МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА № 43

ОТЧЁТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Старший преподаватель | |  |  | | | |  | | М. Д. Поляк |
| должность, уч. степень, звание | |  | подпись, дата | | | |  | | инициалы, фамилия |
| ОТЧЁТ О ЛАБОРАТОРНОЙ РАБОТЕ №5 | | | | | | | | | | |
| Управление памятью | | | | | | | | | | |
| по дисциплине: Операционные системы | | | | | | | | | | |
|  | | | | | | | | | | |
| РАБОТУ ВЫПОЛНИЛА | | | | | | | | | | |
| СТУДЕНТКА ГР. | 4931 | | |  | 01.05.2022 |  | | Е.Ю. Ильченко | | |
|  |  | | |  | подпись, дата |  | | инициалы, фамилия | | |
|  |  | | |  |  |  | |  | | |

Санкт-Петербург 2022

## Вариант 11

## Цель:

Знакомство с принципами организации виртуальной памяти.

#### Задание:

В данной работе необходимо реализовать фрагмент диспетчера памяти и часть функционала операционной системы, отвечающего за замещение страниц при возникновении ошибок отсутствия страниц. Для упрощения работы предполагается использование линейной инвертированной таблицы страниц, работу с которой необходимо реализовать в виде программы. Также для простоты предполагается, что в системе имеется один единственный процесс, поэтому идентификатор процесса в инвертированной таблице страниц не хранится. Входные данные представляют собой последовательность операций обращения к памяти, выходные данные - состояние инвертированной таблицы страниц после каждой операции обращения к памяти.

1. Вычислить номер варианта по списку в журнале и сохранить его в файл [TASKID.txt](https://github.com/suai-os-2022/os-task5-Kateilc/blob/master/TASKID.txt) в репозитории.
2. Написать программу на языке C++ в соответствии со следующей спецификацией.
   1. Входные данные:
      1. Аргумент командной строки (число): номер алгоритма замещения страниц, который должна использовать программа. Принимает значения 1 или 2, соответствующие двум алгоритмам замещения страниц, заданным по варианту.
      2. Перечень инструкций обращения к памяти, считываемый программой из стандартного потока ввода. На каждой строке не более одной инструкции. Инструкция состоит из двух чисел, разделенных пробелом, например: 0 1. Первое число обозначает тип операции доступа к памяти: 0 - чтение и 1 - запись. Второе число является номером виртуальной страницы, к которой происходит обращение.
   2. Выходные данные:
      1. Для каждой операции обращения к памяти, информация о которой поступила на вход программы, на выходе должна быть сгенерирована строка, содержащая содержимое инвертированной таблицы страниц в виде последовательности номеров виртуальных страниц, разделенных пробелом. Если какая-либо из записей в таблице страниц отсутствует (таблица страниц не заполнена до конца), вместо номера виртуальной страницы необходимо вывести символ #.
3. Весь код поместить в файле lab4.cpp. Код должен корректно компилироваться командой g++ lab4.cpp -o lab4 -std=c++11. Настоятельно рекомендуется использовать стандартную библиотеку STL. Полезными могут быть контейнеры [list](https://en.cppreference.com/w/cpp/container/list), [vector](https://en.cppreference.com/w/cpp/container/vector), [bitset](https://en.cppreference.com/w/cpp/utility/bitset) и др.
4. Если в работе алгоритма замещения страниц используется бит R, то необходимо реализовать эмуляцию прерывания таймера. Для этого через каждые 5 операций обращения к памяти необходимо запускать обработчик данного прерывания. Значения битов R по прерыванию таймера сбрасываются.
5. Для алгоритмов, использующих счетчик (NFU, Aging): если несколько страниц имеют одинаковое значение счетчика, одна из них выбирается случайным образом. При повторной загрузке страницы в память ее счетчик обнуляется. В алгоритме старения счетчик имеет размер 1 байт. В алгоритме NFU счетчик имеет размер не меньше 4 байт.
6. Во всех алгоритмах, использующих датчик случайных чисел (Random, NRU, NFU, Aging, ...), разрешается использовать **только** функцию int uniform\_rnd(int a, int b), объявленную в файле [lab4.h](https://github.com/suai-os-2022/os-task5-Kateilc/blob/master/lab4.h). Данная функция генерирует случайное целое число с равномерным распределением из диапазона [a, b]. Использование других функций для работы со случайными числами запрещено!
7. В качестве системного времени в алгоритме рабочего набора следует использовать количество инструкций доступа к памяти, обработанных с момента запуска программы.
8. После успешного прохождения локальных тестов необходимо загрузить код в репозиторий на гитхабе.
9. Сделать выводы об эффективности реализованных алгоритмов замещения страниц. Сравнить количество ошибок отсутствия страниц, генерируемых на тестовых данных при использовании каждого алгоритма.

|  |  |  |  |
| --- | --- | --- | --- |
| Номер варианта | Количество страничных блоков | Алгоритм 1 | Алгоритм 2 |
| 11 | 10 | |  |  | | --- | --- | | Сlock |  | | Aging |

