МІНІСТЕРСТВО ОСВІТИ і НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет прикладної математики
Кафедра системного програмування і спеціалізованих комп'ютерних систем

Лабораторна робота №3

з дисципліни **"Введення до оперативних систем"**

Тема: **" Керування пам'яттю"**

Варіант 9

Виконав:
студент IV-го курсу ФПМ
групи КВ-82
Іваненко О.А.

Київ 2021

Метою лабораторної роботи є ознайомлення з існуючими способами структурування пам’яті, алгоритмами керування пам’яттю та перетворення віртуальних адрес у фізичні адреси.

Завдання на виконання роботи

1. Написати програму, що моделює процес управління пам’яттю (розподілу пам’яті для процесів), перетворення віртуальної адреси у фізичну, пошук у пам’яті за запитами процесів, вивільнення пам’яті) при заданому варіантом способі організації пам’яті (перелік варіантів представлений нижче). Вхідні дані – розмір пам’яті, що підлягає розподілу, розміри сторінок (розділів, сегментів тощо), розміри потрібної процесам пам’яті та ін. Задаються самостійно та у відповідності до завдання.

2. Продемонструвати роботу моделі з виконанням основних операцій з пам’яттю: надання пам’яті потрібного розміру за запитом процесу, перетворення віртуальної адреси у ―фізичну‖ при зверненні до комірки пам’яті, здійнення запису або читання, вивільнення пам’яті при завершенні процесу. Завдання операцій можна реалізувати за допомогою меню.

Варіант 9

9. *Сторінкова організація пам’яті*. Кількість процесів довільна. Віртуальний адресний простір поділяється *на розділи*, а розділи – *на сторінки*. Загальна кількість віртуальних сторінок процесів повинна перебільшувати загальну кількість фізичних сторінок. Додатково продемонструвати процес вивантаження-завантаження сторінок

