Chapter 5: CPU Scheduling
Logistics

- Hwk 2 due today.
- Hwk 3 posted today, due in one week.

- Questions on PA1?
  - I have heard rumors of issues on cygwin+gcc.
  - I will try to verify this myself. In the meantime, I highly recommend people try to use the departmental machines.
    - Solaris systems like ix
    - MacOSX systems like rm. 100 lab
  - Note that you can SSH in from anywhere to the Macs in the lab. You don’t need to physically be sitting there.
Chapter 5: CPU Scheduling

- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
- Thread Scheduling
- Multiple-Processor Scheduling
- Operating Systems Examples
- Algorithm Evaluation
Objectives

- To introduce CPU scheduling, which is the basis for multiprogrammed operating systems
- To describe various CPU-scheduling algorithms
- To discuss evaluation criteria for selecting a CPU-scheduling algorithm for a particular system
Purpose of Scheduling

- Given a set of processes, who runs when?

- Decision is driven by what performance characteristics you want the system to exhibit.

- Remember that multiprogramming was invented to increase utilization.
  - Utilization = % of time CPU is busy

- Unfortunately, the ability to multitask alone is insufficient to achieve optimal utilization.
  - To do so, we need to worry about when different processes are allowed to run. *How* we multitask is critical to achieving good utilization.
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 distribution

What does this say?
- Most bursts are short.
Alternating Sequence of CPU And I/O Bursts

- load store
- add store
- read from file

- store increment index
- write to file

- wait for I/O

- CPU burst
- I/O burst

- load store
- add store
- read from file

- wait for I/O

- CPU burst
- I/O burst

- wait for I/O

- I/O burst
CPU Scheduler

- Selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them.

- 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
Preemption / Nonpreemption

- **Preemption**
  - A process running on the CPU is paused and another process takes its place on the CPU.
  - This is involuntary with respect to the process being preempted.
  - A context switch occurs to make this happen.

- **Nonpreemption**
  - A process gives up its presence on the CPU voluntarily.
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

*Recall: context switch is when the state of the running process is stored safely in its PCB, and the state from another process PCB is placed on the CPU.*
Scheduling Criteria

- **CPU utilization** – keep the CPU as busy as possible
- **Throughput** – # of processes that complete their execution per time unit
- **Turnaround time** – amount of time to execute a particular process
- **Waiting time** – amount of time a process has been waiting in the ready queue
- **Response time** – amount of time it takes from when a request was submitted until the first response is produced, not output (for time-sharing environment)
Finding a good scheduler requires balancing these criteria.

- Maximize CPU utilization
- Maximize throughput
- Minimize turnaround time
- Minimize waiting time
- Minimize response time
Start simple

- What is the simplest scheduler?

- First come, first served.

- Very little required in decision making here – just a queue.

- Let’s see how this performs.
First-Come, First-Served (FCFS) Scheduling

**Process** | **Burst Time**
---|---
\( P_1 \) | 24
\( P_2 \) | 3
\( P_3 \) | 3

- Suppose that the processes arrive in the order: \( P_1, P_2, P_3 \)

The Gantt Chart for the schedule is:

```
    P_1 | P_2 | P_3
0    24  27  30
```

- Waiting time for \( P_1 = 0; P_2 = 24; P_3 = 27 \)
- Average waiting time: \( (0 + 24 + 27)/3 = 17 \)
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></th>
<th>P_2</th>
<th>P_3</th>
<th>P_1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td>30</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></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
- Convoy effect: short process behind long process
What happened?

- FCFS doesn’t minimize average waiting time.
- Highly dependent on order that jobs arrive.
  - Not a big deal if all jobs have the same length though.

- This is not ideal.

- What if we start to take job length into account when we make scheduling decisions?

- Start with Shortest Job First.
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.

