Module 04: CPU Scheduling

1

Wherein we consider this…

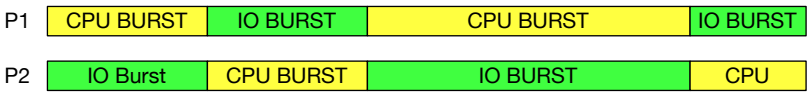
- What process gets the CPU when.
  - When does the kernel get it vs. user processes?
  - Which user processes get it when
  - What about threads inside the same process?
  - What about multiple core / multiple processor systems?
- What about…
  - Deadlocks
  - Livelocks
  - Starvation

2

In simplistic computing systems..

- In simplistic computing systems, there is one processor, one user, and very likely one program/process.
- When running, the program/process would “waste time” during I/O bursts

- Wasting CPU is fine in simple systems, but we generally want to do better one way to do this would be to multi-program multiple processes onto a CPU. In a perfect world, it would look like this…



3

In simplistic computing systems..

- Of course, we’ll never get perfect…. So we try to ”schedule processes threads to CPUs in whatever way we can to maximize the amount of time the CPU (CPUs) is (are) actually busy.
- One BIG problem is that the lengths of CPU and IO bursting is ultimately unpredictable.
- On average, however, CPU burst times look “log-normal” (the book calls this exponential)

4

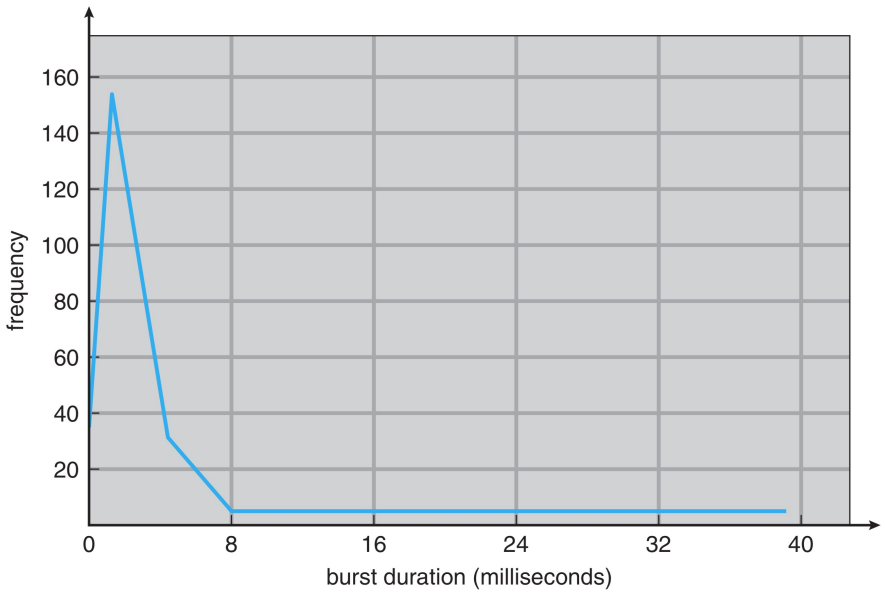


Figure 6.2 Histogram of CPU-burst durations.

5

CPU vs IO Bound Processes

- I/O-bound processes are characterized by many short CPU bursts punctuated with much longer I/O requests. GUIs are an example.
- CPU-bound processes are characterized by having fewer and longer CPU bursts. Computationally intensive simulations are an example.