Лістинг програми

```
package com;

import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.Scanner;

class Memory {
    PhysicalPage [] physicalPages = new PhysicalPage[7];

    void print(){
        System.out.println("----------------OP-MEMORY-------------");
        for(PhysicalPage page : physicalPages) {
            if(page != null){
                String time =
page.time.format(DateTimeFormatter.ofPattern("HH:mm:ss"));
                System.out.printf("| Process %3d| Section %3d|   Page %3d|
(%s)\n", page.processNum, page.secNum, page.virtualPage.number, time);
            }
            else{
                System.out.print("|            |            |
|\n");
            }
        }
        System.out.println("---------------------------------------");
    }

    Page getPage(Page page){
        for(int i = 0; i < physicalPages.length; i++){
            if(physicalPages[i] != null && physicalPages[i].virtualPage ==
page){
                return page;
            }
        }
        return null;
    }

    boolean full(){
        for(PhysicalPage page : physicalPages){
            if(page == null) return false;
        }
        return true;
    }

    void removePage(Page page){
        for(int i = 0; i < physicalPages.length; i++){
            if(physicalPages[i] != null && physicalPages[i].virtualPage ==
page){
                physicalPages[i] = null;
                page.phys_num = null;
                page.presence = 0;
                break;
            }
        }
    }
```

```
    void removeOldest(){
        int oldesIndex = 0;
        LocalDateTime time = physicalPages[0].time;
        for(int i = 0; i < physicalPages.length; i++){
            if(time.isAfter(physicalPages[i].time)){
                oldesIndex = i;
                time = physicalPages[i].time;
            }
        }
        physicalPages[oldesIndex].virtualPage.presence = 0;
        physicalPages[oldesIndex] = null;
    }

    void removeProcess(int index){
        for(int i = 0; i < physicalPages.length; i++){
            if(physicalPages[i] != null && physicalPages[i].processNum ==
index){
                physicalPages[i].virtualPage.phys_num = null;
                physicalPages[i].virtualPage.presence = 0;
                physicalPages[i] = null;
            }
        }

    }

    void addPage(int process, int section, Page page){
        /*try {
            TimeUnit.SECONDS.sleep(1);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }*/
        if(full()){
            removeOldest();
        }
        for(int i = 0; i < physicalPages.length; i++){
            if(physicalPages[i] == null){
                physicalPages[i] = new PhysicalPage( process, section, page);
                page.phys_num = i;
                page.presence = 1;
                break;
            }
        }

    }
}

class PhysicalPage {
    int processNum;
    int secNum;
    Page virtualPage;
    LocalDateTime time;

    PhysicalPage(int processNum, int secNum, Page virtualPage) {
        this.processNum = processNum;
        this.secNum = secNum;
        this.virtualPage = virtualPage;
        time = LocalDateTime.now();
```

```
    }
}

class Process {
    final int number;
    final Section[] sections;

    Process(int number, Section[] sections) {
        this.number = number;
        this.sections = sections;
    }

    void print() {
        System.out.println(" PROCESS #" + number);
        for(Section section : sections){
            section.print();
        }
        System.out.println('\n');
    }
}

class Section {
    final int number;
    final Page[] pages;

    Section(int number, Page[] pages) {
        this.number = number;
        this.pages = pages;
    }

    void print() {
        System.out.println("\t Section #" + number);
        for(Page page : pages){
            page.print();
        }
    }
}

class Page {
    final int number;
    int presence;
    Integer phys_num = null;

    Page(int number, int presence) {
        this.number = number;
        this.presence = presence;
    }

    void setPhysNum(int num){
        phys_num = num;
    }

    void print(){
        System.out.printf("\t\t Page #%d, %d, Phys: %d\n", number, presence,
phys_num);
    }
}
```

```
class Processes {
    Process process1 = new Process(1,
            new Section[] {
                    new Section(0, new Page[] {
                            new Page(0, 0),
                            new Page(1, 0),
                            new Page(2, 0)}),
                    new Section(1, new Page[] {
                            new Page(0, 0),
                            new Page(1, 0),
                            new Page(2, 0)}),
                    new Section(2, new Page[] {
                            new Page(0, 0),
                            new Page(1, 0),
                            new Page(2, 0)})
            });
    Process process2 = new Process(2,
            new Section[] {
                    new Section(0, new Page[] {
                            new Page(0, 0)}),
                    new Section(1, new Page[] {
                            new Page(0, 0),
                            new Page(1, 0)})
            });

    Process getProcess(int num){
        return switch (num) {
            case 1 -> process1;
            case 2 -> process2;
            // . . .
            default -> throw new IllegalArgumentException();
        };
    }

    void printAll(){
        process1.print();
        process2.print();
    }
}


public class Main {

    static void menu(){
        System.out.println("     MENU:      ");
        System.out.println(" 1. Print operating memory");
        System.out.println(" 2. Print process");
        System.out.println(" 3. Load page of process");
        System.out.println(" 4. Unload page of process");
        System.out.println(" 5. Remove process");
        System.out.println(" 6. Get physical addr");
        System.out.println(" 0. Exit");
    }

    static void processPrint(Processes processes){
        Scanner sc = new Scanner(System.in);
```

```
        System.out.print("Input number of process (or input 0 for print all
processes): ");
        int n = sc.nextInt();
        if(n == 0) processes.printAll();
        else processes.getProcess(n).print();
    }

    static void clearProcess(Processes processes, Memory memory){
        Scanner sc = new Scanner(System.in);
        System.out.print("Input number of process: ");
        int n = sc.nextInt();
        Process currentProc = processes.getProcess(n);
        memory.removeProcess(n);
        System.out.print("Process unloaded.\n");
    }

    static void processUnload(Processes processes, Memory memory){
        Scanner sc = new Scanner(System.in);
        System.out.print("Input number of process: ");
        int n = sc.nextInt();
        Process currentProc = processes.getProcess(n);

        System.out.print("Input virtual addr(XXXYYYCCC): ");
        String virtualAddr = sc.nextLine();
        virtualAddr = sc.nextLine();
        int sectionNum = Integer.parseInt(virtualAddr.substring(0,3), 2);
        int pageNum = Integer.parseInt(virtualAddr.substring(3,6), 2);
        Page currentPage = currentProc.sections[sectionNum].pages[pageNum];
        System.out.println(currentPage.number);
        memory.removePage(currentPage);
        System.out.print("Page unloaded.\n");

    }


    static void processLoad(Processes processes, Memory memory){
        Scanner sc = new Scanner(System.in);
        System.out.print("Input number of process: ");
        int n = sc.nextInt();
        Process currentProc = processes.getProcess(n);

        System.out.print("Input virtual addr(XXXYYYCCC): ");
        String virtualAddr = sc.nextLine();
        virtualAddr = sc.nextLine();
        int sectionNum = Integer.parseInt(virtualAddr.substring(0,3), 2);
        int pageNum = Integer.parseInt(virtualAddr.substring(3,6), 2);
        Page currentPage = currentProc.sections[sectionNum].pages[pageNum];
        memory.addPage(n, sectionNum, currentPage);
        System.out.printf("Page loaded. Physical addr : [0x%s|%s]\n",
Integer.toBinaryString(currentPage.phys_num), virtualAddr.substring(6));

    }

    static void getPhysicalAddr(Processes processes, Memory memory){
        Scanner sc = new Scanner(System.in);
        System.out.print("Input number of process: ");
        int n = sc.nextInt();
```

```
        Process currentProc = processes.getProcess(n);

        System.out.print("Input virtual addr(XXXYYYCCC): ");
        String virtualAddr = sc.nextLine();
        virtualAddr = sc.nextLine();
        int sectionNum = Integer.parseInt(virtualAddr.substring(0,3), 2);
        int pageNum = Integer.parseInt(virtualAddr.substring(3,6), 2);
        Page currentPage = currentProc.sections[sectionNum].pages[pageNum];
        if(memory.getPage(currentPage) == null){
            memory.addPage(n, sectionNum, currentPage);
            System.out.printf("Page loaded just now. Physical addr :
[0x%s|%s]\n", Integer.toBinaryString(currentPage.phys_num),
virtualAddr.substring(6));
        }
        else{
            System.out.printf("Page was loaded. Physical addr : [0x%s|%s]\n",
Integer.toBinaryString(currentPage.phys_num), virtualAddr.substring(6));
        }

    }


    public static void main(String[] args) {
        Memory memory = new Memory();
        Processes processes = new Processes();

        /*memory.addPage(5,5,new Page(5,5));
        memory.addPage(5,5,new Page(5,5));
        memory.addPage(5,5,new Page(5,5));
        memory.addPage(5,5,new Page(5,5));*/

        //main loop
        int temp = 1;

        while(temp != 0){

            Scanner sc = new Scanner(System.in);
            menu();
            temp = sc.nextInt();
            switch (temp){
                case 1: memory.print(); break;
                case 2: processPrint(processes); break;
                case 3: processLoad(processes, memory); break;
                case 4: processUnload(processes, memory); break;
                case 5: clearProcess(processes, memory); break;
                case 6: getPhysicalAddr(processes, memory); break;
                default: break;
            }
        }
    }
}
```

