CS370 Operating Systems
Colorado State University
Yashwant K Malaiya
Fall 2017 Lecture 9

Slides based on
• Text by Silberschatz, Galvin, Gagne
• Various sources
CPU Scheduling: Objectives

- CPU scheduling, the basis for multiprogrammed operating systems
- CPU-scheduling algorithms
- Evaluation criteria for selecting a CPU-scheduling algorithm for a particular system
- Scheduling algorithms of several operating systems
Chapter 6: CPU Scheduling

- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
- Thread Scheduling
- Multiple-Processor Scheduling
- Real-Time CPU Scheduling
- Operating Systems Examples
- Algorithm Evaluation
Diagram of Process State

- **Ready to Running**: scheduled by scheduler
- **Running to Ready**: scheduler picks another process, back in ready queue
- **Running to Waiting (Blocked)**: process blocks for input/output
- **Waiting to Ready**: Input available
Process Control Block (PCB)

Information associated with each process (also called task control block)

- Process state – running, waiting, etc
- Program counter – location of instruction to next execute
- CPU registers – contents of all process-centric registers
- CPU scheduling information – priorities, scheduling queue pointers
- Memory-management information – memory allocated to the process
- Accounting information – CPU used, clock time elapsed since start, time limits
- I/O status information – I/O devices allocated to process, list of open files
CPU Switch From Process to Process

process $P_0$  operating system  process $P_1$

executing  

interrupt or system call

save state into PCB$_0$

idle

interrupt or system call

reload state from PCB$_1$

executing

save state into PCB$_1$

idle

reload state from PCB$_0$
Basic Concepts

- Maximum CPU utilization obtained with multiprogramming
- CPU–I/O Burst Cycle – Process execution consists of a cycle of CPU execution and I/O wait
- **CPU burst** followed by I/O burst
- CPU burst distribution is of main concern
Histogram of CPU-burst Times
CPU Scheduler

- **Short-term scheduler** selects from among the processes in ready queue, and allocates the CPU to one of them
  - Queue may be ordered in various ways
- CPU scheduling decisions may take place when a process:
  1. Switches from running to waiting state
  2. Switches from running to ready state
  3. Switches from waiting to ready
  4. Terminates
- Scheduling under 1 and 4 is **nonpreemptive**
- All other scheduling is **preemptive. These need to be considered**
  - access to shared data by multiple processes
  - preemption while in kernel mode
  - interrupts occurring during crucial OS activities
Dispatcher

• Dispatcher module gives control of the CPU to the process selected by the short-term scheduler; this involves:
  – switching context
  – switching to user mode
  – jumping to the proper location in the user program to restart that program

• Dispatch latency – time it takes for the dispatcher to stop one process and start another running
Scheduling Criteria

- **CPU utilization** – keep the CPU as busy as possible: **Maximize**
- **Throughput** – # of processes that complete their execution per time unit: **Maximize**
- **Turnaround time** – time to execute a process from submission to completion: **Minimize**
- **Waiting time** – amount of time a process has been waiting in the ready queue: **Minimize**
- **Response time** – time it takes from when a request was submitted until the first response is produced, not output (for time-sharing environment): **Minimize**
Terms for a single process

- command arrives
- command begins running
- the first output of command appears
- command finishes executing

- time
- wait time
- response time
- execution time
- turnaround time
We will now examine several major scheduling approaches.

Decides which process in the ready queue is allocated the CPU.

Could be preemptive or nonpreemptive:
- Preemptive: remove in middle of execution.

Optimize measure of interest:
- We will use Gantt charts to illustrate schedules.
- Bar chart with start and finish times for processes.
Nonpreemptive vs Preemptive scheduling

• **Nonpreemptive:** Process keeps CPU until it relinquishes it when
  – It terminates
  – It switches to the waiting state
  – Used by initial versions of OSs like Windows 3.x

• **Preemptive** scheduling
  – Pick a process and let it run for a maximum of some fixed time
  – If it is still running at the end of time interval?
    • Suspend it and pick another process to run

• **A clock interrupt** at the end of the time interval to give control back of CPU back to scheduler
 Algorithms

- First-Come, First-Served (FCFS)
- Shortest-Job-First (SJF)
  - Shortest-remaining-time-first
- Priority Scheduling
- Round Robin (RR) with time quantum
- Multilevel Queue
  - Multilevel Feedback Queue

Comparing Performance

- Average waiting time etc.
First- Come, First-Served (FCFS) Scheduling

• Process requesting CPU first, gets it first
• Managed with a FIFO queue
  – When process enters ready queue
    • PCB is tacked to the tail of the queue
  – When CPU is free
    • It is allocated to process at the head of the queue
• Simple to write and understand
First- Come, First-Served (FCFS) Scheduling

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>24</td>
</tr>
<tr>
<td>$P_2$</td>
<td>3</td>
</tr>
<tr>
<td>$P_3$</td>
<td>3</td>
</tr>
</tbody>
</table>

• Suppose that the processes arrive in the order: $P_1$, $P_2$, $P_3$ but almost the same time.

The Gantt Chart for the schedule is:

- Waiting time for $P_1 = 0$; $P_2 = 24$; $P_3 = 27$
- Average waiting time: $(0 + 24 + 27)/3 = 17$
- Throughput: $3/30 = 0.1$ per unit
Suppose that the processes arrive in the order: $P_2, P_3, P_1$

- The Gantt chart for the schedule is:

<table>
<thead>
<tr>
<th>P_2</th>
<th>P_3</th>
<th>P_1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>6</td>
</tr>
</tbody>
</table>

- Waiting time for $P_1 = 6$; $P_2 = 0$; $P_3 = 3$
- Average waiting time: $(6 + 0 + 3)/3 = 3$
- Much better than previous case
- But note - Throughput: $3/30 = 0.1$ per unit
- **Convoy effect** - short process behind long process
  - Consider one CPU-bound and many I/O-bound processes
Shortest-Job-First (SJF) Scheduling

• Associate with each process the length of its next CPU burst
  – Use these lengths to schedule the process with the shortest time

• Reduction in waiting time for short process GREATER THAN Increase in waiting time for long process

• SJF is optimal – gives minimum average waiting time for a given set of processes
  – The difficulty is knowing the length of the next CPU request
  – Could ask the user
Example of SJF

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>6</td>
</tr>
<tr>
<td>$P_2$</td>
<td>8</td>
</tr>
<tr>
<td>$P_3$</td>
<td>7</td>
</tr>
<tr>
<td>$P_4$</td>
<td>3</td>
</tr>
</tbody>
</table>

- All arrive at time 0.
- SJF scheduling chart

- Average waiting time for $P_1, P_2, P_3, P_4 = (3 + 16 + 9 + 0) / 4 = 7$