МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Санкт-Петербургский государственный университет
аэрокосмического приборостроения»

КАФЕДРА 43

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

РУКОВОДИТЕЛЬ

Ст. преподаватель		М.Д. Поляк
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

Отчет о лабораторной работе №2
Разработка многопоточного приложения средствами POSIX

По дисциплине: ОПЕРАЦИОННЫЕ СИСТЕМЫ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. № 4134К		Самарин Д.В
	подпись, дата	инициалы, фамилия

Санкт-Петербург 2024

Цель работы:

Знакомство с принципами организации виртуальной памяти

Индивидуальное задание:

Номер варианта	Количество страничных блоков	Алгоритм 1	Алгоритм 2
16	5	FIFO	Working set

Результат выполнения работы

```
dmitry@dmitry-VirtualBox:~/lab1/os-task4-dYGamma$ ./tests.sh
Downloading shunit2-2.1.8 ...
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 52458    0 52458    0     0  94172      0 --:--:-- --:--:-- --:--:-- 94348
test_TASKID
TASKID is 16
test_build
test_algorithm1
test_algorithm2

Ran 4 tests.

OK
dmitry@dmitry-VirtualBox:~/lab1/os-task4-dYGamma$
```

Рисунок 1 – результат прохождения тестирования

Исходный код программы с комментариями

```
#include <iostream>
#include <deque>
#include <sstream>
#include <string>
#include <vector>
#include <algorithm>

#include "lab4.h"

static int const tableSize = 5;

static int actualFifoRow = 0;

static int systemCounter = 0;

bool debug = false;

struct FIFORow {
    int vpn = -1;
};
std::deque<FIFORow> fifoTable(tableSize);
```

```
void printFifoTable() {
    for (int i = 0; i < tableSize; ++i) {
        if (fifoTable[i].vpn != -1) {
            std::cout << fifoTable[i].vpn;
        }
        else {
            std::cout << "#";
        }
        if (i < tableSize - 1) {
            std::cout << " ";
        }
    }
    std::cout << "\n";
}

bool checkFifoTable(int vpn) {
    for (auto &row : fifoTable) {
        if (row.vpn == vpn) {
            return false;
        }
    }
    return true;
}

void fifoAlg(){
    std::string input;
    int command = 0;
    int vpn = 0;

    while (true) {
        std::getline(std::cin, input);
        if (input.empty()) {
            return;
        }
        std::istringstream istream{ input };
        if (!(istream >>  command) || !(istream >> vpn) || istream >> input)
{
            continue;
        }

        if (checkFifoTable(vpn)){
            fifoTable[(actualFifoRow % tableSize)].vpn = vpn;
            actualFifoRow++;
        }
        printFifoTable();
    }
}

struct WSRow {
    int vpn;
```

```
    bool flagR;
    bool flagM;
    int systemTime;
};
std::vector<WSRow> wsTable;

bool checkWSTable(int vpn, int command) {
    for (auto &row : wsTable) {
        if (row.vpn == vpn) {
            row.flagR = true;
            row.flagM = command;
            return false;
        }
    }
    return true;
}

void wsAlgImplementation(int command, int vpn) {

    if(!checkWSTable(vpn, command)){
        if (debug)
            std::cout<<"page : "<< vpn << " time : " << systemCounter <<
"\n" ;
        return;
    }

    for (auto &row : wsTable) {
        if (row.flagR) {
            row.systemTime = systemCounter; // age = 0
        }
    }

    bool isFlagM = command == 1;

    if (wsTable.size() != tableSize)
    {
        wsTable.push_back({vpn, true, isFlagM, systemCounter});
        return;
    }

    auto notNullAgeIter = std::find_if(std::begin(wsTable),
std::end(wsTable), [](WSRow page){
        if (page.systemTime != systemCounter)
            return true;
        return false;
    });

    if (notNullAgeIter == std::end(wsTable))
    {
        std::vector<int> indexesVec;
```

```
        for (int i = 0; i < tableSize; ++i)
        {
            if (!wsTable.at(i).flagM)
                indexesVec.push_back(i);
        }

        if (!indexesVec.empty())
        {
            int index = uniform_rnd(0, indexesVec.size() - 1);
            wsTable[indexesVec.at(index)] = {vpn, true, isFlagM,
systemCounter};
            return;
        }

        wsTable[uniform_rnd(0, wsTable.size() - 1)] = {vpn, true, isFlagM,
systemCounter};
        return;
    }


    std::vector<int> oldestPagesIndexes;
    std::vector<int> oldestPagesIndexesNotModifyed;
    int oldestAge = 0;
    for (int i = 0; i < tableSize; ++i)
    {
        int age = systemCounter - wsTable[i].systemTime;
        if (age == oldestAge)
        {
            if (!wsTable[i].flagM)
                oldestPagesIndexesNotModifyed.push_back(i);
            else
                oldestPagesIndexes.push_back(i);
            continue;
        }

        if (age > oldestAge)
        {
            oldestAge = age;
            oldestPagesIndexes.clear();
            oldestPagesIndexesNotModifyed.clear();
            if (!wsTable[i].flagM)
                oldestPagesIndexesNotModifyed.push_back(i);
            else
                oldestPagesIndexes.push_back(i);
            continue;
        }
    }

    int index = 0;
    if (!oldestPagesIndexesNotModifyed.empty())
```

```
    {
        index = uniform_rnd(0, oldestPagesIndexesNotModifyed.size() - 1);
        wsTable[oldestPagesIndexesNotModifyed.at(index)] = {vpn, true,
isFlagM, systemCounter};
        return;
    }

    index = uniform_rnd(0, oldestPagesIndexes.size() - 1);
    wsTable[oldestPagesIndexes.at(index)] = {vpn, true, isFlagM,
systemCounter};

}

void printWSTable() {
    for (int i = 0; i < tableSize; ++i) {
        if (i < wsTable.size()) {
            if (debug)
                std::cout << wsTable[i].vpn << "(" << wsTable[i].flagR <<
"," << wsTable[i].flagM << "," << wsTable[i].systemTime << ")";
            else
                std::cout << wsTable[i].vpn;
        }
        else {
            std::cout << "#";
        }
        if (i < tableSize - 1) {
            std::cout << " ";
        }
    }
    std::cout << "\n";
}



void wsAlgo() {

    std::string input;
    int command = 0;
    int vpn = 0;

    while (true) {
        std::getline(std::cin, input);
        if (input.empty()) {
            return;
        }
        std::istringstream istream{ input };
        if (!(istream >>  command) || !(istream >> vpn) || istream >> input)
{
            continue;
        }
```

```
        wsAlgImplementation(command, vpn);
        systemCounter++;
        if (systemCounter % 5 == 0)
        {
            for (auto &virtualPage : wsTable) {
                virtualPage.flagR = 0;
            }
        }

        printWSTable();
    }
}

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

    switch (atoi(argv[1])) {
    case 1:
    {
        fifoAlg();
        break;
    }
    case 2:
    {
        wsAlgo();
        break;
    }
    default:
        break;
    }

    return 0;
}
```

Выводы

В ходе работы мы познакомились с принципами организации виртуальной памяти.