**Описание используемых алгоритмов замещения страниц**

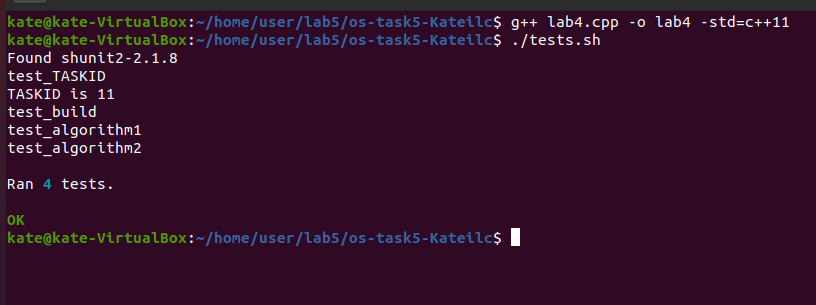
**Clock**

Дальнейшая модификация алгоритма Второй шанс. Поскольку перемещать страницы из головы очереди в конец весьма неэффективно, для увеличения скорости работы алгоритма Второй шанс очередь может быть реализована в виде циклического списка. В этом случае фактического перемещения страниц из головы в конец очереди не происходит. Вместо этого изменяется значение переменной, указывающей на голову очереди. В этом алгоритме переменная-указатель ("стрелка" часов) всегда указывает на голову списка, т.е. на страницу, загруженную раньше всех остальных.

**Aging**

Дальнейшая модификация алгоритмов LRU и NFU. Для каждой страницы, загруженной в память, заводится счетчик. По прерыванию от таймера счетчики всех страниц сдвигаются вправо на 1 бит. Затем, самый старший (левый) бит счетчика устанавливается равным значению бита использования (R) этой страницы. В случае возникновения ошибки отсутствия страницы, удаляется та страница, значение счетчика которой минимально. Если имеется несколько страниц с одинаковым минимальным значением счетчика, необходимо выбрать одну из них случайным образом.

**Результат выполнения работы программы:**

****

**Исходный код программы с комментариями**

#include <iostream>

#include "lab4.h"

#include <iostream>

#include <bitset>

using namespace std;

const int amountPages = 10;

//структура для алгоритма aging

struct RowString {

int numRow; //номер страницы

bool bit\_R, bit\_M;//для бита m и r

bitset<8> cnt = 0;

RowString(int numRow, bool bit\_R, bool bit\_M) {

this->numRow = numRow;

this->bit\_R = bit\_R;

this->bit\_M = bit\_M;

}

};

//структура для алгоритма clock

struct rowtb {

int numRow;

bool bit\_R;

rowtb(int numRow, bool bit\_R) {

this->numRow = numRow;

this->bit\_R = bit\_R;

}

};

//структура для спсиска

struct list

{

rowtb\* row = NULL;

struct list\* next; // указатель на следующий элемент

};

void deleteList(list\* lst) {

struct list\* lst\_next = lst;

while (lst\_next->next != NULL) {

lst = lst\_next;

lst\_next = lst\_next->next;

delete lst;

}

delete lst\_next;

}

//найти размер списка

int listSize(list\* lst) {

int size = 0;

struct list\* lst\_next = NULL;

while (lst\_next != lst) {

if (lst\_next == NULL) lst\_next = lst;

size++;

lst\_next = lst\_next->next;

}

return size;

}

//инициализация списка

struct list\* init(rowtb\* row) // а- значение первого узла

{

struct list\* lst;

// выделение памяти под корень списка

lst = (struct list\*)malloc(sizeof(struct list));

lst->row = row;

lst->next = lst; // указатель на сам корневой узел

return(lst);

}

//добавить элемент

void addelem(list\* lst, rowtb\* new\_row)

{

struct list\* temp = lst;

while (temp->next != lst) {

temp = temp->next;

}

struct list\* new\_elem;

new\_elem = (struct list\*)malloc(sizeof(struct list));

new\_elem->row = new\_row;

new\_elem->next = lst;

temp->next = new\_elem;

}

//проверка наличия страницы

bool hasPages(list\* lst, int num\_row) {

struct list\* lst\_next = NULL;

while (lst\_next != lst) {

if (lst\_next == NULL)lst\_next = lst;

if (lst\_next->row->numRow == num\_row)

return true;

lst\_next = lst\_next->next;

}

return false;

}

//изменить стреницу

void swapPages(list\* lst, rowtb\* new\_row, list\*& pointer) {

struct list\* lst\_next = NULL;

while (lst\_next != pointer) {

if (lst\_next == NULL)lst\_next = pointer;

if (!lst\_next->row->bit\_R) {

lst\_next->row = new\_row;

pointer = lst\_next->next;

break;

}

else {

lst->row->bit\_R = false;

lst\_next = lst\_next->next;

}

}

}

//возвращает номер страницы по указанной позиции(для вывода)

int showPage(list\* lst, int pagePos) {

int pos = 0;

struct list\* lst\_next = NULL;

while (lst\_next != lst) {

if (lst\_next == NULL)lst\_next = lst;

if (pos == pagePos)

return lst\_next->row->numRow;

pos++;

lst\_next = lst\_next->next;

