Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное  
учреждение высшего образования

Новосибирский государственный технический университет

Кафедра Автоматизированных систем управления

**ОТЧЁТ**

**по лабораторной работе № 1**

**«Моделирование системы управления виртуальной памятью»**

**по дисциплине: «Технология программирования»**

Выполнили: Проверил:

Студенты гр. «АВТ-813», «АВТФ» Лукоянычева Ольга Викторовна

Букова Анна

Ланг Татьяна

Пустовских Дмитрий

Новосибирск 2019

**Цель работы:** Получить представление о механизме управления виртуальной памятью, закрепить навыки программирования на выбранном языке с использованием динамических структур данных и прямого доступа к файлу.

**Содержание**

1. Описание программы…………………………………………………..…………. 3
   1. Общие сведения…………………………………………………..……………3
   2. Функциональное назначение……………………………………..………….. 3
   3. Описание логической структуры…………………………………..…………3

Классы………………………………………………………………….………3

Методы………………………………………………………………….……...3

Листинг…………………………………………………………………….….. 4

* 1. Вызов и загрузка……………………………………………………………...10
  2. Выходные данные……………………………………………………….……10

1. **Описание программы**
   1. **Общие сведения**

Данная программа должна моделировать работу виртуальной памяти страничной организации: вся память разбита на фрагменты-страницы, часть из которых находится в оперативной памяти, другие же находятся во внешней памяти.

Программа использует язык С++.

* 1. **Функциональное назначение**

Данная программа решает задачу моделирования работы страничной виртуальной памяти посредством работы с внешней памятью (файловой системой) и моделирования оперативной памяти как динамической структуры.

Файл, с которым осуществляется работа, задается пользователем.

* 1. **Описание логической структуры**

**Классы**

Программа использует объектно-ориентированный стиль написания. В рамках стиля и с целью выполнения поставленной задачи определяются следующие классы:

* buffer (работает с битовым файлом, представляет из себя 1 страницу)
* virtual\_memory (массив страниц)

**Методы**

Для выполнения поставленной задачи должны определяться следующие методы:

* set, vput– метод записи значения в файл.
* swap– метод загрузки в память новой страницы
* unswap– метод выгрузки в файл страницы
* get, vget– метод получения значения из файла

**Листинг**

#pragma once

#include <iostream>

#include <fstream>

#include <ctime>

#include "buffer.h"

using namespace std;

#ifndef LOOK\_VIRTUAL\_MEMORY

#define LOOK\_VIRTUAL\_MEMORY

template<typename T>

class virtual\_memory

{

public:

virtual\_memory(const char\* str, long sizeBuffer,int nPage);

~virtual\_memory();

void vput(long index, T value);

void clearBufer();

T operator[](long index);

T vget(long index);

private:

unsigned int nPage;

buffer<T>\* buffers;

int\* number;

int\* status;

};

template<typename T>

virtual\_memory<T>::virtual\_memory(const char\* str, long sizeBuffer, int nPage)

{

this->nPage = nPage;

buffers = new buffer<T>[nPage];

status = new int[nPage];

number = new int[nPage];

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

{

status[i] = 0;

number[i] = 0;

buffers[i].init(str, sizeBuffer);

}

}

template<typename T>

virtual\_memory<T>::~virtual\_memory()

{

delete[] buffers;

delete[] status;

delete[] number;

}

template<typename T>

inline void virtual\_memory<T>::vput(long index, T value)

{

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

{

int page = (index - 1) / (buffers[i].getSizeBuffer() / sizeof(T)) + 1;

if (number[i] == page)

{

int temp = 1 + (index - 1) % (buffers[i].getSizeBuffer() / sizeof(T));

buffers[i].setBuff(temp, value);

return;

}

}

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

{

if (status[i] == 0)

{

status[i] = 1;

number[i] = (index - 1) / (buffers[i].getSizeBuffer() / sizeof(T)) + 1;

buffers[i].set(index, value);

return;

}

}

srand(time(NULL));

int r = rand() % nPage;

number[r] = (index - 1) / (buffers[r].getSizeBuffer() / sizeof(T)) + 1;

buffers[r].set(index, value);

return;

}

template<typename T>

inline void virtual\_memory<T>::clearBufer()

{

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

{

if (!buffers[i].empty())

buffers[i].unswap();

}

}

template<typename T>

inline T virtual\_memory<T>::operator[](long index)

{

return vget(index);

}

template<typename T>

inline T virtual\_memory<T>::vget(long index)

{

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

{

int page = (index-1) / (buffers[i].getSizeBuffer() / sizeof(T)) + 1;

if (number[i] == page)

{

int temp = 1 + (index-1) % (buffers[i].getSizeBuffer() / sizeof(T));

return buffers[i].getBuff(temp);

}

}

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

{

if (status[i] == 0)

{

number[i] = (index - 1) / (buffers[i].getSizeBuffer() / sizeof(T)) + 1;

status[i] = 1;

return buffers[i].get(index);

}

}

srand(time(NULL));

int r = rand() % nPage;

number[r] = (index - 1) / (buffers[r].getSizeBuffer() / sizeof(T)) + 1;

return buffers[r].get(index);

}

#endif // !LOOK\_VIRTUAL\_MEMORY

#pragma once

#include <iostream>

#include <fstream>

#include <stdexcept>

using namespace std;

