Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут”
Кафедра АСОІУ

ЗВІТ

про виконання комп’ютерного практикуму № 3
з дисципліни
“Операційні системи”

Прийняв:
Проф. Сімоненко В. П.

Виконав:
студент 3-го курсу
гр. ІП-52 ФІОТ
Набоков Едуард
Максимович

Київ 2017

ЗМІСТ:

1. ОПИС АЛГОРИТМУ

Алгоритм – «Часи»

1. Початок
2. Помітити найстаршу сторінку в черги як голову черги.
3. ПОКИ працює алгоритм:

   3.1. Якщо сторінка, що запрошується знаходиться в оперативній пам’яті, то видати сторінку програмі та перейти до пункту 3. Інакше перейти до наступного пункту.

   3.2. Знайти запрошену сторінку в зовнішній пам’яті (Swap).

   3.3. ПОКИ голова черги має флаг R = 1:

      3.3.1. Присвоїти R = 0.

      3.3.2. Перемістити покажчик голови списку на наступний елемент черги.

   3.4. Замістити голову черги на запрошену сторінку з пам’яті та ви грузити голову черги в зовнішню пам’ять.

   3.5. Перемістити покажчик голови списку на наступний елемент черги.

4. Кінець

2. ОПИС ПРОГРАМИ

Клас Application моделює поведінку певного процесу. У момент створення посилає запит ОС на виділення блоку віртуальних адрес заданого розміру (розмір запиту вирівнюється по розміру сторінки). Після з цього блоку вибирає якийсь робочий набір (процес відбувається ітеративно по 1 сторінці). Він передається в ОС і мапиться на фізичну пам'ять. Після чого при кожній ітерації з ймовірністю 90% процес вибере дані, які знаходяться на сторінці з якої даний процес останній раз вибирав дані або з суміжних з нею (локальність звернень). При цьому з ймовірністю 50% процес буде виконувати читання з даного блока та з ймовірністю 50% буде модифікувати значення на вибрані сторінці.

3. СТРУКТУРИ ВІРТУАЛЬНОЇ ПАМЯТІ

Структура адресації віртуальної памяті зображена на рисунку 3.1.

