EOPSY LABORATORY REPORT

TASK 3

OMER AMAC

323796

TABLE OF CONTENTS

I. INTRODUCTION

1. Scheduling

The process scheduling is the activity of the process manager that handles the removal of the running process from the CPU and the selection of another process on the basis of a particular strategy. Process scheduling is an essential part of a Multiprogramming operating systems. Such operating systems allow more than one process to be loaded into the executable memory at a time and the loaded process shares the CPU using time multiplexing.

2. About the First-Come First-Served Algorithm and Preemptive/Non-preemptive Scheduling

Preemptive: Preemptive Scheduling is a CPU scheduling technique that works by dividing time slots of CPU to a given process. The time slot given might be able to complete the whole process or might not be able to it. When the burst time of the process is greater than CPU cycle, it is placed back into the ready queue and will execute in the next chance. This scheduling is used when the process switch to ready state.

Nonpreemptive: Non-preemptive Scheduling is a CPU scheduling technique the process takes the resource (CPU time) and holds it till the process gets terminated or is pushed to the waiting state. No process is interrupted until it is completed, and after that processor switches to another process.[i]

First-Come First-Served Algorithm: First Come First Serve (FCFS) is an operating system scheduling algorithm that automatically executes queued requests and processes in order of their arrival. It is the easiest and simplest CPU scheduling algorithm. In this type of algorithm, processes which requests the CPU first get the CPU allocation first. This is managed with a FIFO queue.[ii]

3. Finite State Machine

A finite state machine (sometimes called a finite state automaton) is a computation model that can be implemented with hardware or software and can be used to simulate sequential logic and some computer programs. Finite state automata generate regular languages. Finite state machines can be used to model problems in many fields including mathematics, artificial intelligence, games, and linguistics.[iii]

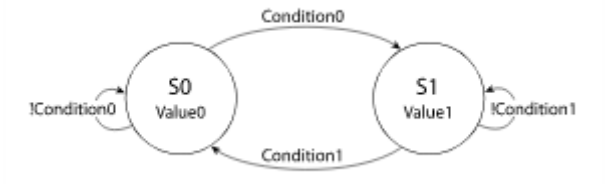


FIGURE 1: FINITE STATE MACHINE EXAMPLE

A system where particular inputs cause particular changes in state can be represented using finite state machines.

4. About The Task

In this project we will create three different scenarios and simulate and run with Nonpreemptive Scheduling. We will use 2, 5, and 10 processes. We will store the results in log files and observe after. Each process will run for 2000 ms, and standard deviation will be 0, we will block every 500 ms and total runtime will be 10000ms.

Goal

The goal of our experiment is understanding the work of finite state machine and the examining the Nonpreemptive model and the FCFS algorithm.

5. Configuration File

```
// # of Process
numprocess 10

// mean deivation
meandev 2000

// standard deviation
standdev 0

// process    # I/O blocking
process 500
process 500
process 500
process 500
process 500
process 500
process 500
process 500
process 500
process 500


// duration of the simulation in milliseconds
runtime 10000
```

numprocess : Total number of processes

meandev : The average length of time in milliseconds that a process should execute before terminating.

standdev : The number of standard deviations from the average length of time a process should execute before terminating.

process : The amount of time in milliseconds that the process should execute before blocking for input or output. There should be a separate process directive for each process specified by the numprocess directive.

runtime : The maximum amount of time the simulation should run in milliseconds.

II. SIMULATION

1. CONFIGURATION WITH 2 PROCESSES

```
oamac@DESKTOP-QC6TRCK:~/task3$ cat Summary-Processes
Process: 0 registered... (2000 500 0 0)
Process: 0 I/O blocked... (2000 500 500 500)
Process: 1 registered... (2000 500 0 0)
Process: 1 I/O blocked... (2000 500 500 500)
Process: 0 registered... (2000 500 500 500)
Process: 0 I/O blocked... (2000 500 1000 1000)
Process: 1 registered... (2000 500 500 500)
Process: 1 I/O blocked... (2000 500 1000 1000)
Process: 0 registered... (2000 500 1000 1000)
Process: 0 I/O blocked... (2000 500 1500 1500)
Process: 1 registered... (2000 500 1000 1000)
Process: 1 I/O blocked... (2000 500 1500 1500)
Process: 0 registered... (2000 500 1500 1500)
Process: 0 completed... (2000 500 2000 2000)
Process: 1 registered... (2000 500 1500 1500)
Process: 1 completed... (2000 500 2000 2000)
```