Тестування програми

Пункт меню:

```
     MENU:
  1. Print operating memory
  2. Print process
  3. Load page of process
  4. Unload page of process
  5. Remove process
  6. Get physical addr
  0. Exit
```

Початковий стан ОП та віртуальних таблиць процесів

```
----------------OP-MEMORY-------------
|            |            |          |
|            |            |          |
|            |            |          |
|            |            |          |
|            |            |          |
|            |            |          |
|            |            |          |
-------------------------------------
```

```
PROCESS #1
    Section #0
        Page #0, 0, Phys: null
        Page #1, 0, Phys: null
        Page #2, 0, Phys: null
    Section #1
        Page #0, 0, Phys: null
        Page #1, 0, Phys: null
        Page #2, 0, Phys: null
    Section #2
        Page #0, 0, Phys: null
        Page #1, 0, Phys: null
        Page #2, 0, Phys: null
```

```
PROCESS #2
    Section #0
        Page #0, 0, Phys: null
    Section #1
        Page #0, 0, Phys: null
        Page #1, 0, Phys: null
```

Процес завантаження певної сторінки відповідного процесу до ОП

```
Input number of process: 1I
Input virtual addr(XXXYYYCCC): 001001011
Page loaded. Physical addr : [0x0|011]
```

Як ми можем бачити, ми отримали фізичну адресу та повідомлення про те, що сторінка завантажена до ОП. Щодо віртуальної адреси для спрощення візуалізації було обрано бінарне представлення – перші три біти відповідають за номер розділу, наступні три біти за номер сторінки в цьому розділі, останні три біт – зсув.

Подивимось вміст віртуальних сторінок та ОП:

```
----------------OP-MEMORY--------------
| Process   1| Section   1|   Page   1|  (20:30:42)
|            |            |           |
|            |            |           |
|            |            |           |
|            |            |           |
|            |            |           |
|            |            |           |
--------------------------------------
```

```
PROCESS #1
    Section #0
        Page #0, 0, Phys: null
        Page #1, 0, Phys: null
        Page #2, 0, Phys: null
    Section #1
        Page #0, 0, Phys: null
        Page #1, 1, Phys: 0
        Page #2, 0, Phys: null
    Section #2
        Page #0, 0, Phys: null
        Page #1, 0, Phys: null
        Page #2, 0, Phys: null
```

Як ми бачимо, в ОП завантажено сторінку 1, з розділу 1 з процесу 1. Щодо віртуальних сторінок, ми можемо побачити що відповідна сторінка отримала номер в оперативній пам'яті (який використовується в отримані фізичної адреси)

```
PROCESS #2
    Section #0
        Page #0, 0, Phys: null
    Section #1
        Page #0, 0, Phys: null
        Page #1, 0, Phys: null
```

Спробуємо отримати фізичну адресу вже завантаженої сторінки:

```
Input virtual addr(XXXYYYCCC): 001001011
Page was loaded. Physical addr : [0x0|011]
```

Як ми бачимо, повідомлення про те, що сторінка вже завантажена, та її фізична адреса.

