КАФЕДРА №

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ
ПРЕПОДАВАТЕЛЬ

должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №4

Управление памятью.

по курсу: Операционные системы

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ гр. №			
		подпись, дата	инициалы, фамилия

Санкт-Петербург 2024

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.

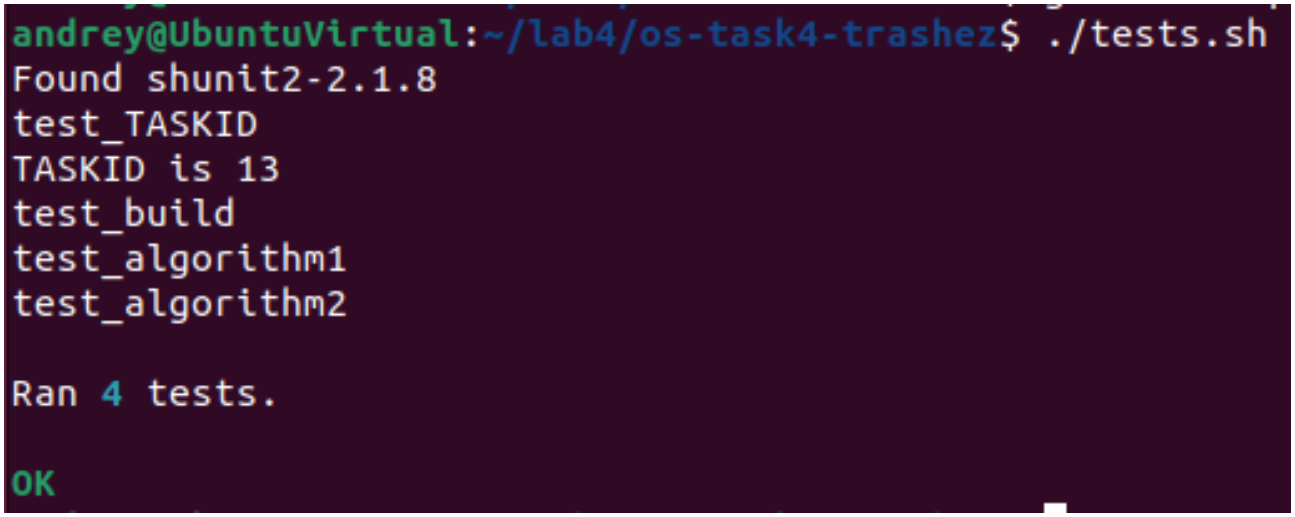
Номер варианта	**Количество страничных блоков**	**Алгоритм 1**	**Алгоритм 2**
13	12	FIFO	LRU

Описание используемых алгоритмов замещения страниц

В рамках лабораторной работы были использованы два алгоритма замещения страниц: FIFO (First-In, First-Out) и LRU (Least Recently Used). Описание каждого алгоритма представлено ниже:

1. FIFO (First-In, First-Out): Этот алгоритм основан на принципе "первым пришёл — первым ушёл". В нём страницы удаляются из физической памяти в порядке их поступления, без учёта частоты использования. Для реализации FIFO используется очередь, где хранятся индексы страниц по порядку их загрузки. При необходимости замещения страницы удаляется та, которая находится в начале очереди, и новая страница добавляется в конец.
2. LRU (Least Recently Used): Алгоритм LRU удаляет страницы, которые не использовались дольше всего. Для реализации этого метода используется время последнего использования каждой страницы, записываемое в специальное поле last_used структуры Page. При возникновении ошибки отсутствия страницы система ищет страницу с наименьшим значением last_used и заменяет её новой. Это позволяет поддерживать высокую производительность системы за счёт частого использования активно используемых данных.

Результат выполнения работы



Исходный код программы с комментариями

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

#include "lab4.h"  // Include the header for random number generation

struct Page {
    int vpn;     // Virtual page number
    bool r;      // Reference bit (used for administrative purposes here)
    bool m;      // Modified bit (not actively used in FIFO or LRU)
    int last_used; // Used for LRU to store the last use time

    Page() : vpn(-1), r(false), m(false), last_used(0) {}  // Default constructor initializes an empty
page
};

int main(int argc, char* argv[]) {
    if (argc != 2) {
        std::cerr << "Usage: " << argv[0] << " [algorithm number]" << std::endl;
        return 1;
    }

    int algorithm = std::stoi(argv[1]); // 1 for FIFO, 2 for LRU
    const int NUM_PAGES = 12;           // Number of physical pages available
    std::vector<Page> page_table(NUM_PAGES);
    std::deque<int> queue;              // Queue for FIFO
    int timer = 0;                      // Used as a timestamp for LRU
```

```
int operation_type, virtual_page;
while (std::cin >> operation_type >> virtual_page) {
    bool page_found = false;
    int empty_index = -1;

    for (int i = 0; i < NUM_PAGES; ++i) {
        if (page_table[i].vpn == virtual_page) {
            page_table[i].r = true;
            page_table[i].m = operation_type == 1 || page_table[i].m;
            page_table[i].last_used = timer;  // Update the last used time for LRU
            page_found = true;
            break;
        }
        if (page_table[i].vpn == -1 && empty_index == -1) {
            empty_index = i;       // Track the first empty spot
        }
    }

    if (!page_found) {
        int replace_index = empty_index != -1 ? empty_index : -1;

        if (replace_index == -1) {
            if (algorithm == 1) {  // FIFO
                if (!queue.empty()) {
                    replace_index = queue.front();
                    queue.pop_front();
                }
            }
            else if (algorithm == 2) {  // LRU
                int oldest_time = INT_MAX;
                for (int i = 0; i < NUM_PAGES; ++i) {
                    if (page_table[i].last_used < oldest_time) {
                        oldest_time = page_table[i].last_used;
                        replace_index = i;
                    }
                }
            }
        }

        if (replace_index != -1) {
            page_table[replace_index].vpn = virtual_page;
            page_table[replace_index].r = true;
            page_table[replace_index].m = (operation_type == 1);
            page_table[replace_index].last_used = timer;  // Update time for LRU
            if (algorithm == 1) {
                queue.push_back(replace_index);  // Add to queue for FIFO
            }
        }
    }

    std::ostringstream output;
    for (int i = 0; i < NUM_PAGES; ++i) {
```

```
            if (i > 0) output << " ";
            output << (page_table[i].vpn == -1 ? "#" : std::to_string(page_table[i].vpn));
        }
        std::cout << output.str() << std::endl;
        timer++;  // Increment timer for LRU
    }

    return 0;
}
```

Вывод

В ходе выполнения лабораторной работы было осуществлено знакомство с принципами организации виртуальной памяти и реализация фрагмента диспетчера памяти операционной системы. Работа с инвертированной таблицей страниц позволила практически изучить механизмы замещения страниц при ошибках отсутствия страниц. Это задание дало возможность лучше понять, как операционные системы управляют памятью и обрабатывают запросы на доступ к памяти.