

**Lab No. 13**

Mehran University of Engineering and Technology Shaheed Zulfiquar Ali Bhutto Campus, Khairpur Mir’s Department of Software Engineering

**Course: Operating System SW-225 (Practical) - Instructor: Engr. Shamshad Naveed**

**Objective: Simulating CPU Scheduling Algorithm**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name:** | **Mohammad Saleem** | **Roll Number:** | | **K20SW033** |
| **Score:** | **Signature:** | **Date:** | **28-10-2022** | |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **LAB PERFORMANCE INDICATOR** | SUBJECT KNOWLEDGE | DATA ANALYSIS AND INTERPRETATION | ABILITY TO CONDUCT EXPERIMENT | PRESENTATION | CALCULATION AND CODING | OBSERVATION  /RESULTS | SCORE |
|  |  |  |  |  |  |  |  |

**Task :**

**Simulating CPU Scheduling Algorithm ?**

Scheduling is a fundamental operating-system function. The concept is to have computer resources shared by a number of processes. Almost all computer resources are scheduled before use. The CPU is, of course, one of the primary computer resources. Thus, its scheduling is central to an operating-system’s design and constitutes an important topic in the computer science curriculum. However, the main difficulty in learning CPU scheduling from a textbook is the lack of interactivity inherent in its static representation. Moreover, textbooks often simplify the illustration of various CPU scheduling algorithms by using an unrealistic process model. For example, although almost all processes alternate CPU bursts with I/O bursts (i.e., alternate bursts of computing with I/O requests), only one CPU burst per process is used in most textbook examples. As a result, students are not able to gain insight into exactly how the algorithms work in real operating systems. Studies have shown that, for many types of computer science algorithms, animation can do much to enhance learning and understanding. Therefore, the author has developed a simulator that uses graphical animation to convey the concepts of various scheduling algorithms for a single CPU. The simulator is unique in a number of respects. First, it uses a more realistic process model that can be configured easily by the user. Second, it graphically depicts each process’ state versus time. The state of a process describes the current activity of that process such as “the process is waiting for an I/O operation to complete” or “the process is currently using the CPU”. Various events can cause a process to change state; the simulator shows these events. Using this representation, it becomes much easier to understand what is going on inside the system, why, at any given time, some processes are candidates for the allocation of the CPU and some are not, and why the currently running process can continue using the CPU or why it cannot. A third unique feature of the simulator is that it allows the user to test and increase his understanding of the concepts he has learnt through its very easy-to-use graphical user interface.

More specifically, the simulator has two operating modes: simulation mode and practice mode. In simulation mode, the user can watch virtually step-by-step how an algorithm works or watch it straight through from the beginning until the end, to achieve a better conceptual understanding of the algorithm. In practice mode, the user can reinforce the concepts studied by making his own scheduling decisions, that is, deciding when and for how long each process runs. The author believes that it is important for the simulator to have not only a simulation mode but also a practice mode, since it has been reported that learners who are actively engaged with visualization technology have consistently outperformed learners who passively view graphics.

For example, Byrne et al. conducted an experiment in which viewers were forced to make predictions about what they would see during an animation. These viewers scored significantly better on a post-test than others who merely watched identical animation without making such predictions. The remainder of this paper is organized as follows: the next section is a brief overview of the process state and scheduling algorithms used in the simulator, the section after that gives a description of the simulator, followed by a section that discusses related work, and the final section draws some conclusions. The CPU scheduler, which is part of the operating system of a computer, manages the allocation of the CPU among processes. A process is said to be running in the running state if it is currently using the CPU. A process is said to be ready in the ready state if it could use the CPU if it were available. A process is said to be blocked in the waiting state if it is waiting for some event to happen, such as an I/O completion event, before it can proceed. Various events can cause a process to change state. For example, when the currently running process makes an I/O request, it will change from running state to waiting state. When its I/O request completes, an I/O interrupt is generated and then that process will change from waiting state to ready state. For a single CPU system, only one process may be running at a time, but several processes may be ready and several may be blocked. All ready processes are kept on a ready queue. All blocked processes are placed on an I/O queue for the requested I/O device. The scheduler uses a scheduling algorithm to decide which process from the ready queue to run when and for how long. There are various scheduling algorithms. The simulator uses the algorithms listed below (which are discussed in

# First-Come, First-Served (FCFS)

Processes are assigned the CPU in the order they request it.

# Round-Robin (RR):

Each process is given a limited amount of CPU time, called a time slice, to execute. If the required CPU burst of the process is less than or equal to the time slice, it releases the CPU voluntarily. Otherwise, the scheduler will preempt the running process after one time slice and put it at the back of the ready queue, then dispatch another process from the ready queue.

# Shortest-Job-First (SJF):

When the CPU is available, it is allocated to the process that has the smallest next CPU burst.

# Shortest-Remaining-Time-First (SRTF):

When the CPU is available, it is allocated to the process that has the shortest remaining CPU burst. When a process arrives at the ready queue, it may have a shorter remaining CPU burst than the currently running process. Accordingly, the scheduler will preempt the currently running process.

# Multilevel Feedback Queues (MLFQ):

There are several ready queues, each with different priority. When the CPU is available, the scheduler selects a process from the highest-priority, non-empty ready queue. Within a queue, it uses RR scheduling. The scheduler adjusts the priority of a process dynamically, for example, to reflect resource requirements (e.g., being blocked awaiting an event) and the amount of resources consumed by the process (e.g., CPU time). Processes are moved between ready queues based on changes in their priority. When a process other than the currently running process attains a higher priority, the scheduler will preempt the currently running process and add it to the appropriate ready queue.