- 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.
### Example of SJF

#### Process Arrival Time Burst Time

<table>
<thead>
<tr>
<th>Process</th>
<th>Arrival Time</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>0.0</td>
<td>6</td>
</tr>
<tr>
<td>$P_2$</td>
<td>2.0</td>
<td>8</td>
</tr>
<tr>
<td>$P_3$</td>
<td>4.0</td>
<td>7</td>
</tr>
<tr>
<td>$P_4$</td>
<td>5.0</td>
<td>3</td>
</tr>
</tbody>
</table>

- SJF scheduling chart

- Average waiting time = \( \frac{3 + 16 + 9 + 0}{4} = 7 \)
Determining Length of Next CPU Burst

- Can only estimate the length
- Can be done by using the length of previous CPU bursts, using exponential averaging

1. \( t_n \) = actual length of \( n^{th} \) CPU burst
2. \( \tau_{n+1} \) = predicted value for the next CPU burst
3. \( \alpha, 0 \leq \alpha \leq 1 \)
4. Define: \( \tau_{n+1} = \alpha t_n + (1 - \alpha) \tau_n \).
Prediction of the Length of the Next CPU Burst

CPU burst \( (t_i) \)  |  6  |  4  |  6  |  4  |  13 |  13 |  13 |  ... \\
"guess" \( (\tau_i) \)  |  10 |  8  |  6  |  6  |  5  |  9  |  11 |  12 |  ...
Examples of Exponential Averaging

- \( \alpha = 0 \)
  - \( \tau_{n+1} = \tau_n \)
  - Recent history does not count

- \( \alpha = 1 \)
  - \( \tau_{n+1} = \alpha \ t_n \)
  - Only the actual last CPU burst counts

- If we expand the formula, we get:
  \[
  \tau_{n+1} = \alpha \ t_n + (1 - \alpha) \alpha \ t_{n-1} + \ldots + (1 - \alpha)^{j} \alpha \ t_{n-j} + \ldots + (1 - \alpha)^{n+1} \tau_0
  \]

- Since both \( \alpha \) and \( 1 - \alpha \) are less than or equal to 1, each successive term has less weight than its predecessor
Priority Scheduling

- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority (smallest integer $\equiv$ highest priority)
  - Preemptive
  - Nonpreemptive

- SJF is a priority scheduling where priority is the predicted next CPU burst time

- Problem $\equiv$ Starvation – low priority processes may never execute
- Solution $\equiv$ Aging – as time progresses increase the priority of the process
Round Robin (RR)

- Each process gets a small unit of CPU time (time quantum), usually 10-100 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue.

- If there are $n$ processes in the ready queue and the time quantum is $q$, then each process gets $1/n$ of the CPU time in chunks of at most $q$ time units at once. No process waits more than $(n-1)q$ time units.

- Performance
  - $q$ large $\Rightarrow$ FIFO
  - $q$ small $\Rightarrow$ $q$ must be large with respect to context switch, otherwise overhead is too high
Example of RR with Time Quantum = 4

<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>

- The Gantt chart is:

```
<table>
<thead>
<tr>
<th>0</th>
<th>4</th>
<th>7</th>
<th>10</th>
<th>14</th>
<th>18</th>
<th>22</th>
<th>26</th>
<th>30</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>$P_2$</td>
<td>$P_3$</td>
<td>$P_1$</td>
<td>$P_1$</td>
<td>$P_1$</td>
<td>$P_1$</td>
<td>$P_1$</td>
<td></td>
</tr>
</tbody>
</table>
```

- Typically, higher average turnaround than SJF, but better *response*
  - Turnaround: time to complete; Response: time to service.
Time Quantum and Context Switch Time

process time = 10

<table>
<thead>
<tr>
<th>quantum</th>
<th>context switches</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>9</td>
</tr>
</tbody>
</table>
Turnaround Time Varies With The Time Quantum

<table>
<thead>
<tr>
<th>process</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P₃</td>
<td>1</td>
</tr>
<tr>
<td>P₂</td>
<td>3</td>
</tr>
<tr>
<td>P₄</td>
<td>7</td>
</tr>
<tr>
<td>P₁</td>
<td>6</td>
</tr>
</tbody>
</table>

![Graph showing the relationship between average turnaround time and time quantum](image)