FIGURE 2: PROCESS SUMMARY FOR 2 PROCESSES

We can see our 2 processes in the summary and the order of their blocking and completion. At the bottom, we can observe that both processes were completed.

```
oamac@DESKTOP-QC6TRCK:~/task3$ cat Summary-Results
Scheduling Type: Batch (Nonpreemptive)
Scheduling Name: First-Come First-Served
Simulation Run Time: 4000
Mean: 2000
Standard Deviation: 0
Process #       CPU Time        IO Blocking     CPU Completed   CPU Blocked
0               2000 (ms)       500 (ms)        2000 (ms)       3 times
1               2000 (ms)       500 (ms)        2000 (ms)       3 times
```

FIGURE 3: FINAL RESULT FOR 2 PROCESSES

We had 10000 milliseconds runtime for it but we can easily see that the total time for completion is 4000 milliseconds, which is 2000 ms for each process. It shows that our processes finished earlier. Both processes were blocked 3 times because every 500 ms we blocked the process and from start to end, there were 3 stops for each process. It means that process is divided into 4 parts. Also, we can observe that 0 started first and completed first, as we can see from the algorithm name First-Come First-Served.

2. CONFIGURATION WITH 5 PROCESSES

```
oamac@DESKTOP-QC6TRCK:~/task3$ cat Summary-Processes
Process: 0 registered... (2000 500 0 0)
Process: 0 I/O blocked... (2000 500 500 500)
Process: 1 registered... (2000 500 0 0)
Process: 1 I/O blocked... (2000 500 500 500)
Process: 0 registered... (2000 500 500 500)
Process: 0 I/O blocked... (2000 500 1000 1000)
Process: 1 registered... (2000 500 500 500)
Process: 1 I/O blocked... (2000 500 1000 1000)
Process: 0 registered... (2000 500 1000 1000)
Process: 0 I/O blocked... (2000 500 1500 1500)
Process: 1 registered... (2000 500 1000 1000)
Process: 1 I/O blocked... (2000 500 1500 1500)
Process: 0 registered... (2000 500 1500 1500)
Process: 0 completed... (2000 500 2000 2000)
Process: 1 registered... (2000 500 1500 1500)
Process: 1 completed... (2000 500 2000 2000)
Process: 2 registered... (2000 500 0 0)
Process: 2 I/O blocked... (2000 500 500 500)
Process: 3 registered... (2000 500 0 0)
Process: 3 I/O blocked... (2000 500 500 500)
Process: 2 registered... (2000 500 500 500)
Process: 2 I/O blocked... (2000 500 1000 1000)
Process: 3 registered... (2000 500 500 500)
Process: 3 I/O blocked... (2000 500 1000 1000)
Process: 2 registered... (2000 500 1000 1000)
Process: 2 I/O blocked... (2000 500 1500 1500)
Process: 3 registered... (2000 500 1000 1000)
Process: 3 I/O blocked... (2000 500 1500 1500)
Process: 2 registered... (2000 500 1500 1500)
Process: 2 completed... (2000 500 2000 2000)
Process: 3 registered... (2000 500 1500 1500)
Process: 3 completed... (2000 500 2000 2000)
Process: 4 registered... (2000 500 0 0)
Process: 4 I/O blocked... (2000 500 500 500)
Process: 4 registered... (2000 500 500 500)
Process: 4 I/O blocked... (2000 500 1000 1000)
Process: 4 registered... (2000 500 1000 1000)
Process: 4 I/O blocked... (2000 500 1500 1500)
Process: 4 registered... (2000 500 1500 1500)
```