}

return -1;

}

//вывод списка на экран

void listprint(list\* lst)

{

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

{

if (i < listSize(lst))

cout << showPage(lst, i);

else cout << "#";

if (i != amountPages - 1)

cout << " ";

}

cout << endl;

}

//сброс регистров в ноль

void changeRegs(list\* lst) {

struct list\* lst\_next = NULL;

while (lst\_next != lst) {

if (lst\_next == NULL)lst\_next = lst;

lst\_next->row->bit\_R = false;

lst\_next = lst\_next->next;

}

}

//изменения регистра с 0 на 1

void setRegs(list\* lst, int row\_num) {

struct list\* lst\_next = NULL;

while (lst\_next != lst) {

if (lst\_next == NULL)lst\_next = lst;

if (lst\_next->row->numRow == row\_num)

lst\_next->row->bit\_R = true;

lst\_next = lst\_next->next;

}

}

//

void algclock()

{

list\* pointer = NULL;//стрелка(страница добавленная раньше всех)

list\* lst = NULL;//указатель на голову

int timer = 0;//для отсчета 5 тактов

while (cin)

{

int comm, num\_page;

cin >> comm;

cin >> num\_page;

if (!cin) { break; }

timer++;

if (lst == NULL) {

rowtb\* new\_row = new rowtb(num\_page, 1);

lst = init(new\_row);

pointer = lst;

}

else {

if (hasPages(lst, num\_page)) {//если страница есть, устанавливаем би R в 1

setRegs(lst, num\_page);

}

else {

if (listSize(lst) < amountPages) { //проверяем заполнен ли список

rowtb\* new\_row = new rowtb(num\_page, 1);

addelem(lst, new\_row); //просто добавляем в конец

}

else {

rowtb\* new\_row = new rowtb(num\_page, 1);

swapPages(lst, new\_row, pointer);//заменяем страницу

}

}

}

listprint(lst);

if (timer % 5 == 0)

changeRegs(lst);//при каждом пятом цикле сброс регистров в 0

}

//deleteList(lst);

}

//проверка есть ли такая страница

bool ListExists(vector<RowString\*>& rows, int Row) {

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

if (rows[i]->numRow == Row) {

return true;

}

}

return false;

}

//изменения бита

void changeRegs(vector<RowString\*>& rows, int Row, int type) {

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

if (rows[i]->numRow == Row) {

rows[i]->bit\_R = 1;

if (type == 1)

rows[i]->bit\_M = 1;

}

}

}

//изменение счетчика

void shiftCounter(vector<RowString\*>& rows) {

bitset<8> temp;

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

temp = rows[i]->cnt;

temp >>= 1;

if (rows[i]->bit\_R == 1)

temp.set(7, 1);

rows[i]->cnt = temp;

rows[i]->bit\_R = 0;

}

}

//находим страницу

unsigned long findMinAge(vector<RowString\*>& rows) {

unsigned long min\_age = rows[0]->cnt.to\_ulong();

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

if (rows[i]->cnt.to\_ulong() < min\_age)

min\_age = rows[i]->cnt.to\_ulong();

}

return min\_age;

}

//изменяем страницу

void changeStr(vector<RowString\*>& rows, int row, int type) {

vector<RowString\*> foundStr;

unsigned long min\_age = findMinAge(rows);

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

if (rows[i]->cnt.to\_ulong() == min\_age)

foundStr.push\_back(rows[i]);

}

int pageToDelete = uniform\_rnd(0, foundStr.size() - 1);

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

if (rows[i]->numRow == foundStr[pageToDelete]->numRow) {

if (type == 1)

rows[i] = new RowString(row, true, true);

else

rows[i] = new RowString(row, true, false);

}

}

}

//алгоритм старения

void AGING()

{

vector<RowString\*> rows;

int timer = 0;

while (cin)

{

int comm, num\_page;

cin >> comm;

cin >> num\_page;

if (!cin) { break; }

timer++;

if (ListExists(rows, num\_page))

{

changeRegs(rows, num\_page, comm);

}

else {

if (rows.size() < amountPages) {

RowString\* row;

if (comm == 1)

row = new RowString(num\_page, true, true);

else

row = new RowString(num\_page, true, false);

rows.push\_back(row);

}

else {

if (comm == 1)

changeStr(rows, num\_page, 1);

else

changeStr(rows, num\_page, 0);

}

}

if (timer % 5 == 0)

shiftCounter(rows);

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

{

if (i < rows.size())

cout << rows[i]->numRow;

else cout << "#";

if (i != amountPages - 1)

cout << " ";

}

cout << endl;

}

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

delete rows[i];

}

}

int main(int argc, char\* argv[])

{

int alg = 0;

if (argc > 1) {

alg = atoi(argv[1]);

}

if (alg == 1)

algclock();

if (alg == 2)

AGING();

return 0;

}

**Выводы:**

В процессе выполнения лабораторной работы ознакомились с принципами организации виртуальной памяти. Были реализованы алгоритмы Clock и Aging.