6

Process State Reminder…

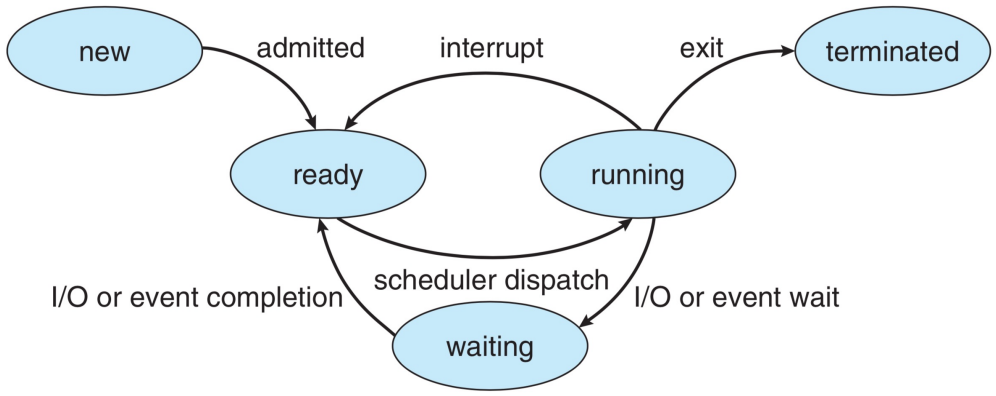


Figure 3.2 Diagram of process state.

7

Each of these states has a “queue”, but of “what kind”

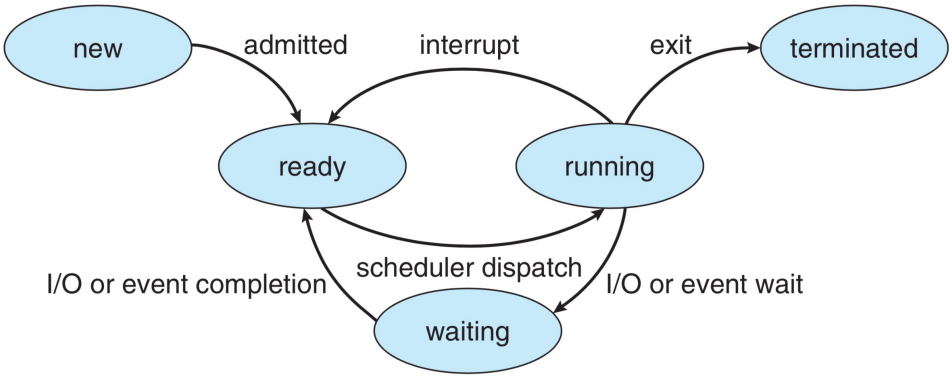


Figure 3.2 Diagram of process state.

8

CPU Scheduler

- "Some guy who decides when something comes off the ready queue and goes to a processor"
- The process state queues are not necessarily FIFOs
- The CPU Scheduler decides when and how process control block (PCBs) move from queue to queue.
- CPU schedulers COULD be FIFOs, priority queues, search trees, or any data structure that efficiently implements whatever scheduling policy is desired.

9

CPU Scheduling Events

CPU scheduling decisions take place under the following four circumstances

1. When a process switches from running to waiting as the result of an I/O request or as the result of waiting on a child process to end
2. When a process switches from running to ready (E.G. an interrupt)
3. When a process switches from waiting to ready (E.G. completion of I/O)
4. When a process terminates

One of these thing is no like the others… one of them is "special" and occurs only in one kind of kernel. Which one is it?

10

CPU Scheduling Events

CPU scheduling decisions take place under the following four circumstances

1. When a process switches from running to waiting as the result of an I/O request or as the result of waiting on a child process to end
2. When a process switches from running to ready (E.G. an interrupt)
3. When a process switches from waiting to ready (E.G. completion of I/O)
4. When a process terminates

If ONLY #1, #3, and #4 occur, it is called a *nonpreemptive* or *cooperative* scheduler. This means that processes only come off of the CPU when they decide to. Yes, it would be possible for a process to grab a CPU and just stay there forever. Processes that are NOT running automatically would go into a single ready queue.

IF #2 (or something like it) exists, then the scheduler is preemptive. This means that other event can kick a process off of a CPU. The event could be a timer (E.G. each process can keep a CPU only for a bit before it has to get off and wait in line again).

11

CPU Scheduling Events

CPU scheduling decisions take place under the following four circumstances

1. When a process switches from running to waiting as the result of an I/O request or as the result of waiting on a child process to end
2. When a process switches from running to ready (E.G. an interrupt)
3. When a process switches from waiting to ready (E.G. completion of I/O)
4. When a process terminates

Preemptive schedulers are more difficult to write than cooperative schedulers. Why is this?

