof(readValue);

file.seekg((element - 1) \* sizeReadValue, std::ios::cur);

file.read((char\*)&readValue, sizeReadValue);

file.close();

return 1;

}

// замена элемента на позиции "element" значением "value"

int writeFile(string name, int element, int value)

{

ofstream file(name, ios::in | ios::out | ios::binary);

if (!file.is\_open()) return -1;

int sizeWriteValue = sizeof(element);

file.seekp((element - 1) \* sizeWriteValue, std::ios::beg);

file.write((char\*)&value, sizeWriteValue);

file.close();

return 1;

}

// вывод элементов файла "name"

int outputFile(string name)

{

ifstream file(name, ios::binary | ios::in);

if (!file) return -1;

file.seekg(0, std::ios::end);

int typeSize = sizeof(int), num, size = file.tellg() / 1024;

file.seekg(0, std::ios::beg);

for (int i = 0; i < size \* 2; i++)

for (int j = 0; j < pageSize / typeSize; j++)

{

file.read((char\*)&num, sizeof(num));

cout << num << " ";

}

file.close();

return 1;

}

// вызов команды по индексу "index"

void useCommand(int index, string userString)

{

switch (index)

{

case Create:

{

string name = getWord(userString, 2, delimiterString);

string sizeString = getWord(userString, 3, delimiterString);

float size;

// проверка параматров команды

if (name == helpString)

{

showMessage\_helpCreate

break;

}

if (isNum(sizeString) == negativeValue)

{

showMessage\_errorParam

break;

}

stringstream stringStream;

stringStream << sizeString;

stringStream >> size;

if (name == nullString || size < 0 || (size - int(size)) || getWord(userString, 4, delimiterString) != nullString)

{

showMessage\_errorParam

break;

}

// проверка создания файла

if (createFile(name, size) == negativeValue)

showMessage\_errorCreateFile

else

indentString

// очистка памяти при пересоздании файла

if (lastUseFile == name)

clearRAM(RAM);

break;

}

case Read:

{

string name = getWord(userString, 2, delimiterString); // Имя файла откуда считывается элемент

string elementString = getWord(userString, 3, delimiterString); // Значение элемента в строковом виде

float element; // Значение элемента в численном виде

int elementPage; // Страница на которой находится элемент

// очистка памяти при работе с новым файлом

if (lastUseFile != name)

{

if (lastUseFile != nullString)

clearRAM(RAM);

lastUseFile = name;

}

// проверка параматров команды

if (name == helpString)

{

showMessage\_helpRead

break;

}

if (isNum(elementString) == negativeValue)

{

showMessage\_errorParam

break;

}

stringstream stringStream;

stringStream << elementString;

stringStream >> element;

if (name == nullString || element <= 0 || (element - int(element)) || getWord(userString, 4, delimiterString) != nullString)

{

showMessage\_errorParam

break;

}

int maxCountElement;

ifstream file(name, ios::binary | ios::in);

if (!file.is\_open())

{

showMessage\_errorOpenFile

break;

}

file.seekg(0, std::ios::end);

maxCountElement = file.tellg() / sizeof(element);

file.close();

if (element > maxCountElement)

{

showMessage\_errorReadElement

break;

}

// получение читаемого значения в "readValue" или вывод об ошибке

// проверка наличия элемента в памяти

elementPage = (element - 1) / (pageCountNum) + 1;

int elementPositionOnPage = element - (elementPage - 1) \* pageCountNum - 1;

for (int i = 0; i < RAMCounter; i++)

if (elementPage == RAM[i].pageNumber)

{

readValue = RAM[i].pageArray[elementPositionOnPage];

lastUseRAM = i;

goto exitFromRead;

}

// элемента нет в памяти

// поиск ячейки для записи

int numberRAMForWrite = findRAMForWrite(); // Номер ячейки памяти в которую произведется запись страницы

lastUseRAM = numberRAMForWrite;

// запись страницы в память

if (RAM[numberRAMForWrite].pageNumber != negativeValue)

readRAM(name, RAM[numberRAMForWrite]); // swap

writeRAM(name, elementPage, RAM[numberRAMForWrite]);

// запоминаем читаемый элемент

readValue = RAM[lastUseRAM].pageArray[elementPositionOnPage];

exitFromRead:

cout << "Считанное значение: " << readValue << endl;

indentString

break;

