Question & Answers

SCHEDULING

Sercan Külcü | Operating Systems | 10.04.2023

Contents

What is CPU scheduling in the context of operating systems?

CPU scheduling is the process by which the operating system selects a process or thread from the ready queue and allocates CPU time to it. The goal of CPU scheduling is to maximize system throughput, responsiveness, and fairness by efficiently utilizing the available CPU resources.

What is the role of CPU scheduling in maximizing system throughput, responsiveness, and fairness?

The role of CPU scheduling is to ensure that the CPU is always busy executing the highest-priority process or thread, thus maximizing system throughput. It also ensures that the system remains responsive to user requests by quickly switching between processes or threads. Finally, CPU scheduling ensures that system resources are fairly distributed among competing processes or threads.

What are some common CPU scheduling algorithms used in operating systems?

Some common CPU scheduling algorithms used in operating systems include:

- **First-Come, First-Served (FCFS):** This algorithm selects the process/thread that has been in the ready queue for the longest time and allocates the CPU to it.

- **Shortest Job First (SJF):** This algorithm selects the process/thread with the shortest estimated execution time and allocates the CPU to it.

- **Priority Scheduling:** This algorithm assigns a priority to each process/thread and selects the one with the highest priority to allocate the CPU to.

- **Round Robin:** This algorithm allocates a fixed time quantum to each process/thread in turn, allowing them to execute for a specified amount of time before being preempted.

How does the choice of CPU scheduling algorithm impact system performance?

The choice of CPU scheduling algorithm can have a significant impact on system performance. For example, SJF can result in better average turnaround time and response time, but may lead to long waiting times for longer processes/threads. Priority scheduling can ensure that high-priority processes/threads are given precedence, but can lead to lower-priority processes/threads being starved of CPU time.

What is the relationship between CPU scheduling and process/thread management in an operating system?

CPU scheduling is closely related to process/thread management in an operating system, as the scheduler must select a process/thread from the ready queue to allocate the CPU to. The process/thread management system must also keep track of the state of each process/thread and manage the transitions between states.

Briefly explain FCFS?

The term FCFS denotes First Come First Serve. In the scheduling algorithm of FCFS, the job that appears first in the ready queue is given to the CPU for execution, followed by the job that arrived second and so on. FCFS is a non-preemptive scheduling algorithm, meaning that a process will hold the CPU until it terminates or performs I/O. Therefore, if a long job is assigned to the CPU, many shorter jobs after it will have to wait.

What is the RR scheduling algorithm?

The round-robin scheduling algorithm is employed to ensure that processes are scheduled fairly by assigning each job a time slot or quantum and interrupting it if it is not finished by then. Jobs that arrive during the quantum time are placed after the other jobs, making the scheduling fair. Since round-robin is cyclic, starvation does not occur. It is a variant of the first-come, first-served scheduling algorithm, with no priority given to any process or task. Round-robin scheduling is also referred to as time-slicing scheduling.

What is the difference between preemptive and non-preemptive scheduling?

Preemptive scheduling and non-preemptive scheduling are two scheduling techniques used in operating systems. In preemptive scheduling, the CPU is allocated to processes for a limited time, and the executing process is interrupted in the middle of execution when a higher-priority process arrives. On the other hand, in non-preemptive scheduling, the CPU is allocated to the process till it terminates or switches to waiting for state, and the executing process is not interrupted in the middle of execution. Preemptive scheduling incurs overhead in switching processes between the ready and running state and maintaining the ready queue. In comparison, non-preemptive scheduling has no overhead in switching processes between the running and ready state. Preemptive scheduling offers flexibility by allowing critical processes to access the CPU as they arrive in the ready queue, whereas non-preemptive scheduling is considered rigid as the process running CPU is not disturbed. Preemptive scheduling is also cost-associative, as it must maintain the integrity of shared data, whereas this is not the case with non-preemptive scheduling.

What are starvation and aging in OS?

Starvation is a common problem in resource management that can occur in a scheduling system when a process does not receive the resources it needs for a prolonged period. This can happen when the resources are continually being allocated to other processes, leaving the affected process unable to make progress. Starvation can lead to reduced system performance, deadlock, and other issues. To avoid starvation in a scheduling system, a technique called aging can be used. Aging involves adding an aging factor to the priority of each request, which

increases the priority of the request as time passes. This ensures that a request will eventually become the highest priority request and will be serviced by the system, preventing it from being starved of resources.

What is Context Switching?

Context switching is a critical component of modern multitasking operating systems. When the CPU is shared among multiple processes, the operating system must quickly and efficiently switch between executing processes. This process is called context switching. During a context switch, the CPU saves the current state of the process, including the program counter, registers, and other relevant information, to a data structure called the Process Control Block (PCB). The PCB holds all the necessary information about a process, including its current state, memory allocation, and other important metadata. Once the old process's state is saved, the operating system can load the state of the next process and begin executing its instructions. This is an essential function for operating systems and enables the seamless execution of multiple processes concurrently.

What are the goals of CPU scheduling?

