[EOPSY] Laboratory task 3 : Process Simulation

- Rajagopal Thirugnanasambandam(309263)

==========

## Instruction :

Create a configuration file in which all processes run an average of 2000 milliseconds with a standard deviation of zero, and which are blocked for input or output every 500 milliseconds. Run the simulation for 10000 milliseconds with 2 processes. Examine the two output files. Try again for 5 processes. Try again for 10 processes. Explain what's happening.

==========

1. Simulation for two (2) processes

==========

## Configuration:

```
// # of Process
numprocess 2
// mean deivation
meandev 2000
// standard deviation
standdev 0
// process     # I/O blocking
process 500
process 500
// duration of the simulation in milliseconds
runtime 10000
```

==========

After running the scheduling configuration using make run in the system

---

**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 |

---

**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)

---

**Analysis of the simulation :**

Once 2 processes for a total of 4000ms have been simulated, you can observe from the output Summary Processes file that the other process is blocked when one process is running/executed.

According to the settings, after 500ms, a process is blocked, and the previously blocked process is now active and operating.

The CPU, as can be seen, ran processes in pairs and sequentially. The context switching mechanism manages the processor's time and divides it evenly between the two processes.

The simulation lasted 4000ms since each procedure took 2000ms to complete.

The processes were executed using the non-preemptive scheduling technique, which means that a process begins running only when an I/O block has halted the previously active process.

A process may only start or continue after 500ms, which is the I/O blockage time, according to our settings.

==========

2. Simulation for five (5) processes

==========

Configuration:

// # of Process

numprocess 5

// mean deivation

meandev 2000

// standard deviation

standdev 0

// process  # I/O blocking

process 500

process 500

process 500

process 500

process 500

// duration of the simulation in milliseconds

runtime 10000

==========

After running the scheduling configuration using make run in the system

---

**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 |

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

**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)

==========

**Analysis of the simulation :**

Once running the second simulation with the number of processes raised to five (5), we can see that the overall period of conducting the simulation occupied the entire allotted time we provided in the setup.

5 processes * 2000ms each process = 10000ms simulation time.

This could be noted that, similar to the previous simulation with two (2) processes, the processes were ran and then swapped in pairs after being stuck.

The only difference was that by the time the processor began processing the last process (5th process, Process :4), all other processes had completed their execution since their allotted execution time had expired. As a result,

the processor just completed the last process by itself, stalling the process every 500ms.

==========

3. Simulation for ten (10) processes

==========

**Configuration:**

// # 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

==========

After running the scheduling configuration using make run in the system

---

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 |

---

**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)

==========

**Analysis of the simulation :**

As observed in the third example, they all work just like the first and second simulations. Because non-preemptive scheduling is utilized here, and processes are switched and run in pairs one after the other, just like in the first and second simulations. The only difference in this simulation (third simulation) was that the allocated time specified in our configuration,

which was 10000ms, was insufficient to run/execute all processes, hence as seen above in the Summary processes output file, only the first four (4) processes completed their execution while the fifth and sixth processes were still in execution or did not exceed their time limit.

As can be seen on the summary results output file, the last remaining 4 processes did not start as the simulation time was already exceeded.

A simple overview of the context switching technique may be seen in this simulation, where the CPU was permitted to be shared by numerous processes by storing and restoring the process's processor state (context).

This is accomplished by using the configured I/O blockage time (500ms) for each process, instructing the scheduler to run, block, or switch when a process has surpassed its CPU time interval.

We employed non-preemptive scheduling, in which processes were run in pairs one after the other. Non-preemption, on the other hand, can be accomplished in two ways. We either wait for the process to complete its execution or the process willingly gives over the CPU to another process.

We simply told the process to give up CPU time by employing I/O blockage, and we set minimum execution timeframes for each process.

One of the bad sides to this scheduling approach is that if a program is not designed properly, that program may take the whole processor's time and other

programs/processes will not be able to execute unless it has finished executing first.

==========

**First Come First Serve (FCFS) Algorithm:**

First Come First Serve 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. The full form of FCFS is First Come First Serve. As the process enters the ready queue, its PCB (Process Control Block) is linked with the tail of the queue and, when the CPU becomes free, it should be assigned to the process at the beginning of the queue.

==========

**Why First come First Serve Algorithm is better than Round robin Algorithm?**

FCFS algorithm doesn't include any complex logic, it just puts the process requests in a queue and executes it one by one. Hence, FCFS is pretty simple and easy to implement. Eventually, every process will get a chance to run, so starvation doesn't occur. In Round Robin Algorithm Each process is served by the CPU for a fixed time quantum, so the throughput in Round robin is largely depends on the choice of the length of the time quantum. If time quantum is longer than needed, it tends to exhibit the same behavior as FCFS. If time quantum is shorter than needed, the number of times that CPU switches from one process to another process, increases. This leads to decrease in CPU efficiency.

==========

==========