Тепер спробуємо отримати фізичну адресу сторінки, якої ще немає в оперативній пам'яті:

```
Input number of process: 2
Input virtual addr(XXXYYYCCC): 000000001
Page loaded just now. Physical addr : [0x1|001]
```

Ми бачимо повідомлення про те, що сторінка була тільки завантажена, тобто її не було в ОП, і вона завантажилась. Щодо вмісту ОП та віртуальних таблиць:

```
----------------OP-MEMORY--------------
| Process   1| Section   1|   Page   1|  (20:37:29)
| Process   2| Section   0|   Page   0|  (20:37:46)
|            |            |           |
|            |            |           |
|            |            |           |
|            |            |           |
|            |            |           |
--------------------------------------
```

```
PROCESS #1
    Section #0
        Page #0, 0, Phys: null
        Page #1, 0, Phys: null
        Page #2, 0, Phys: null
    Section #1
        Page #0, 0, Phys: null
        Page #1, 1, Phys: 0
        Page #2, 0, Phys: null
    Section #2
        Page #0, 0, Phys: null
        Page #1, 0, Phys: null
        Page #2, 0, Phys: null
```

Бачимо що сторінка завантажилась автоматично при спробі отримання її адреси. Відповідний процес та біт присутності також змінились.

```
PROCESS #2
    Section #0
        Page #0, 1, Phys: 1
    Section #1
        Page #0, 0, Phys: null
        Page #1, 0, Phys: null
```

Тепер спробуємо вигрузити сторінку з ОП та подивимось на вміст ОП та віртуальних таблиць.

```
Input virtual addr(XXXYYYCCC): 001001011
1
Page unloaded.
```

```
----------------OP-MEMORY--------------
|            |            |           |
| Process   2| Section   0|   Page   0|  (20:37:46)
|            |            |           |
|            |            |           |
|            |            |           |
|            |            |           |
|            |            |           |
-------------------------------------
```

```
PROCESS #1
    Section #0
        Page #0, 0, Phys: null
        Page #1, 0, Phys: null
        Page #2, 0, Phys: null
    Section #1
        Page #0, 0, Phys: null
        Page #1, 0, Phys: null
        Page #2, 0, Phys: null
    Section #2
        Page #0, 0, Phys: null
        Page #1, 0, Phys: null
        Page #2, 0, Phys: null


PROCESS #2
    Section #0
        Page #0, 1, Phys: 1
    Section #1
        Page #0, 0, Phys: null
        Page #1, 0, Phys: null
```

Також необхідно продемонструвати випадок, коли вся оперативна пам'ять зайнята. Для прикладу я загрузив оперативну пам'ять певним процесом 5.

```
----------------OP-MEMORY--------------
| Process   5| Section   1|   Page   5|  (20:44:58)
| Process   5| Section   2|   Page   5|  (20:44:59)
| Process   5| Section   3|   Page   5|  (20:45:00)
| Process   5| Section   4|   Page   5|  (20:45:01)
| Process   5| Section   5|   Page   5|  (20:45:02)
| Process   5| Section   6|   Page   5|  (20:45:03)
| Process   5| Section   7|   Page   5|  (20:45:04)
-------------------------------------
```

Тепер спробуємо додати сторінку до ОП:

```
Input number of process: 1
Input virtual addr(XXXYYYCCC): 010010111
Page loaded. Physical addr : [0x0|111]
```

Отримали повідомлення про завантаження сторінки. Вміст ОП та віртуальних таблиць:

```
----------------OP-MEMORY--------------
| Process   1| Section   2|   Page   2|  (20:46:02)
| Process   5| Section   2|   Page   5|  (20:44:59)
| Process   5| Section   3|   Page   5|  (20:45:00)
| Process   5| Section   4|   Page   5|  (20:45:01)
| Process   5| Section   5|   Page   5|  (20:45:02)
| Process   5| Section   6|   Page   5|  (20:45:03)
| Process   5| Section   7|   Page   5|  (20:45:04)
-------------------------------------
```

