МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ

И ПРОГРАММНОЙ ИНЖЕНЕРИИ (КАФЕДРА №43)

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Старший препо даватель |  |  |  | М.Д. Поляк |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ОТЧЁТ О ЛАБОРАТОНОЙ РАБОТЕ № 5 | | | | | |
| по дисциплине: ОПЕРАЦИОННЫЕ СИСТЕМЫ | | | | | |
| **ЛР5. Управление памятью** | | | | | |
|  | | | | | |
| РАБОТУ ВЫПОЛНИЛ | | | | | |
| СТУДЕНТ ГР. | 4936 |  |  |  | Е.А. Цыганкова |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург  
2021

**Цель работы:**

Знакомство с принципами организации виртуальной памяти.

**Задание на лабораторную работу**

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-tsugankova-ekaterina/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-tsugankova-ekaterina/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 |
| 20 | 5 | Random | Working set |

**Исходный код программы с комментариями**

#include <iostream>

#include <bitset>

#include "lab4.h"

using namespace std;

const int N = 5;

struct MyStruct

{

int vpn = -1;

bool R = 0;

bool M = 0;

bitset<8> t{ "00000000" };

};

void clearR(MyStruct table[]) {

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

table[i].R = 0;

}

//очищение всех битов R

}

bitset<8> myincrement(bitset<8> in) {

// add 1 to each value, and if it was 1 already, carry the 1 to the next.

for (size\_t i = 0; i < 8; ++i) {

if (in[i] == 0) { // There will be no carry

in[i] = 1;

break;

}

in[i] = 0; // This entry was 1; set to zero and carry the 1

}

return in;

}

void Random(MyStruct table[],bool a, int b, bool f, bitset<8> timer) {

int i = uniform\_rnd(0, N - 1);

table[i].vpn = b;

table[i].R = 1;

if (a == 1)

table[i].M = 1;

else

table[i].M = 0;

if(f)

table[i].t = timer;

}

int findvpn(MyStruct table[], int b) {

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

if (table[i].vpn == b)

return i;

}

return -1;

}

bool fullSubtractor(bool b1, bool b2, bool& borrow)

{

bool diff;

if (borrow) {

diff = !(b1 ^ b2);

borrow = !b1 || (b1 && b2);

}

else {

diff = b1 ^ b2;

borrow = !b1 && b2;

}

return diff;

}

// Function to calculate difference between two bitsets

bitset<8> mybitsetSubtract(bitset<8> x, bitset<8> y)

{

bool borrow = false;

// bitset to store the sum of the two bitsets

bitset<8> ans;

for (int i = 0; i < 8; i++) {

ans[i] = fullSubtractor(x[i], y[i], borrow);

}

return ans;

}

bool compare(const std::bitset<8>& x, const std::bitset<8>& y)

{

for (int i = 7; i >= 0; i--) {

if (x[i] ^ y[i]) return y[i];

}

return false;

}

void WorkingSet(MyStruct table[], bitset<8> timer, int a, int b) {

int s = 0; int sm = 0;

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

if (table[i].R == 1) {

table[i].t = timer;

s++;

}

}

if (s != N) {//не у всех стоит бит R

bitset<8> diff[N];

for (int i = 0; i < N; i++) //подсчет разницы времени

diff[i] = mybitsetSubtract(timer, table[i].t );

int imax = 0; //Присвоение номеру наибольшего элемента номер нулевого

for (int i = 1; i < N; i++)

{//max < i

if (compare(diff[imax], diff[i]))//Сравнение

imax = i;

}

bool f(0);

vector <int> indexs;

int i(0);

while (i < N) {

if (diff[i] == diff[imax]) {

indexs.push\_back(i);

if(i != imax)

f = 1;

}

i++;

}

if (f) {

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

if (table[indexs[i]].M == 1)

sm++;

}

if (sm == indexs.size()) {//У всех установлены биты M

int i = uniform\_rnd(0, indexs.size() - 1);

table[indexs[i]].vpn = b;

table[indexs[i]].R = 1;

table[indexs[i]].t = timer;

if (a == 1)

table[indexs[i]].M = 1;

else

table[indexs[i]].M = 0;

}

else {//ищем строку, где бит M == 0

vector <int> inm;

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

if (table[indexs[i]].M == 0)

inm.push\_back(indexs[i]);

}

int i;

i = uniform\_rnd(0, inm.size() - 1);

table[inm[i]].vpn = b;

table[inm[i]].R = 1;

table[inm[i]].t = timer;

if (a == 1)

table[inm[i]].M = 1;

else

table[inm[i]].M = 0;