}

case Write:

{

string name = getWord(userString, 2, delimiterString);

string elementString = getWord(userString, 3, delimiterString);

string valueString = getWord(userString, 4, delimiterString);

float element, value;

// очистка памяти при работе с новым файлом

if (lastUseFile != name)

{

if (lastUseFile != nullString)

clearRAM(RAM);

lastUseFile = name;

}

// проверка параматров команды

if (name == helpString)

{

showMessage\_helpWrite

break;

}

if (isNum(elementString) == negativeValue || isNum(valueString) == negativeValue)

{

showMessage\_errorParam

break;

}

stringstream stringStreamElement;

stringStreamElement << elementString;

stringStreamElement >> element;

stringstream stringStreamValue;

stringStreamValue << valueString;

stringStreamValue >> value;

if (name == nullString || element <= 0 || (element - int(element)) || (value - int(value)) || getWord(userString, 5, delimiterString) != nullString)

{

showMessage\_errorParam

break;

}

int maxCountElement;

ifstream file(name, ios::binary | ios::in);

if (!file.is\_open())

{

showMessage\_errorOpenFile

break;

}

file.seekg(0, std::ios::end);

maxCountElement = file.tellg() / sizeof(element);

file.close();

if (element > maxCountElement)

{

showMessage\_errorReadElement

break;

}

if (abs(value) > MAX)

{

showMessage\_errorWriteValue

break;

}

// получение читаемого значения в "readValue" или вывод об ошибке

// проверка наличия элемента в памяти

int elementPage = (element - 1) / (pageCountNum)+1;

int elementPositionOnPage = element - (elementPage - 1) \* pageCountNum - 1;

for (int i = 0; i < RAMCounter; i++)

if (elementPage == RAM[i].pageNumber)

{

//readValue = RAM[i].pageArray[elementPositionOnPage];

RAM[i].pageArray[elementPositionOnPage] = value;

lastUseRAM = i;

goto exitFromWrite;

}

int numberRAMForWrite = findRAMForWrite(); // Номер ячейки памяти в которую произведется запись страницы

lastUseRAM = numberRAMForWrite;

if (RAM[numberRAMForWrite].pageNumber != negativeValue)

readRAM(name, RAM[numberRAMForWrite]);

writeRAM(name, elementPage, RAM[numberRAMForWrite]);

RAM[lastUseRAM].pageArray[elementPositionOnPage] = value;

exitFromWrite:

indentString

break;

}

case Output:

{

string name = getWord(userString, 2, delimiterString);

// проверка параматров команды

if (name == helpString)

{

showMessage\_helpOutput

break;

}

if (name == nullString || getWord(userString, 3, delimiterString) != nullString)

{

showMessage\_errorParam

break;

}

// запись данных из памяти в файл

for (int i = 0; i < RAMCounter; i++)

if (RAM[i].pageNumber != negativeValue)

readRAM(lastUseFile, RAM[i]);

if (outputFile(name) == negativeValue)

showMessage\_errorOpenFile

else

indentString

break;

}

case Help:

{

string str = getWord(userString, 2, delimiterString);

// проверка параматров команды

if (str == helpString)

{

showMessage\_helpHelp

break;

}

if (str != nullString)

showMessage\_helpHelpParameter

else

{

showMessage\_helpHelpCommand

showMessage\_Create

showMessage\_Read

showMessage\_Write

showMessage\_Output

showMessage\_Help

showMessage\_Exit

indentString

}

break;

}

case Exit:

{

if (getWord(userString, 2, delimiterString) == helpString)

{

showMessage\_helpExit

break;

}

for (int i = 0; i < RAMCounter; i++)

if (RAM[i].pageNumber != negativeValue)

readRAM(lastUseFile, RAM[i]);

break;

}

}

}

void main()

{

setlocale(LC\_ALL, "Russian");

string userString; // Строка ввода

int commandIndex; // Индекс команды

do

{

showMessage\_podskazka

getline(cin, userString);

if ( (commandIndex = checkString(userString)) != negativeValue)

useCommand(commandIndex, userString);

}

while (toLower(getWord(userString, 1, delimiterString)) != commandString[Exit] || getWord(userString, 2, delimiterString) == helpString);

}