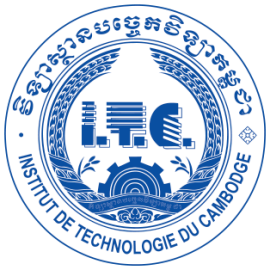


Institute of Technology of Cambodia

Department of Information and Communication Engineering

Project Report-Group 5

GIC-I4 (A)

Deadline: 01 July 2022

Course: Operating System II

Lecturer: Mr. Heng Rathpisey

2021~2022

Team members

	Name	ID
1.	Hun Ravit	e20180328
2.	Keo Linna	e20180368
3.	Kheang Sokuntheary	e20170337
4.	Hai MengKuong	e20180236

Table of Contents

I. Introduction

Virtual memory and process scheduler programs is a program that contain the function for calculate the page fault and success of virtual memory also average waiting time and average turnaround time of process scheduler. Users may simply to calculate virtual memory and process scheduler in their study for checking their result true or false by this program.

II. Purpose of the project

The purpose of this project is to build a program that can simulate **Virtual Memory** and **Process Scheduler**. And that it includes such as **FIFO, LRU, First-Come, First Served, Shortest Job Next, Shortest Remaining Time, Round Robin and Earliest Deadline First.** In this project we have divided all tasks to make it easier for users to find the values of each Virtual Memory or Process Scheduler by entering the value of each requirement in methods that we assign.

III. Implementation

Language	Java Programming
Data Structure	Array, Linked List, Queue
Class	Index, FIFO, LRU, FCFS, SJN, SRT, RoundRobin, EDF

There are functions in Index for manages all class together.

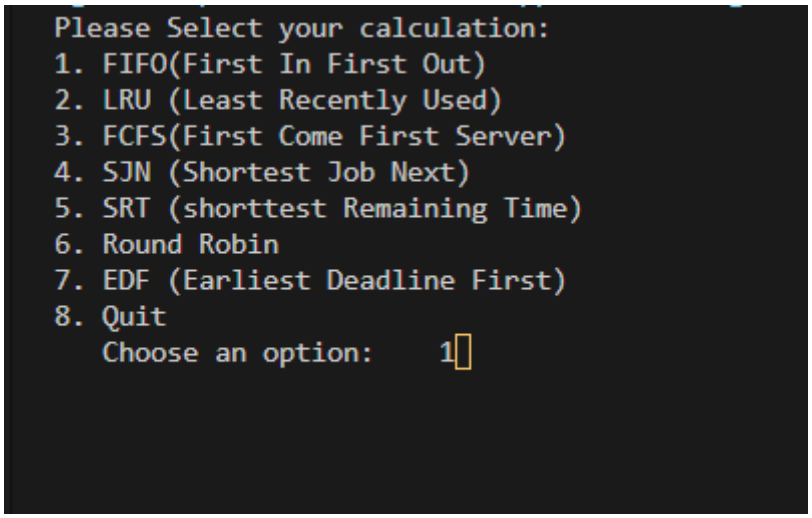
1. int menu(): Help user to find a class that right for their calculate
2. void ControlFIFO(): Used to control input and output of **FIFO.** We create an input in controlFIFO function and call class FIFO for finding number page fault and then we find success page by minus total of page with page fault.

After get all number of page success and page fault we can find page fault ratio and success ratio.

3. void ControlLRU(): Used to control input and output of **LRU.** We create an input in controlLRU function and call class LRU for find number page fault and after, we find page success, success ratio and page fault ratio the same First In First Out.
4. void ControlFCFS(): Used to control input and output of **FCFS.** We create function for manages FCFS and call it to menu for make it easier for us to manage. But calculate we put into a function name solve in class FCFS.
5. void ControlSJN(): Used to control input and output of **SJN.** We create function for manages SJN and calculate we put into function name Solve() in class SJN.
6. void ControlSRT(): Used to control input and output of **SRT.** Calculate function we put into function name solve in class SRT.
7. void ControlroundRobin(): Used to control input and output of **RoundRobin.** Calculate function we put into function name solve in class RoundRobin.
8. void ControlEDF(): Used to control input and output of **EDF.** Calculate function we put into function name solve in class EDF.