FIGURE 4: PROCESS SUMMARY FOR 5 PROCESSES

We observe that the first four processes were completed but the last process did not appear as completed. When 0 and 1 were completed, 2 and 3 started to run. We observe that processes are run by pairs. Only process 4 was alone. So, no process blocked it and it was running all by itself without waiting for another.

```
oamac@DESKTOP-QC6TRCK:~/task3$ cat Summary-Results
Scheduling Type: Batch (Nonpreemptive)
Scheduling Name: First-Come First-Served
Simulation Run Time: 10000
Mean: 2000
Standard Deviation: 0
Process #       CPU Time        IO Blocking     CPU Completed   CPU Blocked
0               2000 (ms)       500 (ms)        2000 (ms)       3 times
1               2000 (ms)       500 (ms)        2000 (ms)       3 times
2               2000 (ms)       500 (ms)        2000 (ms)       3 times
3               2000 (ms)       500 (ms)        2000 (ms)       3 times
4               2000 (ms)       500 (ms)        2000 (ms)       3 times
```

FIGURE 5: FINAL RESULT FOR 5 PROCESSES

When we check the CPU Completed column, we see that all processes ran for 2000 ms, and process 4 looks completed as well.

3. CONFIGURATION WITH 10 PROCESSES

```
oamac@DESKTOP-QC6TRCK:~/task3$ cat Summary-Processes
Process: 0 registered... (2000 500 0 0)
Process: 0 I/O blocked... (2000 500 500 500)
Process: 1 registered... (2000 500 0 0)
Process: 1 I/O blocked... (2000 500 500 500)
Process: 0 registered... (2000 500 500 500)
Process: 0 I/O blocked... (2000 500 1000 1000)
Process: 1 registered... (2000 500 500 500)
Process: 1 I/O blocked... (2000 500 1000 1000)
Process: 0 registered... (2000 500 1000 1000)
Process: 0 I/O blocked... (2000 500 1500 1500)
Process: 1 registered... (2000 500 1000 1000)
Process: 1 I/O blocked... (2000 500 1500 1500)
Process: 0 registered... (2000 500 1500 1500)
Process: 0 completed... (2000 500 2000 2000)
Process: 1 registered... (2000 500 1500 1500)
Process: 1 completed... (2000 500 2000 2000)
Process: 2 registered... (2000 500 0 0)
Process: 2 I/O blocked... (2000 500 500 500)
Process: 3 registered... (2000 500 0 0)
Process: 3 I/O blocked... (2000 500 500 500)
Process: 2 registered... (2000 500 500 500)
Process: 2 I/O blocked... (2000 500 1000 1000)
Process: 3 registered... (2000 500 500 500)
Process: 3 I/O blocked... (2000 500 1000 1000)
Process: 2 registered... (2000 500 1000 1000)
Process: 2 I/O blocked... (2000 500 1500 1500)
Process: 3 registered... (2000 500 1000 1000)
Process: 3 I/O blocked... (2000 500 1500 1500)
Process: 2 registered... (2000 500 1500 1500)
Process: 2 completed... (2000 500 2000 2000)
Process: 3 registered... (2000 500 1500 1500)
Process: 3 completed... (2000 500 2000 2000)
Process: 4 registered... (2000 500 0 0)
Process: 4 I/O blocked... (2000 500 500 500)
Process: 5 registered... (2000 500 0 0)
Process: 5 I/O blocked... (2000 500 500 500)
Process: 4 registered... (2000 500 500 500)
Process: 4 I/O blocked... (2000 500 1000 1000)
Process: 5 registered... (2000 500 500 500)
```

