Scheduling on Kernel Threads

Operating Systems

21K-4939 Umer Khan

INTRODUCTION

The kernel thread scheduling is the process of choosing which thread from a pool of kernel-level threads should run on a specific processor core at a given time. A thread that is created and controlled by the operating system's kernel as opposed to a user-level application is referred to as a kernel thread. Managing I/O operations and handling interrupts are two examples of tasks that are frequently handled by kernel threads. These tasks require direct access to the kernel. Scheduling kernel threads is a crucial task in a multi-core system to guarantee effective use of processor resources. A processor core is assigned to each available kernel thread by the scheduler based on several factors, such as resource utilization, execution time, and priority level. The equitable and effective distribution of the kernel threads among the available processor cores is another requirement for the scheduler.

MOTIVATION

I decided to demonstrate our command over scheduling algorithms through a code-based approach in order to understand and demonstrate our command.

PROBLEM STATEMENT

The difficulty lies in creating a scheduler that equitably distributes kernel threads among the available processor cores by taking into account variables like priority, execution time, and resource utilization. The project's goal is to apply and evaluate various scheduling algorithms to determine which strategy is best.

METHODOLOGY, RESULTS, DISCUSSIONS AND EXPLANATION OF ALGORITHMS

The project incorporates 5 sorts of scheduling using kernel threads that are as follows:

1. Round Robin Scheduling
2. Priority Scheduling
3. Shortest Job First
4. Multilevel Feedback Queue
5. First Come First Serve

Round Robin Scheduling

Round Robin Scheduling is a CPU scheduling algorithm used by operating systems to schedule tasks or processes preemptively. In Round Robin Scheduling, each process is assigned a fixed time slice or quantum, and the CPU executes each process for the same amount of time. Once the time slice expires, the process is preempted, and the next process in the queue is executed.

Priority Scheduling

Priority Scheduling is a CPU scheduling algorithm in which the process with the highest priority is executed first. Each process is assigned a priority and the scheduler selects the process with the highest priority for execution. Processes with the same priority are executed in a first-come, first-served (FCFS) order.

Shortest Job First

Shortest Job First (SJF) is a scheduling algorithm in which the process with the shortest execution time is scheduled for execution first. This algorithm is non-preemptive, which means that once a process has been scheduled, it is allowed to run until it completes, or it blocks for I/O. The SJF algorithm is designed to minimize the average waiting time of processes and is often used in batch systems.

Multilevel Feedback Queue

A multilevel feedback queue (MLFQ) is a scheduling algorithm that incorporates multiple queues with different priorities and scheduling algorithms. Each queue has a different priority, and each priority is assigned a different scheduling algorithm. The idea behind MLFQ is that different processes have different scheduling needs, and the scheduler should adapt its scheduling algorithm based on the behaviour of the processes.

First Come First Serve

It is among the most basic and easy-to-understand scheduling algorithms. In FCFS, processes are carried out in the order in which they arrive, starting with the process that arrives and ending with it. Envision a line of procedures awaiting the CPU's attention. Upon arrival, a process moves to the back of the queue and is executed by the CPU in the order that it entered the queue. Accordingly, the process that is the first in line receives the CPU first and so on until the queue is cleared.

RESULTS AND DISCUSSION

The most efficient scheduling algorithm depends on the specific requirements and characteristics of the system and workload. There is no best solution, and the choice of algorithm often depends on fairness, response time, and overall system. So no result is deduced from our algorithm