12

The Dispatcher

- The *dispatcher* is the kernel module that gives control of a CPU to a specific process
  - It switches context
  - It initiates user mode
  - It transfer control to the proper location in the user process
- The *dispatcher* MUST be fast. The amount of time it takes for the dispatcher to do its job is called *dispatch latency.*

13

Scheduling Criteria

In order to design a scheduler, one needs to designate *scheduling criteria* that define what good scheduling even is. It shouldn't be surprising that there are many situationally acceptable criteria for what "good scheduling" even means. Sometimes multiple criteria may be used together or at different scales (long term vs. short term scheduling).

14

Scheduling Criteria

CPU Utilization: The amount of time that a CPU is actually doing work. Ideally one would want each CPU operating 100% of the time. This is a measure of resource utilization.

Throughput: The number of processes that are completed per unit time. This is a measure of the the rate that we're pushing work through the system.

15

Scheduling Criteria

Turnaround Time: The amount of time it takes from process submission to the new queue to its exit from the termination queue. This is a measure of how long it takes to get a single job done.

Waiting Time: The amount of time a process spends in the wait queue. This is a measure of how good a kernel is at moving work to the CPU.

Response Time: The amount of time it takes for the first I/O to occur. This factors out the I/O times that turnaround time count.

16

Scheduling Criteria

Generally, we try to build schedulers that

MAXIMIZE: CPU utilization and throughput

MINIMIZE: turnaround time, waiting time, and response time

17

Scheduling Algorithms

First Come First Served: Processes are moved from ready to running in the order they entered ready (FIFO)

average waiting time can be LONG

If the processes arrive in the order P_1, P_2, P_3, and are served in FCFS order, we get the result shown in the following Gantt chart, which is a bar chart that illustrates a particular schedule, including the start and finish times of each of the participating processes:

The waiting time is 0 milliseconds for process P_1, 24 milliseconds for process P_2, and 27 milliseconds for process P_3. Thus, the average waiting time is (0 + 24 + 27)/3 = 17 milliseconds. If the processes arrive in the order P_2, P_3, P_1, however, the results will be as shown in the following Gantt chart:

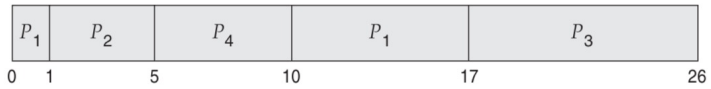
The average waiting time is now (6 + 0 + 3)/3 = 3 milliseconds. This reduction is substantial. Thus, the average waiting time under an FCFS policy is generally not minimal and may vary substantially if the processes' CPU burst times vary greatly.

18

Scheduling Algorithms

Shortest Job First (SJF): Processes are moved from wait to run in order of shortest NEXT CPU burst

Algorithm is provably optimal in terms of minimizing average wait time...



… but we don't REALLY know the length of the next CPU burst for a process…. So what do we do?

19

Shortest Job First

Shortest Job First: Processes are moved from wait to run in order of shortest NEXT CPU burst

We generally keep statistics on the a history of burst times and estimate the expected next burst time of each CPU burst. Could this starve?

$$\tau_{n+1} = \alpha\, t_n + (1 - \alpha)\tau_n.$$

$$\tau_{n+1} = \alpha t_n + (1 - \alpha)\alpha t_{n-1} + \cdots + (1 - \alpha)^j \alpha t_{n-j} + \cdots + (1 - \alpha)^{n+1}\tau_0.$$

20

Priority Scheduling

Priority Scheduling: Each process gets a "priority score". It comes out of the wait queue in priority order.

SJF is a special case of priority where priority is set by shortness of next burst

Can this starve?

How could we fix the starvation?

21

Round Robin Scheduling

Round Robin Scheduling (RR): In its most basic form, it's FCFS (FIFO) scheduling with preemption. Each process that moves to a CPU may keep the CPU for its whole CPU burst OR a set "quantum" of time, whichever happens first. As soon as a process surrenders or is kicked off a CPU, it goes to the back of the wait queue.

Can this starve?

22

Round Robin Scheduling (Advanced)

Round Robin Scheduling (RR): In its advanced form, RR scheduling is a priority scheduler with preemption. Each process that moves to a CPU may keep the CPU for its whole CPU burst OR a set "quantum" of time, whichever happens first. As soon as a process surrenders or is kicked off a CPU, it goes to into the wait queue. Processes come OFF the wait queue in priority order.

Can this starve?

23