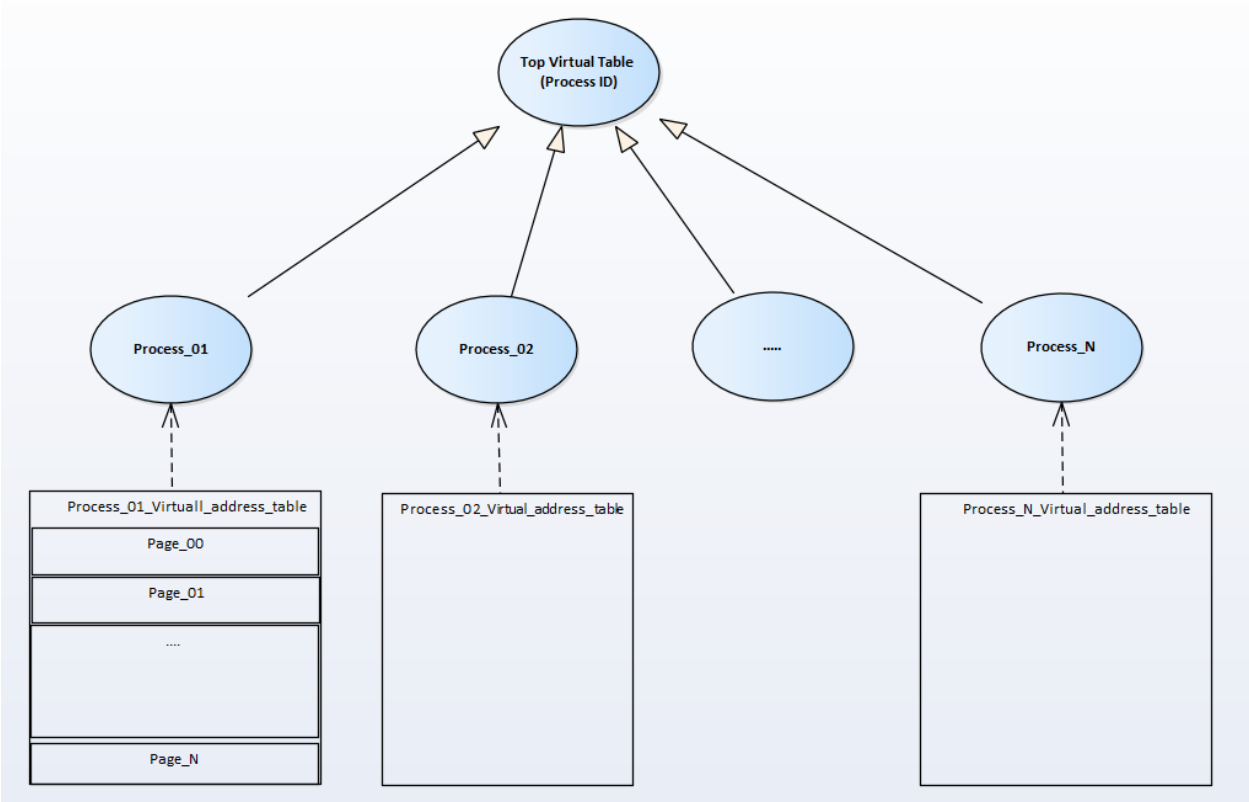


Рисунок 3.1. – Діаграма структури адресації віртуальної пам’яті

4. ПЕРЕВАГИ ТА НЕ ДОЛІКИ РОЗРОБЛЕНОГО АЛГОРИТМУ

Розроблений алгоритм дозволяє (швидкість залежить від обраного підходу заміщення сторінок зі зовнішньої пам’яті) виконувати заміщення сторінок віртуальної пам’яті, але його ефективність досить низка та мало чим відрізняється від алгоритму FIFO чи «Другий шанс». До переваг можна віднести простоту його реалізації та низькі затрати на службову інформацію та їх обробку (вони фактично відсутні).

5. ЛІСТИНГ ПРОГРАМИ

main.cpp

```
#include "Application.h"
#include "MemoryManager.h"
#include <cstdio>
#include <time.h>

const int N_PAGES = 32;
const int N_APPLICATIONS = 16;
const double LOCAL_READ_PROBABILITY = 0.9;
const double READ_PROBABILITY = 0.5;
char buf[80];

int main(void) {
	srand(time(0));
	MemoryManager mem_manager = MemoryManager(N_PAGES, page_size_hack);
	Application* applications[N_APPLICATIONS];
	for (int i = 0; i < N_APPLICATIONS; i++) {
		size_t mem_work_size = (rand() % 10 + 1) * page_size_hack;
		size_t id = mem_manager.start_application(mem_work_size);
		applications[i] = new Application(id, mem_work_size,
		 LOCAL_READ_PROBABILITY, READ_PROBABILITY);
	}
	while (true) {
		size_t app_id = rand() % N_APPLICATIONS;
		size_t page_id = applications[app_id]->emulate_work();
		mem_manager.get_by_virtual_address(app_id, page_id);
		printf("Continue (y/n)\n");
```

```
                scanf("%s", buf);
                if (buf[0] == 'n') {
                        break;
                }
        }
}
```

PageDescriptor.h

```
#pragma once
#include <stdlib.h>
struct PageDescriptor {
        bool loaded; // page loaded to memory or in swap
        size_t address; // page addresss in physical or virtual memory
        size_t last_use_session_id;
};
```

