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: |  | Виконав: |
| Проф. Сімоненко В. П. |  | студент 3-го курсу  гр. ІП-51 ФІОТ  Зарічковий Олександр Анатолійович |

Київ – 2017

**ЗМІСТ:**

[1 ОПИС АЛГОРИТМУ 3](#_Toc494055106)

[2 Опис програми 4](#_Toc494055107)

[3 Структури Віртуальної памяті 5](#_Toc494055108)

[4 Переваги та не доліки розробленого алгоритму 6](#_Toc494055109)

[5 Лістинг програми 7](#_Toc494055110)

[6 Приклад виконання програми 13](#_Toc494055111)

# ОПИС АЛГОРИТМУ

Індивідуальне завдання: останні\_дві\_цифри\_заліковки mod 5 + 1 =

= 06 mod 5 + 1 = 2, отже обраний алгоритм – «Часи»

1. Початок
2. Помітити найстаршу сторінку в черги як голову черги.
3. ПОКИ працює алгоритм:
   1. Якщо сторінка, що запрошується знаходиться в оперативній пам’яті, то видати сторінку програмі та перейти до пункту 3. Інакше перейти до наступного пункту.
   2. Знайти запрошену сторінку в зовнішній пам’яті (Swap).
   3. ПОКИ голова черги має флаг R = 1:
      1. Присвоїти R = 0.
      2. Перемістити покажчик голови списку на наступний елемент черги.
   4. Замістити голову черги на запрошену сторінку з пам’яті та ви грузити голову черги в зовнішню пам’ять.
   5. Перемістити покажчик голови списку на наступний елемент черги.
4. Кінець

# 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.

# Структури Віртуальної памяті

Структура адресації віртуальної памяті зображена на рисунку 3.1.

