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 №1

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

Тема: **"Планування процесів"**

Варіант 9

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

Київ 2021

Завдання

Написати програму, що моделює послідовність обслуговування процесів, які знаходяться у черзі готових до виконання, за певним алгоритмом планування (перелік варіантів представлений нижче).

Вхідні дані (студент задає самостійно з урахуванням особливостей заданого алгоритму): послідовність процесів, які надходять до черги готових процесів; час надходження процесів до черги; очікуваний час виконання; пріоритет процесів та/або інщі додаткові дані, що необхідні для реалізації алгоритму планування згідно завдання.

Мета: ознайомлення з основними алгоритмами планування процесів в операційних системах і моделювання роботи планувальника (побудова програмної моделі) за певним, конкретно заданим алгоритмом

Варіант 9

1) черга інтерактивних процесів – алгоритм **RR**

2) черга фонових процесів – алгоритм **SRTF**

Час розподіляється між чергою фонових і чергою інтерактивних процесів. Наприклад, для інтерактивних процесів – 80 % і для фонових – 20 % часу. Якщо при досягненні ліміту часу виконується фоновий процес, то він переривається. Його продовження відбуватиметься при наступному переході на обслуговування черги фонових процесів з урахуванням особливостей алгоритму **SRTF**.

Лістинг програми

```
package com;

class Process{
    static int globalCount = 1;
    int arrive;
    int exec;
    int waiting = 0;
    int plus = 0;

    public Process(int arrive, int exec) {
        this.arrive = arrive;
        this.exec = exec;
    }

    static Process [] createProcesses(int n){
        Process [] result = new Process[n];
        result[0] = (new Process(0,  (int)(Math.random() * 10) + 1));
        for(int i = 1; i < n; i++){
            result[i] = (new Process((int)(Math.random() * 2) + i,
(int)(Math.random() * 10) + 1));
        }
        return result;
    }

    static void runSRTF(Process [] lst, int time){
        for(int i = 1; i <= time; i++){
            if(allZero(lst)) break;
            int idx = minExecIndex(lst);
            updateWaiting(lst, idx);
            lst = minusArriveTime(lst);
            if(lst[idx].exec != 0) lst[idx].exec--;
            System.out.printf("%3d (%3d): \n", i, globalCount++);
            printExec(lst);
        }
    }

    private static void updateWaiting(Process[] lst, int except) {
        for(int i = 0; i < lst.length; i++){
            if(i == except) continue;
            if(lst[i].arrive == 0 && lst[i].exec != 0) lst[i].waiting++;
        }
    }

    private static boolean allZero(Process[] lst) {
        for (Process process : lst) {
            if (process.exec != 0) return false;
        }
        return true;
    }

    private static Process[] minusArriveTime(Process[] lst){
        for (Process process : lst) {
            if (process.arrive != 0) process.arrive--;
        }
        return lst;
```

```
}

static private int minExecIndex(Process[] lst){
    int min = 100000000;
    int idx = 0;
    for(int i = 0; i < lst.length; i++){
        if(lst[i].arrive == 0 && lst[i].exec < min && lst[i].exec > 0){
            min = lst[i].exec;
            idx = i;
        }
    }
    return idx;
}

static void printExec(Process[] lst){
    System.out.print("Exec :    ");
    for(Process process : lst){
        System.out.printf("%3d ", process.exec);
    }
    System.out.print("|\n");
    System.out.print("Arrrive : ");
    for(Process process : lst){
        System.out.printf("%3d ", process.arrive);
    }
    System.out.print("|\n");
    System.out.print("Waiting : ");
    for(Process process : lst){
        System.out.printf("%3d ", process.waiting);
    }
    System.out.print("|\n----------");
    for(int i = 0; i < lst.length; i++)
        System.out.print("----");
    System.out.print("+\n");
}

static void runRR(Process[] lst, int quantum, int time){
    int currIdx = 0;
    int counter = 0;
    for(int i = 1; i <= time; i++) {
        if (allZero(lst)) break;
        int flag = 0;
        if(counter == quantum) {
            currIdx = getNextIdx(lst, currIdx);
            counter = 0;
        }
        while(lst[currIdx].exec == 0 && flag < lst.length) {
            currIdx = getNextIdx(lst, currIdx);
            counter = 0;
            flag++;
        }
        if(lst[currIdx].exec != 0 && counter < quantum) {
            lst[currIdx].exec--;
            counter++;
        }
        minusArriveTime(lst);
        updateWaiting(lst, currIdx);
        System.out.printf("%3d (%3d): \n", i, globalCount++);
```

```
            printExec(lst);
        }
    }

    static int getNextIdx(Process[] lst, int currIdx){
        if(currIdx == -1) return 0;
        if(currIdx >= lst.length - 1) return 0;
        if(lst[currIdx++].arrive != 0) return 0;
        return currIdx++;
    }

    static void runAll(int processorTime, Process[] srtf, Process [] rr, int
quantum){

        while (!allZero(rr) || !allZero(srtf)) {
            if (!allZero(rr)) {
                System.out.println("-------------------------------------
INTERACTIVE -------------------------------------");
                Process.runRR(rr, quantum, (int) (processorTime * 0.8));
                System.out.println("---------------------------------------
BACK ---------------------------------------");
            }
            Process.runSRTF(srtf, (int) (processorTime * 0.2));
        }
    }

    static double averageWait(Process[] lst){
        int sum = 0;
        for(Process process : lst){
            sum += process.waiting;
        }
        return (double) sum / lst.length;
    }
}




public class Main {

    public static void main(String[] args) {
        Process [] SRTF = Process.createProcesses(3);
        Process [] RR = Process.createProcesses(4);
        System.out.println("SRTF processes:");
        Process.printExec(SRTF);
        System.out.println("RR processes:");
        Process.printExec(RR);
        Process.runAll(20, SRTF, RR, 2);

        System.out.printf("Average wait time for SRTF: %5.5f\n",
Process.averageWait(SRTF));
        System.out.printf("Average wait time for RR: %5.5f \n",
Process.averageWait(RR));

    }
}
```