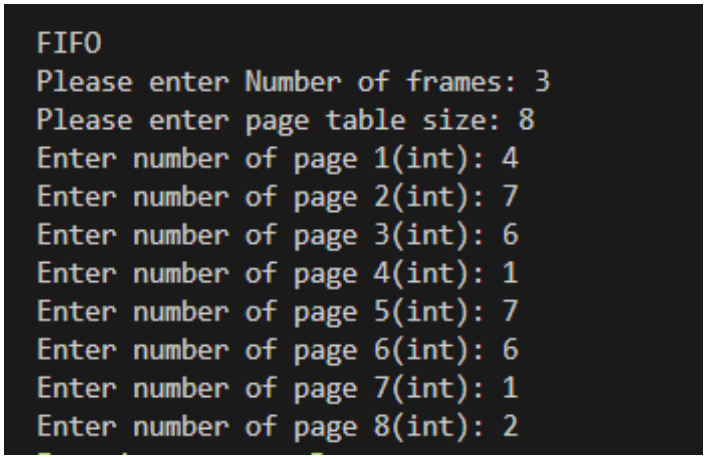
- **Index.java**: we create it for management classes and computational loop for calculate that make user and tester easy to calculate and test all our function by running programs one time and they just change menu number according to what they want to calculate.

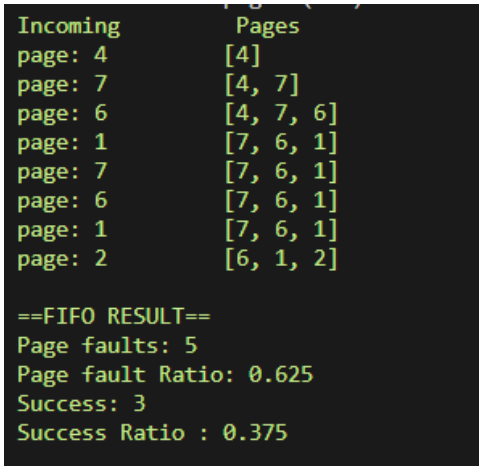
Result:



- **FIFO.java**: we create FIFO class for simulate FIFO (First in First out). FIFO is process of page replacement policy will remove the pages that have been in memory the longest. When users running programs, they have to enter data such as *Number of frames*, *page table size* and *Number of each page*. After they input success, it will return results such as **Page Faults**, **Page Fault Ratio, Success** and **Success Ratio.**