#ifndef LOOK\_BUFFER

#define LOOK\_BUFFER

template <typename T>

class buffer

{

public:

buffer();

buffer(const char\* str, long sizeBuffer);

void init(const char\* str, long sizeBuffer);

void setBuff(long index,T element);

void set(long index,T element);

T getBuff(long index);

long getSizeBuffer();

void swap(int page);

T get(long index);

int getSize();

long getPage();

void unswap();

bool empty();

~buffer();

private:

void cleanBuf();

void fileSize();

const char\* name\_File;

long m\_sizeBuffer;

char\* m\_buffer;

bool m\_empty;

int sizeFile;

int m\_bufnum;

};

template<typename T>

inline buffer<T>::buffer()

{

m\_sizeBuffer = 0;

name\_File = nullptr;

m\_empty = true;

this->m\_buffer = nullptr;

m\_bufnum = -1;

}

template <typename T>

buffer<T>::buffer(const char\* str,long sizeBuffer)

{

init(str, sizeBuffer);

m\_empty = true;

m\_bufnum = -1;

}

template <typename T>

buffer<T>::~buffer()

{

if (!m\_empty)

unswap();

}

template<typename T>

inline void buffer<T>::init(const char\* str, long sizeBuffer)

{

name\_File = str;

m\_sizeBuffer = sizeBuffer;

this->m\_buffer = new char[sizeBuffer];

fileSize();

}

template<typename T>

void buffer<T>::swap(int page)

{

if (!m\_empty)

unswap();

if (this->m\_bufnum == page)

return;

this->m\_bufnum = page;

fstream m\_file(name\_File, ios::binary | ios::out | ios::in);

m\_file.seekg(0, ios::beg);

if(m\_buffer == nullptr)

this->m\_buffer = new char[m\_sizeBuffer];

m\_file.seekg(page \* m\_sizeBuffer, ios::beg);

m\_file.read(this->m\_buffer, m\_sizeBuffer);

m\_empty = false;

m\_file.close();

}

template<typename T>

inline void buffer<T>::unswap()

{

if (m\_empty)

return;

fstream m\_file(name\_File, ios::binary | ios::out | ios::in);

m\_file.seekg(0, ios::beg);

m\_file.seekg(m\_bufnum \* m\_sizeBuffer, ios::beg);

m\_file.write(m\_buffer, m\_sizeBuffer);

m\_file.close();

cleanBuf();

}

template<typename T>

inline bool buffer<T>::empty()

{

return this->m\_empty;

}

template<typename T>

inline T buffer<T>::getBuff(long index)

{

if (m\_empty)

throw new logic\_error("The buffer is empty");

if (index < 1 || index >(m\_sizeBuffer / sizeof(T)))

throw new out\_of\_range("The index of the element goes beyond the bounds of the array");

int len = sizeof(T);

char\* inBof = new char[len];

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

{

inBof[i] = m\_buffer[(index - 1)\* len + i];

}

T\* rez = (T\*)inBof;

return \*rez;

}

template<typename T>

inline T buffer<T>::get(long index)

{

if (index < 1 || index >(sizeFile / sizeof(T)))

throw new out\_of\_range("The index of the element goes beyond the bounds of the array");

int page = (index-1) / (m\_sizeBuffer / sizeof(T));

int newIn = (index-1) % (m\_sizeBuffer / sizeof(T));

this->swap(page);

return getBuff(newIn + 1);

}

template<typename T>

inline int buffer<T>::getSize()

{

return this->sizeFile;

}

template<typename T>

inline long buffer<T>::getPage()

{

return m\_bufnum;

}

template<typename T>

inline void buffer<T>::setBuff(long index, T element)

{

if (m\_empty)

throw new logic\_error("The buffer is empty");

if (index < 1 || index >(m\_sizeBuffer / sizeof(T)))

throw new out\_of\_range("The index of the element goes beyond the bounds of the array");

char\* inBof = (char\*)&element;

int len = sizeof(T);

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

{

m\_buffer[(index - 1)\*len + i ] = inBof[i];

}

}

template<typename T>

inline void buffer<T>::set(long index, T element)

{

if (index < 1 || index > (sizeFile/ sizeof(T)))

throw new out\_of\_range("The index of the element goes beyond the bounds of the array");

int page = (index-1) / (m\_sizeBuffer/ sizeof(T));

int newIn = (index-1) % (m\_sizeBuffer / sizeof(T));

this->swap(page);

setBuff(newIn + 1, element);

}

template<typename T>

inline long buffer<T>::getSizeBuffer()

{

return m\_sizeBuffer;

}

template<typename T>

inline void buffer<T>::cleanBuf()

{

if (m\_buffer == nullptr)

return;

delete[] this->m\_buffer;

m\_buffer = nullptr;

m\_empty = true;

}

template<typename T>

inline void buffer<T>::fileSize()

{

fstream m\_file(name\_File, ios::binary | ios::out | ios::in);

m\_file.seekg(0, ios::end);

sizeFile = m\_file.tellg();

m\_file.close();

}

#endif // !LOOK\_BUFFER

* 1. **Вызов и загрузка**

Работа программы начинается с исполнения метода «main».

Вызов к исполнению программы происходит из интегрированной среды разработки.

* 1. **Выходные данные**

Программа не возвращает значений, после завершения работы формируется бинарный файл с результатами работы – записями заданных чисел.