FIGURE 6: PROCESS SUMMARY FOR 10 PROCESSES

Now, we observe something interesting. We have a total of 10 processes but we only can see the first six processes in the log file. Again processes were running by pairs and it shows that 0 -1, 2-3, and 4-5 were displayed. However, processes 4 and 4 did not completed. Let's check the Summary-results and see the reason.

```
oamac@DESKTOP-QC6TRCK:~/task3$ cat Summary-Results
Scheduling Type: Batch (Nonpreemptive)
Scheduling Name: First-Come First-Served
Simulation Run Time: 10000
Mean: 2000
Standard Deviation: 0
Process #       CPU Time        IO Blocking     CPU Completed   CPU Blocked
0               2000 (ms)       500 (ms)        2000 (ms)       3 times
1               2000 (ms)       500 (ms)        2000 (ms)       3 times
2               2000 (ms)       500 (ms)        2000 (ms)       3 times
3               2000 (ms)       500 (ms)        2000 (ms)       3 times
4               2000 (ms)       500 (ms)        1000 (ms)       2 times
5               2000 (ms)       500 (ms)        1000 (ms)       1 times
6               2000 (ms)       500 (ms)        0 (ms)          0 times
7               2000 (ms)       500 (ms)        0 (ms)          0 times
8               2000 (ms)       500 (ms)        0 (ms)          0 times
9               2000 (ms)       500 (ms)        0 (ms)          0 times
```

FIGURE 7: FINAL RESULT FOR 10 PROCESSES

Now, it is clear that processes 0, 1, 2, and 3 were completed. 2000 ms CPU time was completed for the first four processes and the fifth and sixth processes started as a pair.

$$4 \times 2000 \text{ ms} = 8000 \text{ ms}$$

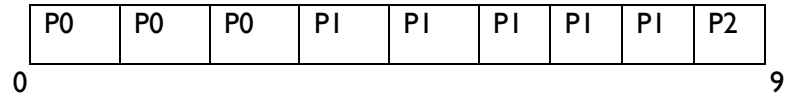
$$10000 - 8000 = 2000 \text{ ms}$$

So in the end, we had a 2000 ms run time for the rest of the process, and for the next pair of the process, we had 1000 ms for each. In the end, They also could not have enough time to terminate. Also, an interesting thing is process 4 was blocked 2 times but 5 was blocked 1 time, we can also refer that thing to the First-Come First-Served algorithm.

III. DISCUSSION

In our example we used FCFS algorithm and output was the same order as input order. However, in some real life cases it may not be efficient. Because all processes can have different features. Let's choose the Round Robin algorithm and compare with the FCFS.

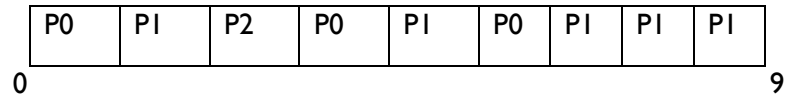
With FCFS algorithm they will be treated with their arrival time. Let's assume that time starts from 0 and each square is 1 unit. All arrived at time 0.



As we can see, P0 will be finished first. However, we can start to P2 at time 8. To run a process, we have to wait fort he previous ones to complete.

Termination order will be same as input order. P0 -> P1 -> P2

Round Robin Algorithm: Round Robin is a CPU scheduling algorithm where each process is assigned a fixed time slot in a cyclic way. (Processor Sharing algorithm)



Now all arrived at time 0 but they are sharing the CPU. Because of that, all processes will act like a package, completed ones will be terminated without waiting for other ones.

Termination order will be depend on their execution time : P2 -> P0 -> P1

IV. CONCLUSION

We had a chance to observe how the Nonpreemtive Scheduling works. As it was in the theoritical explanation of it, it was not interrupted and each pair of process waited for previous one to finish. We had a chance to observe the effect of the run-time. When we limit the run-time and increase the number of processes, some of them were not able to terminate and even some of them could not start. We can conclude that, we need runtime as (number of process x mean deviation). If we do not provide that it may cause some problems in some applications.

Our goal was the understanding the work of states of processes and examining the algorithm with compare to theorical explanation. We find out that, our experiment results were confirming the theorical explanation.

To understand better, we can implement Round Robin algorithm. Also, we can change the execution times of the processes. That will make things more complex but it will give us more clear information about the scheduling.

V. REFERENCES

[i] Preemptive and Non-Preemptive Scheduling (tutorialspoint.com)
[ii] FCFS Scheduling Algorithm: What is, Example Program (guru99.com)
[iii] Finite State Machines | Brilliant Math & Science Wiki