Я вирішив видаляти сторінку по стратегії FIFO, тобто сторінка, яка була завантажена першою буде першою вивільнена.

```
PROCESS #1
    Section #0
        Page #0, 0, Phys: null
        Page #1, 0, Phys: null
        Page #2, 0, Phys: null
    Section #1
        Page #0, 0, Phys: null
        Page #1, 0, Phys: null
        Page #2, 0, Phys: null
    Section #2
        Page #0, 0, Phys: null
        Page #1, 0, Phys: null
        Page #2, 1, Phys: 0


PROCESS #2
    Section #0
        Page #0, 0, Phys: null
    Section #1
        Page #0, 0, Phys: null
        Page #1, 0, Phys: null
```

Також реалізовано вивільненя ОП при завершені процесу. Якщо маємо наприкалад відповідний вміст ОП та віртуальних сторінок:

```
----------------OP-MEMORY-------------
| Process   1| Section   2|   Page   2|  (21:03:40)
| Process   1| Section   1|   Page   1|  (21:03:44)
| Process   1| Section   0|   Page   0|  (21:03:52)
| Process   5| Section   4|   Page   5|  (21:03:33)
| Process   5| Section   5|   Page   5|  (21:03:33)
| Process   5| Section   6|   Page   5|  (21:03:33)
| Process   5| Section   7|   Page   5|  (21:03:33)
-------------------------------------
```

```
PROCESS #1
    Section #0
        Page #0, 1, Phys: 2
        Page #1, 0, Phys: null
        Page #2, 0, Phys: null
    Section #1
        Page #0, 0, Phys: null
        Page #1, 1, Phys: 1
        Page #2, 0, Phys: null
    Section #2
        Page #0, 0, Phys: null
        Page #1, 0, Phys: null
        Page #2, 1, Phys: 0


PROCESS #2
    Section #0
        Page #0, 0, Phys: null
    Section #1
        Page #0, 0, Phys: null
        Page #1, 0, Phys: null
```

При вивільнені всього процесу, всі сторінки цього процесу буде вивантажено з ОП:

```
----------------OP-MEMORY-------------
|            |            |           |
|            |            |           |
|            |            |           |
| Process   5| Section   4|   Page   5|  (21:03:33)
| Process   5| Section   5|   Page   5|  (21:03:33)
| Process   5| Section   6|   Page   5|  (21:03:33)
| Process   5| Section   7|   Page   5|  (21:03:33)
-------------------------------------
```

Висновок

Отже, в цій лабораторній було досліджено сторінковий спосіб розподілу пам’яті. Структура віртуальних таблиць – процеси складаються з розділів, розділи с сторінок. На початковий момент часу ОП пуста, коли ми починаємо завантажувати сторінки в ОП відповідні сторінки в відповідних розділах починають змінюватись. Наприклад при завантажені першої сторінки першого розділу, вона додається до ОП, та в неї змінюється ознака присутності, та з’являється номер фізичної сторінки.

В програмі реалізовані основні функції роботи з сторінковою організацією пам’яті. При спробі отримати віртуальну адресу, якої ще немає в ОП, вона автоматично додається до таблиці фізичних сторінок. При вивантажені сторінки у віртуальної сторінки біт присутності змінюється на 0, а порядковий номер фізичної сторінки пропадає. Якщо ОП переповнена, то вивільненя сторінок відбувається по стратегії FIFO, тобто сторінка яка була першою завантажена – перша вивільняється, для цього в програмі реалізований лічильник часу для кожної з фізичних сторінок (що видно у прикладах в таблиці ОП). Щодо перетворення віртуальних адрес у фізичні, для спрощення використовується три біти для номеру розділу, три біти для номеру сторінки, та три біти на зсув.

Сторінкова організація пам’яті вирішує деякі проблеми з фрагментацію: зовнішня фрагментація – через застосування блоків фіксованого розміру, всі запити фіксовані, через що не буде залишатись некратних зон, внутрішня фрагментація – через користування досить малими блоками не буде з’являтись великих «дірок» в пам’яті. Також плюсом є віртуальний адресний простір. Оскільки простір для маніпуляцій обмежений,

віртуальні сторінки допомагають використовувати одні і ті ж фізичні сторінки під різні процеси.

Щодо мінусів використання сторінкової організації – інколи процесам потрібно використовувати розміри, некратні розміру сторінки, що може бути проблемою, також є досить великі апаратні затрати при звернені до пам’яті (спочатку до таблиць, потім до пам’яті). В кінці кінців, потрібні великі обсяги пам’яті для зберігання таблиць сторінок. Попри всі мінуси, сторінкова організація пам’яті залишається актуальною та використовується.