CPU scheduling algorithms are designed to optimize the performance of the system in terms of various objectives. Max CPU utilization aims to keep the CPU as busy as possible, utilizing it to its fullest capacity. Fair allocation of CPU is important to ensure that no process monopolizes the CPU, and all processes are given a fair share of CPU time. Max throughput focuses on maximizing the number of processes that complete their execution per time unit. Min turnaround time aims

to minimize the time taken by a process to finish execution, whereas min waiting time focuses on minimizing the time a process spends in the ready queue waiting for CPU time. Finally, min response time targets minimizing the time when a process produces the first response, which is important in interactive systems where users expect quick responses. Different scheduling algorithms prioritize these objectives differently based on the system requirements and workload.

How do operating systems handle priority-based scheduling and preemption?

Operating systems typically implement priority-based scheduling by assigning a priority value to each process or thread. The scheduler then selects the process or thread with the highest priority for execution. If a higher-priority process becomes ready to run, it may preempt the currently running process, meaning the scheduler will interrupt the running process and start executing the higher-priority process instead.

How does the choice of time quantum impact CPU scheduling performance?

The time quantum, also known as the time slice, is the amount of time a process is allowed to run on the CPU before being preempted by the scheduler. A longer time quantum can reduce the overhead of the scheduler, but may decrease system responsiveness. A shorter time quantum can increase system responsiveness, but may increase the overhead of the scheduler.

What is the difference between preemptive and non-preemptive CPU scheduling?

Preemptive CPU scheduling allows a running process to be interrupted and replaced with another process if a higher-priority process becomes ready to run. Non-preemptive CPU scheduling does not allow a running process to be interrupted, and the current process must voluntarily relinquish the CPU.

What are some common techniques used in real-time CPU scheduling, and how do they differ from general-purpose scheduling?

Real-time CPU scheduling typically uses techniques such as earliest deadline first (EDF), rate monotonic scheduling (RMS), and deadline monotonic scheduling (DMS). These techniques prioritize processes based on their deadlines and guarantee that critical processes meet their deadlines. General-purpose scheduling algorithms, on the other hand, focus on maximizing system throughput and responsiveness, without necessarily providing real-time guarantees.

How do operating systems handle CPU scheduling in a multi-core/multi-processor environment?

In a multi-core/multi-processor environment, the operating system must distribute the workload among the available processors. This can be done using techniques such as symmetric multiprocessing (SMP) or

non-uniform memory access (NUMA) scheduling. The scheduler must also consider factors such as cache affinity and processor affinity when making scheduling decisions. Additionally, load balancing algorithms may be used to distribute the workload evenly across all available processors.

What are some advanced techniques for improving the efficiency and fairness of CPU scheduling?

Some advanced techniques for improving CPU scheduling in operating systems include:

- **Multi-level feedback queue scheduling**: This technique assigns processes to different priority levels based on their CPU usage and other factors, allowing for better fairness and responsiveness.

- **Round-robin scheduling with dynamic time quantum**: This approach dynamically adjusts the time quantum given to each process based on its CPU usage and other factors, allowing for more efficient use of system resources.

- **Proportional-share scheduling**: This technique assigns CPU time to processes based on their relative weight, allowing for better resource allocation and fairness.

- **Gang scheduling**: This technique schedules a group of related processes to run simultaneously on different processors, allowing for better utilization of system resources and improved performance.

How do operating systems handle CPU scheduling in distributed computing and cloud computing environments?

In distributed computing and cloud computing environments, CPU scheduling is typically handled by a central scheduler or resource manager that manages resources across multiple nodes or instances. These systems often use a combination of local and global scheduling algorithms to balance workload and optimize resource usage. Additionally, virtualization technologies such as containers and virtual machines can be used to isolate and manage resource usage for individual applications or services.

What is thread-level speculation, and how can it be used to improve CPU scheduling performance?

Thread-level speculation is a technique used to improve CPU scheduling performance by allowing the processor to speculatively execute multiple threads in parallel, based on predicted outcomes. This allows the processor to overlap the execution of multiple threads and better utilize available resources, leading to improved performance and efficiency.

How do operating systems handle dynamic workload changes, and what are some strategies for adapting CPU scheduling to these changes?

Operating systems handle dynamic workload changes by dynamically adjusting CPU scheduling parameters based on current system conditions and workload demands. Some strategies for adapting CPU scheduling to these changes include:

- **Load balancing**: This involves redistributing workloads across multiple processors or nodes to balance resource usage and optimize performance.

- **Adaptive scheduling**: This involves dynamically adjusting scheduling parameters such as time quantum and priority based on workload demands and system conditions.

- **Predictive scheduling**: This involves using historical data and predictive models to anticipate future workload changes and adjust scheduling accordingly.

What are some emerging trends and technologies in CPU scheduling?

Some emerging trends and technologies in CPU scheduling and performance optimization include:

- **Machine learning-based scheduling**: This involves using machine learning algorithms to dynamically adjust scheduling parameters based on workload demands and system conditions.

- **Energy-efficient scheduling**: This involves optimizing CPU scheduling to reduce energy consumption and improve battery life on mobile devices.

- **Neuromorphic computing**: This involves using hardware and software inspired by the structure and function of the brain to improve performance and energy efficiency in CPU scheduling and other areas of computing.