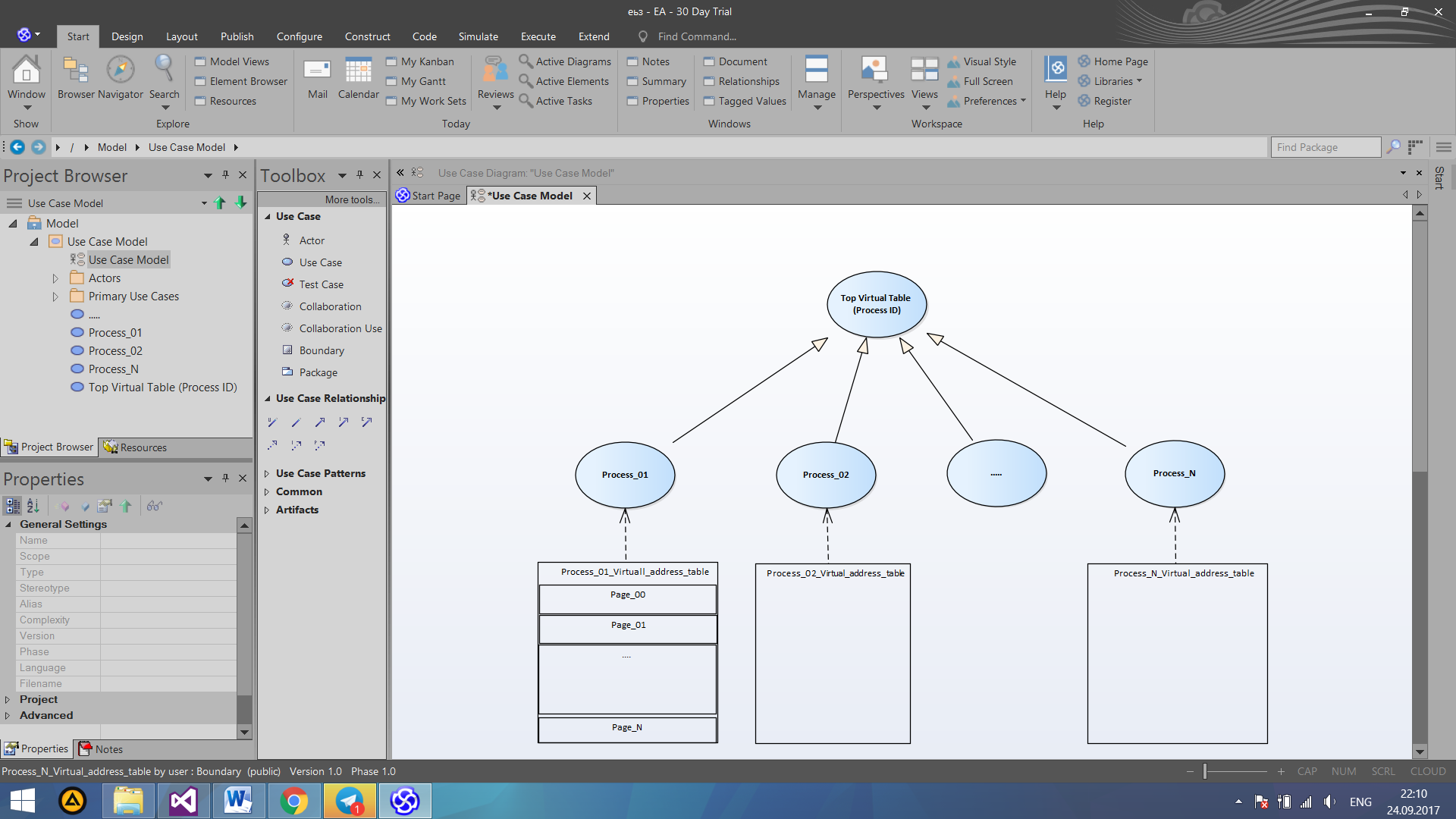


Рисунок 3.1. – Діаграма структури адресації віртуальної пам’яті

# 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).

# Лістинг програми

“Logger.h”

#pragma once

#include <stdio.h>

#include <time.h>

class Logger {

private:

FILE\* file;

void write(const char\* type, const char\* message) {

time\_t now = time(0);

struct tm tstruct;

char buf[80];

tstruct = \*localtime(&now);

strftime(buf, sizeof(buf), "%Y-%m-%d.%X", &tstruct);

fprintf(file, "%15s - %10s - %s\n", buf, type, message);

printf("%15s - %10s - %s\n", buf, type, message);

}

public:

Logger() {

file = fopen("log.txt", "w");

}

void info(const char\* message) {

write("INFO", message);

}

void debug(const char\* message) {

write("DEBUG", message);

}

~Logger() {

fclose(file);

}

} LoggerSingelton;

“PageDescriptor.h”

#pragma once

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++;

}

“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') {

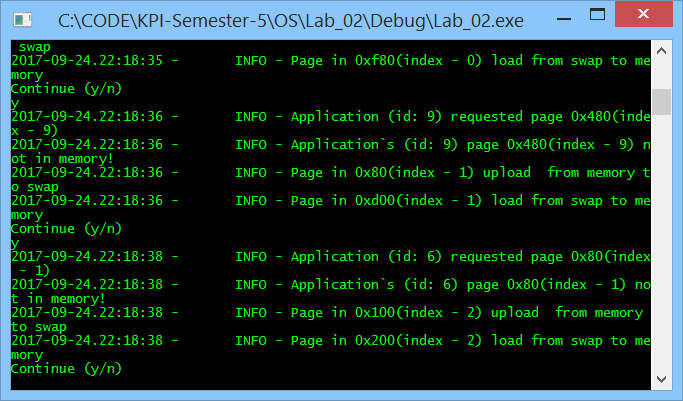
break;

