ГУАП

КАФЕДРА № 43

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

ст. преп.		М.Д. Поляк
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №4

Управление памятью

по курсу: ОПЕРАЦИОННЫЕ СИСТЕМЫ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. № 4131	16.04.2024	А. А. Тафеева
	подпись, дата	инициалы, фамилия

Санкт-Петербург 2024

Цель работы

Знакомство с принципами организации виртуальной памяти.

Задание

Вариант 21.

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.

Вычислить номер варианта по списку в журнале и сохранить его в файл TASKID.txt в репозитории.

Написать программу на языке C++ в соответствии со следующей спецификацией.

Входные данные:

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.

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.

Выходные данные:

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 #.

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, vector, bitset и др.

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.

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.

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. Данная функция генерирует случайное целое число с равномерным распределением из диапазона [a, b]. Использование других функций для работы со случайными числами запрещено!

В качестве системного времени в алгоритме рабочего набора следует использовать количество инструкций доступа к памяти, обработанных с момента запуска программы.

После успешного прохождения локальных тестов необходимо загрузить код в репозиторий на гитхабе.

Сделать выводы об эффективности реализованных алгоритмов замещения страниц. Сравнить количество ошибок отсутствия страниц, генерируемых на тестовых данных при использовании каждого алгоритма.

Подготовить отчет о выполнении лабораторной работы и загрузить его под именем report.pdf в репозиторий. В случае использования системы компьютерной верстки LaTeX также загрузить исходный файл report.tex.

Описание используемых алгоритмов

FIFO

Данный алгоритм является одним из самых простых в реализации. Операционная система ведет очередь загруженных в память страниц. При загрузке в память новой

страницы она добавляется в конец очереди. При возникновении ошибки отсутствия страницы место под новую страницу высвобождается путем удаления страницы из головы очереди.

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 работы программы

```
1 10
10 # # # # # # # # # # # # # #
1 30
10 30 # # # # # # # # # # # # #
1 4
10 30 4 # # # # # # # # # # # #
0 27
10 30 4 27 # # # # # # # # # # #
0 15
10 30 4 27 15 # # # # # # # # # #
0 19
10 30 4 27 15 19 # # # # # # # # #
0 14
10 30 4 27 15 19 14 # # # # # # # #
1 43
10 30 4 27 15 19 14 43 # # # # # # #
0 24
10 30 4 27 15 19 14 43 24 # # # # # #
0 18
10 30 4 27 15 19 14 43 24 18 # # # # #
0 5
10 30 4 27 15 19 14 43 24 18 5 # # # #
0 30
10 30 4 27 15 19 14 43 24 18 5 # # # #
0 10
10 30 4 27 15 19 14 43 24 18 5 # # # #
0 2
10 30 4 27 15 19 14 43 24 18 5 2 # # #
1 4
10 30 4 27 15 19 14 43 24 18 5 2 # # #
0 34
10 30 4 27 15 19 14 43 24 18 5 2 34 # #
0 7
10 30 4 27 15 19 14 43 24 18 5 2 34 7 #
0 14
10 30 4 27 15 19 14 43 24 18 5 2 34 7
```

```
1 10
10 # # # # # # # # # # # # # #
1 30
10 30 # # # # # # # # # # # # #
1 4
10 30 4 # # # # # # # # # # # #
0 27
10 30 4 27 # # # # # # # # # # #
0 15
10 30 4 27 15 # # # # # # # # # #
0 19
10 30 4 27 15 19 # # # # # # # # #
0 14
10 30 4 27 15 19 14 # # # # # # # #
1 43
10 30 4 27 15 19 14 43 # # # # # # #
0 24
10 30 4 27 15 19 14 43 24 # # # # # #
0 18
10 30 4 27 15 19 14 43 24 18 # # # # #
0 5
10 30 4 27 15 19 14 43 24 18 5 # # # #
0 30
10 30 4 27 15 19 14 43 24 18 5 # # # #
0 10
10 30 4 27 15 19 14 43 24 18 5 # # # #
0 2
10 30 4 27 15 19 14 43 24 18 5 2 # # #
1 4
10 30 4 27 15 19 14 43 24 18 5 2 # # #
0 34
10 30 4 27 15 19 14 43 24 18 5 2 34 # #
0 7
10 30 4 27 15 19 14 43 24 18 5 2 34 7 #
0 14
10 30 4 27 15 19 14 43 24 18 5 2 34 7
```