Application.h

```
#pragma once
#include <random>
#include <cstdlib>

#define page_size_hack 128

class Application {
private:
```

```
        const double _local_read_probability;
        const double _read_probability;
        size_t application_id;
        size_t work_memory_size; // in bytes
        size_t last_readed_block;

public:

        Application(size_t application_id,
                        size_t work_memory_size,
                        double local_read_probability,
                        double read_probability) :
_local_read_probability(local_read_probability),

_read_probability(read_probability) {
                this->application_id = application_id;
                this->work_memory_size = work_memory_size;
                last_readed_block = 0;
        }

        size_t emulate_work() {
                double random_number = rand() * 1. / RAND_MAX;

                if (last_readed_block == 0 || random_number > _local_read_probability) {
```

```
                last_readed_block = size_t(work_memory_size * random_number / page_size_hack);
        } else {
                if (last_readed_block > 1 && last_readed_block < (work_memory_size / page_size_hack - 1)) {
                        random_number -= 0.5;
                } else {
                        random_number = 0;
                }

                if (random_number < -0.17) {
                        last_readed_block -= 1;
                } else if (random_number > 0.17) {
                        last_readed_block += 1;
                }
        }

        return last_readed_block;
    }
};
```

MemoryManager.h

```
#pragma once
#include <vector>
#include <map>
#include <vector>
```

```
#include "PageDescriptor.h"

using std::vector;
using std::map;

class MemoryManager {
private:
	const static int _time_to_live_of_operation = 3;
	const size_t _physical_pages_count;
	const size_t _pages_sizes;

	size_t used_physical_pages;
	size_t used_virtual_pages;
	size_t next_application_unique_id;
	size_t session_id;
	size_t clock_arrow_index; // reliaze "clock method"

	vector<PageDescriptor*> pages_queue;
	map <size_t, vector<PageDescriptor> > applications_virtual_pages_table;

	void load_page_from_swap(PageDescriptor swap_page);
	bool check_page(size_t application_id, size_t page_index);
	void find_page_to_replace();
```

```
public:

        MemoryManager(size_t physical_pages_count, size_t pages_size);

        size_t start_application(size_t memory_size);

        void get_by_virtual_address(size_t application_id, size_t virtual_page_index);

};
```

MemoryManager.cpp

```
#include "MemoryManager.h"

#include "Logger.h"

#include <algorithm>

#include <sstream>

#include <iomanip>


using std::swap;

using std::stringstream;


/*  Create Memory manager

 *  Args:

 *     - physical_pages_count: count of physical memory

 *           - pages_size: size of each page

 */

MemoryManager::MemoryManager(size_t physical_pages_count, size_t pages_size) :
_physical_pages_count(physical_pages_count),


                                        _pages_sizes(pages_size)
```

```
{
        next_application_unique_id = 0;
        session_id = 0;
        clock_arrow_index = 0;
        clock_arrow_index = 0;
        used_physical_pages = 0;
        used_virtual_pages = 0;

        pages_queue = vector<PageDescriptor*>(physical_pages_count);
}


/*  Swap pages in swap and in physical memory
 *  Args:
 *                  - swap_page: page that going to be loaded
 */
void MemoryManager::load_page_from_swap(PageDescriptor swap_page) {
        find_page_to_replace();

        stringstream  message;
        message << std::dec << "Page in 0x" << std::hex << pages_queue[clock_arrow_index]->address
<<
                "(index - " << pages_queue[clock_arrow_index]->address / _pages_sizes << ") upload
from memory to swap";
        LoggerSingelton.info(message.str().c_str());
```

```
        stringstream  message1;
        message1 << std::dec << "Page in 0x" << std::hex << swap_page.address <<
                "(index - " << pages_queue[clock_arrow_index]->address / _pages_sizes << ") load from
swap to memory";
        LoggerSingelton.info(message1.str().c_str());

        swap(pages_queue[clock_arrow_index]->address, swap_page.address);
        pages_queue[clock_arrow_index]->loaded = false;
        swap_page.loaded = true;
        pages_queue[clock_arrow_index] = &swap_page;
        clock_arrow_index = (clock_arrow_index + 1) % (_physical_pages_count);
}

/*  Check is page loaded to memory
 *      Args:
 *              - application_id: application id
 *              - page_index: index of page for application
 */
bool MemoryManager::check_page(size_t application_id, size_t page_index) {
        return applications_virtual_pages_table[application_id][page_index].loaded;
}

/*  Find index of page to replace
 */
```