}

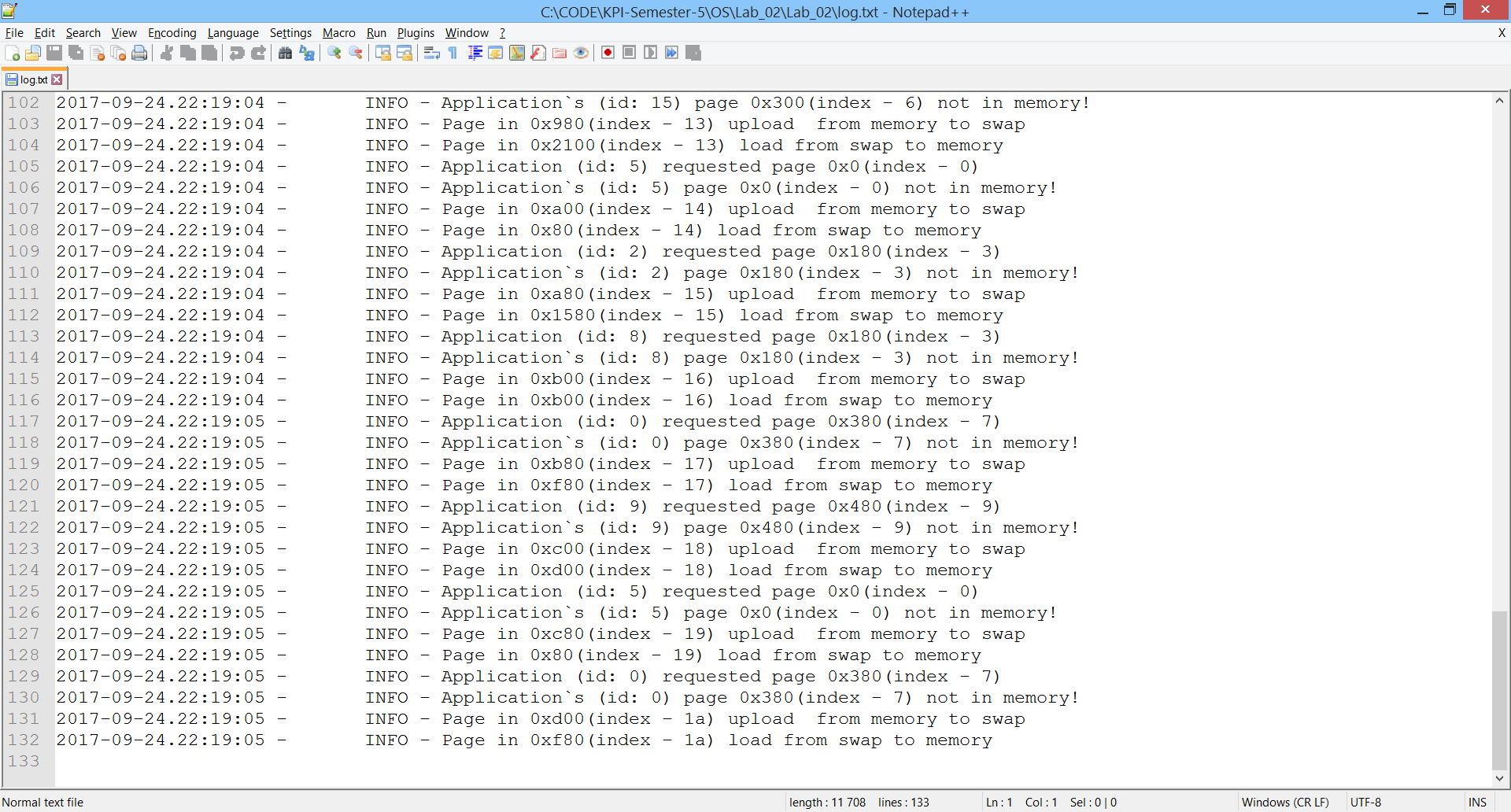
}

}

# Приклад виконання програми



Робота програми



Лог-файл створений програмою