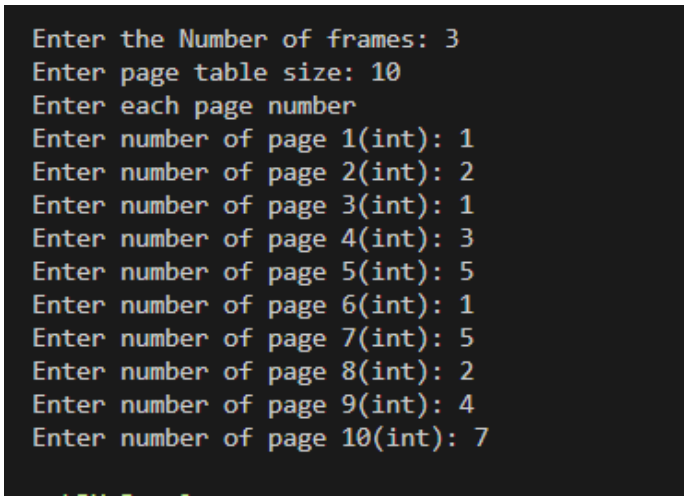
Input: **Result:**

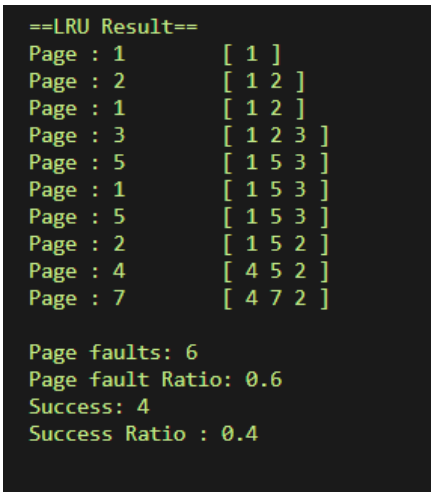




- **LRU.java**: we create LRU class for simulate LRU (Least Recently Used). LRU is process page replacement policy swaps out the page that show the least recent activity, figuring that these page are the least likely to be used again in the immediate future. When users running this programs they have to enter data such as *Number of frames*, *page table size* and *Number of each page*. After they input success, it will return results such as **Page Faults**, **Page Fault Ratio, Success** and **Success Ratio.**

Input: **Result:**





❖ **FCFS.java**: we create FCFS class for simulate FCFS (First Come First Served). FCFS is a non preemptive scheduling algorithm that handle all incoming objects according to their arrival time. The earlier they arrive the sooner they're served. When users want to calculate it, they have to enter data such as *Number of jobs, Arrival time of each job and CPU cycle time of each job*. After they input success, it will return result such as **Average Waiting Time** and **Average Turnaround Time.**

Input:

```
FCFS
Enter number of jobs: 3
Job1:
Enter Job 1 arrival time: 0
Enter Job 1 cpu cycle time: 15
Job2:
Enter Job 2 arrival time: 0
Enter Job 2 cpu cycle time: 2
Job3:
Enter Job 3 arrival time: 0
Enter Job 3 cpu cycle time: 1
```

Result:

```
FCFS RESULT:

pid  arrival  brust  complete turn waiting
1       0       15      15      15      0
2       0       2       17      17      15
3       0       1       18      18      17

Average waiting time: 10.666667
Average turnaround time:16.666666
```

❖ **SJN.java**: we create SJN class for simulate SJN (Shortest Job Next). SJN is non preemptive scheduling algorithm that handles jobs based on the length of their CPU cycle time. When users want to calculate it, they have to enter data such as *Number of Jobs, Arrival time of each job* and *CPU cycle time of each jobs*. After they input success, it will return result such as **Average Waiting Time** and **Average Turnaround Time.**

Input:

```
SJN
Enter no of process: 4
Enter Job 1 arrival time: 0
Enter Job 1 CPU cycle time: 5
Enter Job 2 arrival time: 0
Enter Job 2 CPU cycle time: 2
Enter Job 3 arrival time: 0
Enter Job 3 CPU cycle time: 6
Enter Job 4 arrival time: 0
Enter Job 4 CPU cycle time: 4
```

Result:

```
==SJN Result==

pid  arrival brust  complete turn waiting
1       0       5       11      11      6
2       0       2       2       2       0
3       0       6       17      17      11
4       0       4       6       6       2

Average Turnaround Time:9.0
Average Waiting Time:4.75
```

- **SRT.java**: we create SRT class for simulate SRT (Shortest Remaining Time). SRT is preemptive version of the SJN algorithm. It involves more overhead than SJN. When users want to calculate it, they have to enter data such as *Number of Jobs, Arrival time of each jobs* and *CPU cycle time of each jobs*. After they input success, it will return result such as **Average Waiting Time** and **Average Turnaround Time.**

Input:

```
SRT
Enter number of job: 4
Job1:
Enter Job 1 arrival time: 0
Enter Job 1 cpu cycle time: 6
Job2:
Enter Job 2 arrival time: 1
Enter Job 2 cpu cycle time: 3
Job3:
Enter Job 3 arrival time: 2
Enter Job 3 cpu cycle time: 1
Job4:
Enter Job 4 arrival time: 3
Enter Job 4 cpu cycle time: 4
```