Исходный код

```
#include <iostream>
#include <sstream>
#include <string>
#include <vector>
#include <climits>

#include "lab4.h"

int const kTableSize = 14;
int const kResetStep = 5;

enum Algorithms {
    eFIFO,
    eAging
};

struct Row {
    unsigned int pageNumber;
    unsigned char read;
    unsigned char modified;
    unsigned char counter;
};

using InvTable = std::vector<Row>;

InvTable createTable() {
    InvTable table(kTableSize);
    for (auto &row : table) {
        row.pageNumber = -1;
    }

    return table;
}
```

```
    void updateTableStats(InvTable &table, Algorithms const
&algo) {
        if (algo != Algorithms::eAging) {
            return;
        }

        for (auto &row : table) {
            row.counter = row.read << (sizeof(row.read) * 8 - 1)
| row.counter >> 1;
            row.read = 0;
        }
    }

    void fillRow(Row *row, unsigned int const &op, unsigned int
const &pageNumber, unsigned int counter) {
        row->pageNumber = pageNumber;
        row->read = 1;
        row->modified = op;
        row->counter = counter;
    }

    void processTable(InvTable &table, unsigned int const &op,
unsigned int const &pageNumber, unsigned int counter,
        Algorithms const &algo) {

        if (counter % kResetStep == 0) {
            updateTableStats(table, algo);
        }

        if (algo == Algorithms::eAging) {
            counter = 0;
        }

        Row *emptyRow = nullptr;
        std::vector<Row *> rowsForReplacement;
        unsigned char minCounter = CHAR_MAX;

        for (auto &row : table) {
```

```
            // Check if the page is already present.
            if (row.pageNumber == pageNumber) {
                // If so, set read and modified ...
                row.read = 1;
                if (op == 1) {
                    row.modified = 1;
                }
                // and return.
                return;
            }

            // Search for empty row.
            if (emptyRow == nullptr && row.pageNumber == -1) {
                emptyRow = &row;
            }

            // Select potential rows for replacement.
            if (minCounter >= row.counter) {
                if (minCounter > row.counter) {
                    minCounter = row.counter;
                    rowsForReplacement.clear();
                }
                rowsForReplacement.emplace_back(&row);
            }
        }

        if (emptyRow != nullptr) {
            fillRow(emptyRow, op, pageNumber, counter);
        } else {
            int rowIndex = 0;
            if (rowsForReplacement.size() > 1) {
                rowIndex            =            uniform_rnd(0,
rowsForReplacement.size() - 1);
            }

            fillRow(rowsForReplacement[rowIndex],           op,
pageNumber, counter);
```

```
        }
    }

    void outputTable(InvTable const &table) {
        for (int i = 0; i < table.size(); ++i) {
            if (table[i].pageNumber == -1) {
                std::cout << "#";
            } else {
                std::cout << table[i].pageNumber;
            }

            if (i < table.size() - 1) {
                std::cout << " ";
            }
        }

        std::cout << "\n";
    }

    void process(Algorithms const &algo) {
        InvTable table = createTable();

        unsigned int counter = 0;

        std::string line;
        unsigned int op = 0;
        unsigned int pageNumber = 0;

        while (true) {
            std::getline(std::cin, line);
            if (line.empty()) {
                return;
            }

            std::istringstream istream{line};
            if (!(istream >> op) || !(istream >> pageNumber) ||
istream >> line) {
```

```
                continue;
            }

            processTable(table, op, pageNumber, counter, algo);
            outputTable(table);

            counter++;
        }
    }

    int main(int argc, char *argv[]) {
        std::string algo;

        switch (argc) {
        case 2:
            algo = argv[1];
            if (algo != "1" && algo != "2") {
                std::cout << "Invalid algorithm: " << algo <<
"\n";
                return 1;
            }
            break;
        case 1:
            std::cout << "Too few arguments"
                      << "\n";
            return 1;
            break;
        default:
            std::cout << "Too many arguments"
                      << "\n";
            return 1;
            break;
        }

        process(algo    ==    "1"    ?    Algorithms::eFIFO    :
Algorithms::eAging);
```

```
        return 0;
    }
```

Выводы

В ходе выполнения лабораторной работы были изучены принципы организации виртуальной памяти.