Результат програми

```
SRTF processes:
    Exec :      9   1   2 |
    Arrrive :   0   2   3 |
    Waiting :   0   0   0 |
    ----------------------+
    RR processes:
    Exec :      4  10   7   7 |
    Arrrive :   0   1   2   4 |
    Waiting :   0   0   0   0 |
    --------------------------+
    ------------------------------------ INTERACTIVE ----------------------
    1 (  1):
    Exec :      3  10   7   7 |
    Arrrive :   0   0   1   3 |
    Waiting :   0   1   0   0 |
    --------------------------+
    2 (  2):
    Exec :      2  10   7   7 |
    Arrrive :   0   0   0   2 |
    Waiting :   0   2   1   0 |
    --------------------------+
    3 (  3):
    Exec :      2   9   7   7 |
    Arrrive :   0   0   0   1 |
    Waiting :   1   2   2   0 |
    --------------------------+
    4 (  4):
    Exec :      2   8   7   7 |
    Arrrive :   0   0   0   0 |
    Waiting :   2   2   3   1 |
    --------------------------+
    5 (  5):
    Exec :      2   8   6   7 |
    Arrrive :   0   0   0   0 |
    Waiting :   3   3   3   2 |
    --------------------------+
    6 (  6):
    Exec :      2   8   5   7 |
    Arrrive :   0   0   0   0 |
    Waiting :   4   4   3   3 |
    --------------------------+
    7 (  7):
    Exec :      2   8   5   6 |
    Arrrive :   0   0   0   0 |
    Waiting :   5   5   4   3 |
    --------------------------+
    8 (  8):
    Exec :      2   8   5   5 |
    Arrrive :   0   0   0   0 |
    Waiting :   6   6   5   3 |
```

```
--------------------------+
9 (  9):
Exec :      1   8   5   5 |
Arrrive :   0   0   0   0 |
Waiting :   6   7   6   4 |
--------------------------+
10 ( 10):
Exec :      0   8   5   5 |
Arrrive :   0   0   0   0 |
Waiting :   6   8   7   5 |
--------------------------+
11 ( 11):
Exec :      0   7   5   5 |
Arrrive :   0   0   0   0 |
Waiting :   6   8   8   6 |
--------------------------+
12 ( 12):
Exec :      0   6   5   5 |
Arrrive :   0   0   0   0 |
Waiting :   6   8   9   7 |
--------------------------+
13 ( 13):
Exec :      0   6   4   5 |
Arrrive :   0   0   0   0 |
Waiting :   6   9   9   8 |
--------------------------+
14 ( 14):
Exec :      0   6   3   5 |
Arrrive :   0   0   0   0 |
Waiting :   6  10   9   9 |
--------------------------+
15 ( 15):
Exec :      0   6   3   4 |
Arrrive :   0   0   0   0 |
Waiting :   6  11  10   9 |
--------------------------+
16 ( 16):
Exec :      0   6   3   3 |
Arrrive :   0   0   0   0 |
Waiting :   6  12  11   9 |
--------------------------+
---------------------------------------- BACK ---------------------------
1 ( 17):
Exec :      8   1   2 |
Arrrive :   0   1   2 |
Waiting :   0   0   0 |
----------------------+
2 ( 18):
Exec :      7   1   2 |
Arrrive :   0   0   1 |
Waiting :   0   0   0 |
----------------------+
3 ( 19):
```

```
Exec :      7   0   2 |
Arrrive :   0   0   0 |
Waiting :   1   0   0 |
----------------------+
4 ( 20):
Exec :      7   0   1 |
Arrrive :   0   0   0 |
Waiting :   2   0   0 |
----------------------+
------------------------------------- INTERACTIVE ----------------------
1 ( 21):
Exec :      0   5   3   3 |
Arrrive :   0   0   0   0 |
Waiting :   6  12  12  10 |
--------------------------+
2 ( 22):
Exec :      0   4   3   3 |
Arrrive :   0   0   0   0 |
Waiting :   6  12  13  11 |
--------------------------+
3 ( 23):
Exec :      0   4   2   3 |
Arrrive :   0   0   0   0 |