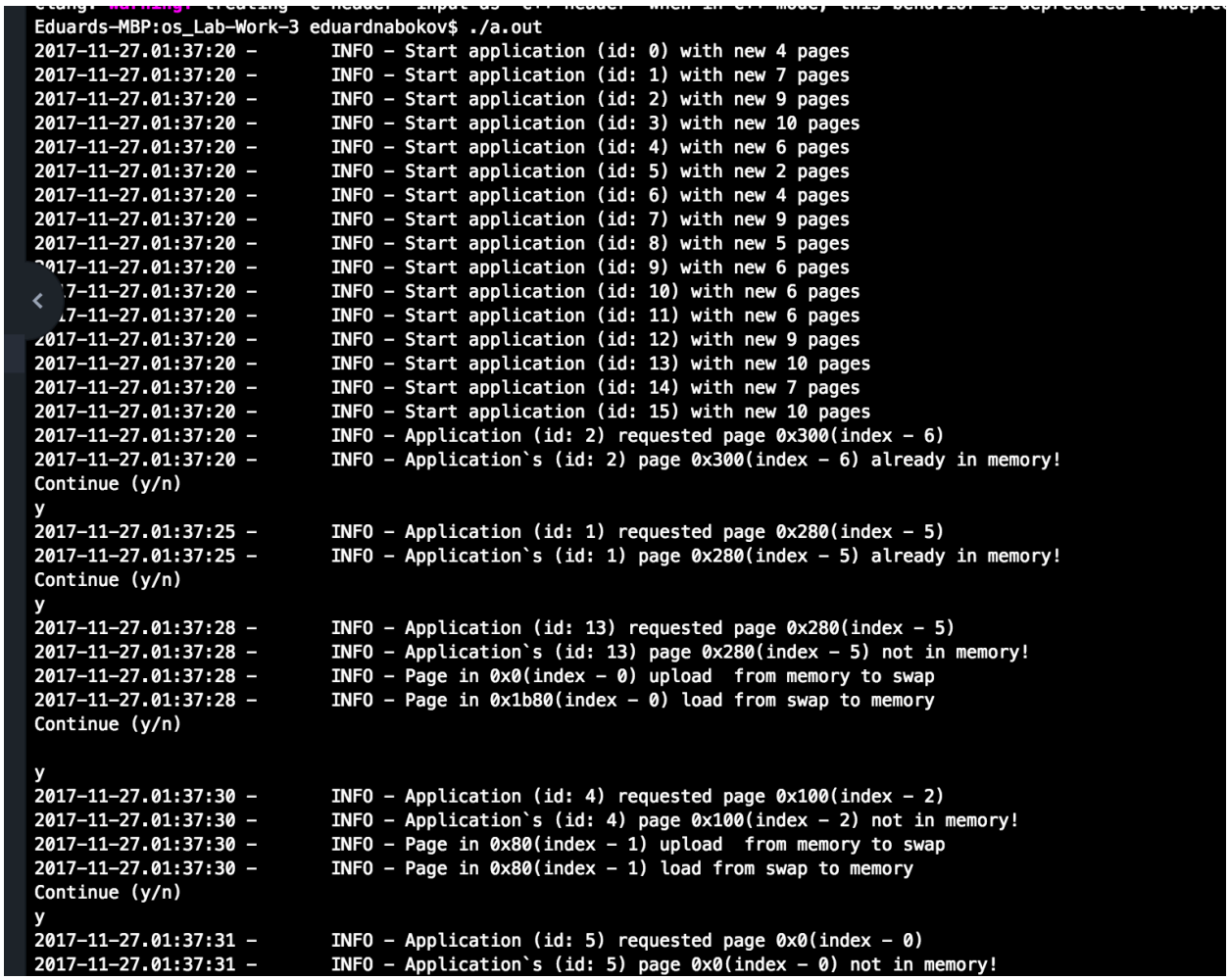
```
void MemoryManager::find_page_to_replace() {
	for (; clock_arrow_index < _physical_pages_count; clock_arrow_index++) {
		if (session_id - pages_queue[clock_arrow_index]->last_use_session_id > _time_to_live_of_operation) {
			pages_queue[clock_arrow_index]->last_use_session_id = session_id;
			return;
		}
	}

	for (clock_arrow_index = 0; clock_arrow_index < _physical_pages_count; clock_arrow_index++) {
		if (session_id - pages_queue[clock_arrow_index]->last_use_session_id > _time_to_live_of_operation) {
			pages_queue[clock_arrow_index]->last_use_session_id = session_id;
			return;
		}
	}
}

/*	Add one application for managing
 *	Args:
 *		- memory_size : size of memory (in bytes) that needed for application
 */
size_t MemoryManager::start_application(size_t memory_size) {
	int new_pages_count = memory_size / _pages_sizes;
```

```
        if (memory_size % _pages_sizes) {
                new_pages_count += 1;
        }

        stringstream  message;
        message << std::dec << "Start application (id: " << next_application_unique_id << ") with new " <<
                new_pages_count << " pages";
        LoggerSingelton.info(message.str().c_str());

        applications_virtual_pages_table[next_application_unique_id] = vector <PageDescriptor>(new_pages_count);
        while (new_pages_count) {
                new_pages_count--;
                if (used_physical_pages < _physical_pages_count) {
                        applications_virtual_pages_table[next_application_unique_id] [new_pages_count].address = (used_physical_pages * _pages_sizes);
                        applications_virtual_pages_table[next_application_unique_id] [new_pages_count].loaded = true;
                        pages_queue[used_physical_pages] = &applications_virtual_pages_table[next_application_unique_id][new_pages_count];
                        used_physical_pages++;
                } else {
                        applications_virtual_pages_table[next_application_unique_id] [new_pages_count].address = (used_virtual_pages * _pages_sizes);