Result:

```
==SJN Result==

Processes  Burst time  Waiting time  Turn around time
 1             6              8              14
 2             3              1              4
 3             1              0              1
 4             4              2              6

Average waiting time = 2.75
Average turn around time = 6.25
```

- **RoundRobin.java**: we create RoundRobin class for simulate RoundRobin. RoundRobin is a preemptive process scheduling algorithm that is used extensively in interactive systems. When users want to calculate it, they have to enter data such as *Time Quantum, Number of Jobs, Arrival time of each jobs* and *CPU cycle time of each jobs*. After they input success, it will return result such as **Average Waiting Time** and **Average Turnaround Time.**

Input:

```
Round RObin
Enter Time Quantum: 4
Enter number of job: 4
Job1:
Enter Job 1 arrival time: 0
Enter Job 1 cpu cycle time: 8
Job2:
Enter Job 2 arrival time: 1
Enter Job 2 cpu cycle time: 4
Job3:
Enter Job 3 arrival time: 2
Enter Job 3 cpu cycle time: 9
Job4:
Enter Job 4 arrival time: 3
Enter Job 4 cpu cycle time: 5
```

Result:

```
==Round Robin Result==

PN      Cycle Time      WT      TAT
 1              8       12      20
 2              4       4       8
 3              9       17      26
 4              5       20      25

Average waiting time = 13.25
Average turn around time = 19.75
```

- **EDF.java**: we create EDF class for simulate EDF (Earliest Deadline First). EDF is builds to address the critical processing requirements of real-time systems and their pressing deadlines. When users want to calculate it, they have to enter data such as *Number of Jobs, Arrival time of each job, CPU cycle time of each job* and *dateline of each job*. After they input success, it will return result as a Deadlines and part of job finish

Input: **Result:**

```
EDF
Enter number of processes : 6
Job1:
Enter Job 1 Arrival time: 0
Enter Job 1 Cpu cycle time: 3
Enter Job 1 Deadline of process : 6
Job2:
Enter Job 2 Arrival time: 0
Enter Job 2 Cpu cycle time: 1
Enter Job 2 Deadline of process : 2
Job3:
Enter Job 3 Arrival time: 0
Enter Job 3 Cpu cycle time: 5
Enter Job 3 Deadline of process : 10
Job4:
Enter Job 4 Arrival time: 0
Enter Job 4 Cpu cycle time: 2
Enter Job 4 Deadline of process : 8
Job5:
Enter Job 5 Arrival time: 0
Enter Job 5 Cpu cycle time: 2
Enter Job 5 Deadline of process : 8
Job6:
Enter Job 6 Arrival time: 0
Enter Job 6 Cpu cycle time: 5
Enter Job 6 Deadline of process : 15
```

```
==EDR Diagram==
00 _____
   |     |
   |job 2|
   |_____|
1  _____
   |     |
   |job 1|
   |_____|
4  _____
   |     |
   |job 4|
   |_____|
6  _____
   |     |
   |job 5|
   |_____|
8  _____
   |     |
   |job 3|
   |_____|
13 _____
   |     |
   |job 6|
   |_____|
18
```

IV. Result

A. Responsibility

Name	Responsibility
Hun Ravit	FIFO, LRU, Index
Keo Linna	SJN, SRT
Kheang Sokuntheary	EDF
Hai MengKuong	Round Robin

B. Success Function

In our mini project, there are seven main functions that are control on class and easy to manage class that we used to simulated virtual memory and Process Schedular. Our team can complete them successfully and on time as well.

V. Conclusion

In conclusion, after completing this project as a team we have learned the power of teamwork and we have a better understanding of the functions that are being used in this project. We have also learned about new algorithms that can improve our coding fast. And we also learn about research and improve soft skills to be a good team worker. One more, this project also makes us understand about the lessons more than before.