Lecture 5: CPU Scheduling

Adam Hawley

January 25, 2019

Contents

1 Introduction

Most processes exhibit the following behaviour:

- CPU burst followed by I/O burst.
- Process execution consists of a cycle of CPU execution and I/O wait.

Maximum CPU utilisation obtained with multiprogramming. This is where one process uses the CPU while the other waits for I/O. The CPU burst usually takes less than 8ms.

Whenever the CPU becomes idle, OS selects one of the processes in the ready queue to be executed. The selection process is carried out by the CPU scheduler. The ready queue may be ordered in various ways. CPU scheduling decisions may take place when a process:

- switches from running state to waiting state
- switches from running state to ready state
- switches from waiting state to ready state
- terminates

2 Dispatcher

The dispatcher switches context, switches to user mode, jumps to the proper location in the user program to restart that progmram. This brings **dispatch latency** (overhead) — the time it takes for the dispatcher to stop one process and start another running. This latency needs to be minimised as occurs frequently.

3 Scheduling Algorithms

Existing algorithms aim to satisfy the following criteria:

- **Max CPU utilisation** — keep the CPU as busy as possible.
- **Max throughput** — number of processesthat complete their execution per time unit.
- **Min turnaround time** — amount of time to execute a particular process.
- **Min waiting time** — total amount of time a process has been waiting in the ready queue.
- **Min response time** — amount of time it takes from when a requestwas submitted until the first response is produced (for time-sharing environment).

- **Fairness** — each process should get its fair share of the CPU.

There are three main families of scheduling algorithms:

- **Non-preemptive** — Process runs to completion, OS schedules next process upon termination.
- **Preemptive** — Process runs until suspended by OS. Preemption can be time or event triggered.
- **Cooperative** — Process explicitly relinquishes control and allows OS to make a scheduling decision.

4 Classic Scheduling Algorithms

The three classic sheduling algorithms are:

- FCFS (or FIFO) — first come, first served. **Non-Preemptive**
- Round-Robin (RR) — aims for fairness. **Preemptive**
- Priority Scheduling — **Preemptive or Non-Preemptive**

Or it is possible to have combinations of the above.

4.1 FCFS

Easy to implement and maintain but can give a wide variety of average wait times.

4.2 RR

Each process is assigned a *quantum* when it enters the CPU (a quantum is a time slot using the CPU). If the process is still running at the end of the quantum then preempt it:

- Start a context switch
- The process is put at the back of the "runnable queue"

Process may become no longer runnable **before** the end of the quantum e.g I/O operation. In which case, do as though it was the end of the quantum.

Overall Round-Robin ensures fairness, but does not reflect relative importance of processes.

4.3 Priority Scheduling

Not all processes are equally important, so priority scheduling assigns a priority for each of them.

Note: Priority values vary between books, courses etc. In this course, higher priority processes have lower priority numbers (e.g 1 or 0 = highest priority).

In the preemptive version, the runnable process with the highest priority is allowed to run immediately. Otherwise, the highest priority process may have to wait for an ongoing process to finish. This algorithm has one large problem and that is that the highest priority process may run indefinitely or to completion and hence, may lead to starvation for other processes. Therefore the OS should have a way of identifying and recovering from such scenarios (e.g aging — increase the priority of the process as time progresses).

4.4 Comparison of Classic Algorithms

	Pros	Cons
FCFS	Simple & Low overheads	Bad turnaround for short processes
Round-Robin	Good turnaround for short processes	High Overhead
Priorities	Predictability for high priority tasks	May delay lower priority tasks for long periods

4.5 Bonus Algorithm SJF

Shortest-Job-First associates the length of each process' next CPU burst with the process itself. Then we use these lengths to schedule the process with the shortest time. This is non-preemptive. SJF is optimal in that it gives the minimum average waiting time for a given set of processes (assuming that the estimation of the next CPU burst time is correct.

Why don't we always use SJF? The burst sizes can be unpredictable, they may be dependent on input sizes.