}

}

else {

table[imax].vpn = b;

table[imax].R = 1;

table[imax].t = timer;

if (a == 1)

table[imax].M = 1;

else

table[imax].M = 0;

}

}

else {//У всех установлены биты R

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

if (table[i].M == 1)

sm++;

}

if (sm == N) {//У всех установлены биты M

Random(table, a, b, 1, timer);

}

else {//ищем строку, где бит M == 0

vector <int> inm;

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

if (table[i].M == 0)

inm.push\_back(i);

}

int i;

i = uniform\_rnd(0, inm.size() - 1);

table[inm[i]].vpn = b;

table[inm[i]].R = 1;

table[inm[i]].t = timer;

if (a == 1)

table[inm[i]].M = 1;

else

table[inm[i]].M = 0;

}

}

}

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

{

if (argc > 1) {

if (argv[1]== string("1") || argv[1] == string("2")) {

int a, b, i(0);

bitset<8> timer{00000000};

MyStruct table[N];

while (cin >> a >> b) {

int f = findvpn(table, b);

if (f != -1) {//страница старая

table[f].R = 1;

if (a == 1)

table[f].M = 1;

else

table[f].M = 0;

}

else if (i < N) {//место есть

table[i].vpn = b;

table[i].R = 1;

if (a == 1)

table[i].M = 1;

else

table[f].M = 0;

i++;

}

else if(i>=N) {//мест нет, нужно замещать

if (argv[1] == string("1")) {

Random(table, a, b, 0, timer);

}

else {

WorkingSet(table, timer, a, b);

}

}

timer = myincrement(timer);

if (timer.to\_ulong() % 5 == 0) {

//cout << timer.to\_ulong();

clearR(table);

}

//вывод

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

if (table[i].vpn == -1)

cout << "#";

else

cout << table[i].vpn;

if(i!=(N-1))

cout << ' ';

else

cout << '\n';

//cout << "(" << table[i].R << "," << table[i].M << "," << table[i].t << ")" << " ";

}

//cout << '\n';

}

}

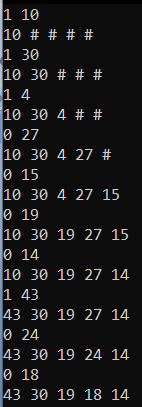
}

return 0;

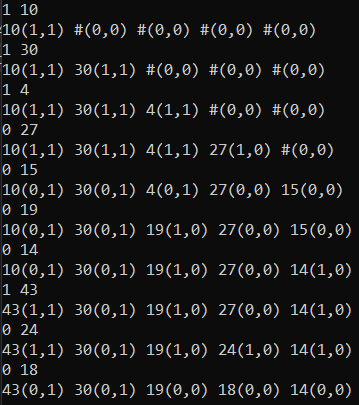
}

**Результат выполнения работы**

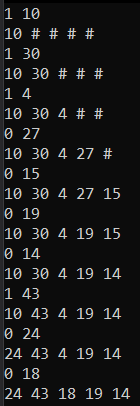
**Алгоритм Random**

**…**

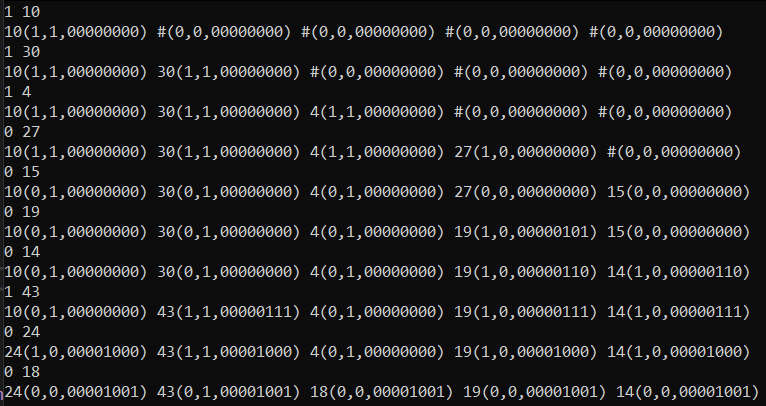
**Вывод с дополнительной информацией:**

**…**

**Алгоритм Working Set**

…

**Вывод с дополнительной информацией:**

…

**Выводы**

В результате выполнения лабораторной работы можно сделать вывод о том, что алгоритм Working set является более эффективным, чем алгоритм Random, так как учитывает возраст страниц и биты обращения и модификации. Но количество ошибок отсутствия страниц, генерируемых на данных тестовых данных при использовании алгоритма «Рабочего набора», равно 53, тогда как при использовании алгоритма Random – 47. Для проверки эффективности необходимо провести больше тестов.