                        applications_virtual_pages_table[next_application_unique_id] [new_pages_count].loaded = false;
```

```
                used_virtual_pages++;
            }
            applications_virtual_pages_table[next_application_unique_id]
[new_pages_count].last_use_session_id = session_id;
        }

        session_id++;
        return next_application_unique_id++;
}

/*      Read specific page
 *      Args:
 *              - application_id : application id that send requests
 *              - virtual_page_index : index of virual page that must be readed
 */
void MemoryManager::get_by_virtual_address(size_t application_id, size_t virtual_page_index) {
        stringstream  message;
        message << std::dec << "Application (id: " << application_id << ") requested page 0x" <<
            std::hex << (virtual_page_index * _pages_sizes) << "(index - " << virtual_page_index <<
")";
        LoggerSingelton.info(message.str().c_str());

        stringstream  message1;
        if (check_page(application_id, virtual_page_index)) {
            message1 << std::dec << "Application`s (id: " << application_id << ") page 0x" <<
```

```
                std::hex << (virtual_page_index * _pages_sizes) << "(index - " <<
virtual_page_index << ") already in memory!";

                LoggerSingelton.info(message1.str().c_str());

        } else {

                message1 << std::dec << "Application`s (id: " << application_id << ") page 0x" <<

                std::hex << (virtual_page_index * _pages_sizes) << "(index - " <<
virtual_page_index << ") not in memory!";

                LoggerSingelton.info(message1.str().c_str());

                load_page_from_swap(applications_virtual_pages_table[application_id]
[virtual_page_index]);

        }


        applications_virtual_pages_table[application_id][virtual_page_index].last_use_session_id =
session_id;

        session_id++;

}
```

6. ПРИКЛАД ВИКОНАННЯ ПРОГРАМИ



Робота програми