Waiting :   6  13  13  12 |
--------------------------+
4 ( 24):
Exec :      0   4   1   3 |
Arrrive :   0   0   0   0 |
Waiting :   6  14  13  13 |
--------------------------+
5 ( 25):
Exec :      0   4   1   2 |
Arrrive :   0   0   0   0 |
Waiting :   6  15  14  13 |
--------------------------+
6 ( 26):
Exec :      0   4   1   1 |
Arrrive :   0   0   0   0 |
Waiting :   6  16  15  13 |
--------------------------+
7 ( 27):
Exec :      0   3   1   1 |
Arrrive :   0   0   0   0 |
Waiting :   6  16  16  14 |
--------------------------+
8 ( 28):
Exec :      0   2   1   1 |
Arrrive :   0   0   0   0 |
Waiting :   6  16  17  15 |
--------------------------+
9 ( 29):
Exec :      0   2   0   1 |
Arrrive :   0   0   0   0 |
```

```
Waiting :   6  17  17  16 |
-------------------------+
10 ( 30):
Exec :      0   2   0   0 |
Arrrive :   0   0   0   0 |
Waiting :   6  18  17  16 |
-------------------------+
11 ( 31):
Exec :      0   1   0   0 |
Arrrive :   0   0   0   0 |
Waiting :   6  18  17  16 |
-------------------------+
12 ( 32):
Exec :      0   0   0   0 |
Arrrive :   0   0   0   0 |
Waiting :   6  18  17  16 |
-------------------------+
---------------------------------------- BACK ----------------------------
1 ( 33):
Exec :      7   0   0 |
Arrrive :   0   0   0 |
Waiting :   3   0   0 |
----------------------+
2 ( 34):
Exec :      6   0   0 |
Arrrive :   0   0   0 |
Waiting :   3   0   0 |
----------------------+
3 ( 35):
Exec :      5   0   0 |
Arrrive :   0   0   0 |
Waiting :   3   0   0 |
----------------------+
4 ( 36):
Exec :      4   0   0 |
Arrrive :   0   0   0 |
Waiting :   3   0   0 |
----------------------+
1 ( 37):
Exec :      3   0   0 |
Arrrive :   0   0   0 |
Waiting :   3   0   0 |
----------------------+
2 ( 38):
Exec :      2   0   0 |
Arrrive :   0   0   0 |
Waiting :   3   0   0 |
----------------------+
3 ( 39):
Exec :      1   0   0 |
Arrrive :   0   0   0 |
Waiting :   3   0   0 |
----------------------+
```

```
4 ( 40):
Exec :      0    0    0 |
Arrrive :   0    0    0 |
Waiting :   3    0    0 |
----------------------+
Average wait time for SRTF: 1.00000
Average wait time for RR: 14.25000
```

Висновки

*Як ми бачимо, процеси алгоритму **Round Robin** проводять в часі очікування набагато більше часу, ніж процеси алгоритму **Shortest Remaining Time First**.* Це відбувається тому що в першому алгоритмі планування процеси проводять дуже багато часу в черзі на свій квант часу. Другий алгоритм більш оптимізований, бо обов'язково буде декілька процесів з маленьким часом виконання, і вони одразу виконаються, можливо більш важкі процеси зачекають, але через те, що доки швидкі процеси виконуються, час надходження довгих процесів зменшується, а в великому очікуванні проведуть тільки найважчі процеси, що добре впливає на загальній середній час очікування.

Розділення по часу було реалізовано за системою 20/80. Тобто на інтерактивні процеси виділено 80% часу на фонові – 20%. Коли/якщо інтерактивні процеси закінчать своє виконання, фонові будуть вже допрацьовувати 100% часу. У нашому прикладі 20 одиниць часу розподіляється як 4 одиниці часу на фонові процеси та 16 одиниць часу на інтерактивні. У програмі можна налаштувати кількість квантів на алгоритм планування ***Round Robin***, кількість часу, що буде розподілятися між процесами, кількість та «важкість» процесів.