**Operating Systems**

**Lab 3**

**Kaushikram Krishnamoorthy**

**302335**

**Aim:**

The objective of this task is to create multiple processes in MOSS Scheduling simulator and observe how the process scheduling works for different number of processes to be executed at same time.

We set an average run time of 2000 milliseconds, standard deviation of zero and which are blocked for I/O operations for every 500 milliseconds with total simulation time of 10000 milliseconds.

**Introduction:**

* **CPU Scheduling:**

CPU scheduling is the basis of multiprogrammed operating system. By switching the CPU among processes. The operating system can make the computer more productive. It allows one process to use the CPU while the execution of another is in waiting state due any reason for example – unavailability of resources etc. hence making the full use of CPU which in turn makes the system fast and efficient.

As the CPU gets idle, the operating system selects one of the processes present in ready queue to be executed.

* **Types of CPU scheduling:**

There are four circumstances under which CPU scheduling decisions may take place:

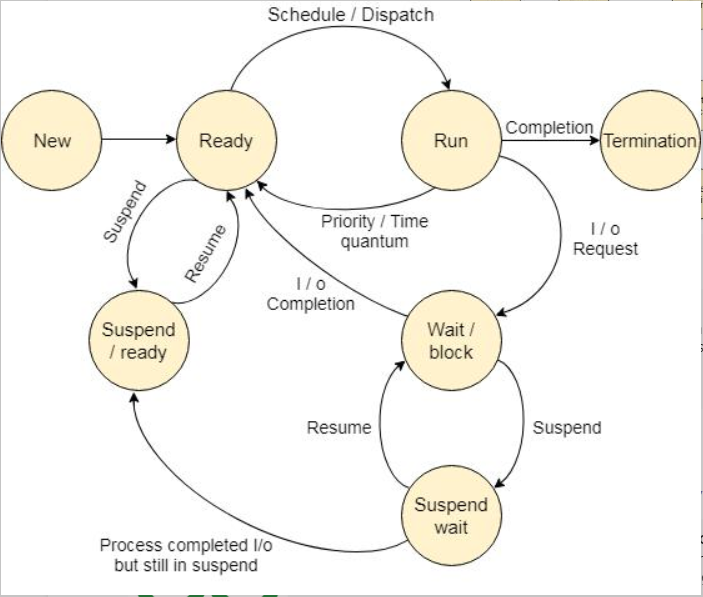
1. When a process sees a shift from running state to waiting state for I/O request or calling for wait for the termination of one of the child processes.

2. When a process sees a shift from running state to ready state. For example – occurrence of interrupt.

3. When a process sees a shift from waiting state to ready state. For example – I/O completion.

4. When a process gets terminated.

Generally, a processes goes through several states from its creation to its termination. A diagram representing such states is shown below.



A program or several lines of code that is going to be picked up by the operating system from secondary memory to its main memory to initiate its execution is said to be in its new state. In ready state, the process gets picked up in main memory and it waits for the execution by the processor(waits for the CPU to be assigned). A process can go to either suspend/ready state or run state from the ready state. If there exists any other process which has a higher priority than the current process, it comes to the ready state, and at the same time if there is no free space in main memory the current process is suspended and its send to suspend/ready state and it is brought back again to ready state when the main memory is available. If the main memory is free it will go to the run state where the process is assigned to the CPU execution. From the run state, the process will terminate(goes to termination state) if the process does not have any I/O operation or any other priorities. If it has any resource to be assigned or any I/O operation it is sent to wait/block state. Again in this state if there is another process with higher priority and the main memory is full the current process is sent to suspend wait state where it waits until the main memory becomes free or available. When it is available it is sent back to the block state. From suspend wait state it can also be sent to suspend ready state when the I/O operation is completed but the main memory is unavailable. After the completion of I/O operation in wait/block state the process is sent to ready state and the same repeats again.

When the CPU is free(not totally occupied by processes) the processes which are in the ready queue should be selected. This selection process is done by the CPU scheduler. After a process is selected the dispatcher gives actual control of CPU to the process.

When a process goes from running state to waiting state or when a process terminates the only choice left for the CPU scheduler is to schedule the next process which is waiting in the ready queue, this scheduling scheme is called nonpreemptive scheduling. When a process goes from running sate to ready state when it is interrupted or from waiting state to ready state after completing I/O operation they are not yet finished and still waits for CPU to complete its execution. In this situation the CPU schedules using different algorithms and selects the process with highest priority, this scheduling scheme is called preemptive scheduling.

**Results:**

**Results with 2 processes:**

*Config:*

// # 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

*Outputs:*

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

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)

*Observations:*

We observe that both processes are completed successfully. First “Process 0” is completed, then “Process 1” is completed. This is due to the “First Come First Serve” rule. Both processes are also alternatively started, blocked and then finally completed.

**Results with 5 processes:**

*Config:*

// # 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

*Outputs:*

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

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)

*Observations:*

Firstly, process 0 gets registered and after 500ms gets blocked. Then, process 1 gets registered and then 500ms gets blocked. This cycle goes on for 4 times and then Process 0 gets completed and after that process 1 gets completed. Likewise, process 2 and process 3 gets completed respectively. And in last process 4 gets registered and after 500ms gets blocked. This cycle gets repeated for 3 times and in 4th time process 4 gets registered but doesn’t get completed. This happens because the max run time is reached exactly when process 4 is gets completed due to which there is no notification of process 4 getting complete.

**Results with 10 processes:**

*Config:*

// # 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

*Outputs:*

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

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)

*Observations:*

In this case, we run for 10 processes. Now for the first 8000ms everything was performed same as in the previous task, where we had taken 5 processes. In these 8000ms, process 0, 1, 2 and 3 get completed respectively but after that Processes 4 gets registered and then get blocked after 500ms. Then process 5 gets registered and then gets blocked. Again process 4 gets registered and after 500ms gets blocked. And again process 5 gets registered and after 500ms when process 5 was just about to get blocked the max run time gets saturated. Hence, blocking of process 5 doesn’t get notified in summary processes. Both process 4 and 5 run for 1000ms each. Process 4 gets blocked for twice whereas process 5 gets blocked for once.

**Conclusions:**

We clearly see that CPU does work with all of those processes one after another, there is no concurrency of these processes - when one process is working, CPU is entirely dedicated to that process. When the CPU is blocked for one process, the next process will use it while it can- the CPU will go back to the previous process when that process is blocked too.

"Scheduling Name: First-Come First-Served"

Scheduler will always prioritize processes which are not blocked and are invoked earliest when choosing which process to handle at the moment.

We see that scheduler prefers finishing current process over switching to other processes with same priority. When CPU block occurs, the scheduler is forced to switch currently handled process to some other - at that moment the scheduler needs to evaluate which process has highest priority, since all of our processes have the same priority, scheduler decides to handle them all one after another, in the order of invocation.

We also see that with 10 processes, there is not enough time to handle each process during the simulation - scheduler does not see how long it will be working, so it works without changes until the simulation ends.

At this time, all the processes are stopped.