CPU Scheduling

- Introduce CPU scheduling
- Describe various CPU scheduling algorithms
- Discuss evaluation criteria of CPU scheduling algorithms

Agenda

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

Basic Concepts

- Maximum CPU utilization obtained with multiprogramming
- CPU–I/O Burst Cycle – Process execution consists of
  + cycle of CPU execution, and
  + I/O wait
- CPU burst distribution

Alternating Sequence of CPU And I/O Bursts

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 (task arrives).
  4. Terminates.

- Scheduling under 1 and 4 is nonpreemptive.

- All other scheduling is preemptive.
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 – quotient of time the CPU is busy and the observation period
- Throughput – number of processes that complete their execution per time unit
- Turnaround time – the time between job submit time and job completion time.
- 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)

Optimization Criteria

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

FCFS Scheduling

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>0</th>
<th>5</th>
<th>10</th>
<th>15</th>
<th>20</th>
<th>25</th>
<th>30</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P_2 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( P_3 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( P_1 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Waiting time for \( P_1 \) = 0ms; \( P_2 \) = 0ms; \( P_3 \) = 3ms
- Average waiting time: \( (0 + 0 + 3)/3 = 1ms \)
- Much better than previous case.

Convoy effects: short process behind long process

First-Come, First-Served (FCFS) Scheduling

Example:
- Process \( P_1 \), \( P_2 \), \( P_3 \)

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst Time (ms)</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: \( P_1, P_2, P_3 \)

The Gantt chart for the schedule is:

\[ \begin{array}{ccc}
0 & 24 & 27 \\
| P_1 | P_2 | P_3 |
\end{array} \]

- Waiting time for \( P_1 \) = 6ms; \( P_2 \) = 24; \( P_3 \) = 27ms
- Average waiting time: \( (6 + 24 + 27)/3 = 17ms \)

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.
- Two schemes:
  - Nonpreemptive – once CPU given to the process it cannot be preempted until it completes its CPU burst.
  - Preemptive – if a new process arrives with CPU burst length less than remaining time of current executing process, preempt.
    Shortest-Remaining-Time-First (SRTF).
- SJF is optimal – gives minimum average waiting time for a given set of processes.
**SJF (nonpreemptive)**

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>P₁</td>
<td>6</td>
</tr>
<tr>
<td>P₂</td>
<td>8</td>
</tr>
<tr>
<td>P₃</td>
<td>7</td>
</tr>
<tr>
<td>P₄</td>
<td>3</td>
</tr>
</tbody>
</table>

Average waiting time = (3 + 16 + 9 + 0)/4 = 7 ms

**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. \( t_i \) = actual length of \( n^\text{th} \) CPU burst
2. \( \tau_{n+1} \) = predicted value for the next CPU burst
3. \( \alpha, 0 \leq \alpha \leq 1 \)
4. Define:

\[
\tau_{n+1} = \alpha t_i + (1 - \alpha)\tau_n
\]

**Examples of Exponential Averaging**

- \( \alpha = 0 \)
  
  + \( t_{n+1} = t_n \)
  
  + Recent history does not count.

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

- If we expand the formula, we get:

\[
\tau_{n+1} = \alpha t_n + (1 - \alpha)\alpha t_{n-1} + \ldots + (1 - \alpha)^j \alpha \tau_0 + \ldots + (1 - \alpha)^{n+1} \tau_0
\]

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

**SJF (preemptive)**

<table>
<thead>
<tr>
<th>Process</th>
<th>Arrival Time (ms)</th>
<th>Burst Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>P₁</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>P₂</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>P₃</td>
<td>2</td>
<td>9</td>
</tr>
<tr>
<td>P₄</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>

**Priority Scheduling**

- A priority number (integer) is associated with each process
- Priorities
  
  + Internal (time limits, memory requirement, ...)
  
  + External (process importance, political/economical factors, ...)
Priority Scheduling (Cont.)

- The CPU is allocated to the process with the highest priority (smallest integer = highest priority).
  - Preemptive
  - Nonpreemptive

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

- Problem: Starvation – low priority processes may never execute.

  Solution: Aging – as time progresses increase the priority of the process.

Round Robin (RR)

- Each process gets a small unit of CPU time (time quantum, time slice), 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 $\Rightarrow$ Processor sharing (virtually $n$ processes running at the speed of $1/n$)
  Observe that $q$ must be large with respect to context switch, otherwise overhead is too high.

Example of RR with Time Quantum=20

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>53</td>
</tr>
<tr>
<td>$P_2$</td>
<td>17</td>
</tr>
<tr>
<td>$P_3$</td>
<td>68</td>
</tr>
<tr>
<td>$P_4$</td>
<td>24</td>
</tr>
</tbody>
</table>

- The Gantt chart is:

  \[ \begin{array}{cccccccccccc}
  & P_1 & P_2 & P_3 & P_4 & P_1 & P_3 & P_4 & P_1 & P_3 & P_3 & P_4 \\
 0 & 20 & 37 & 57 & 77 & 97 & 117 & 134 & 154 & 162 \\
  \end{array} \]

  - RR has typically higher average turnaround than SJF, but better response.

Turnaround Time Varies with The Time Quantum

- Ready queue is partitioned into separate queues:
  - foreground (interactive)
  - background (batch)

- Each queue has its own scheduling algorithm
  - foreground – RR
  - background – FCFS

- Scheduling must be done between the queues.
  - Fixed priority scheduling; i.e., serve all from foreground then from background. Possibility of starvation.
  - Time slice – each queue gets a certain amount of CPU time which it can schedule amongst its processes; i.e.,
    - 80% to foreground in RR
    - 20% to background in FCFS

Multilevel Queue

- Multilevel queue system: provides a set of priorities to classify processes into different levels of queues.

- Each priority level has its own scheduling algorithm.
  - Each process has a fixed priority that ranges from 0 to 1.

- Each queue has its own scheduling algorithm:
  - foreground – RR
  - background – FCFS

- Scheduling must be done between the queues.
  - Fixed priority scheduling: i.e., serve all from foreground then from background. Possibility of starvation.
  - Time slice – each queue gets a certain amount of CPU time which it can schedule amongst its processes; i.e.,
    - 80% to foreground in RR
    - 20% to background in FCFS
Example of Multilevel Queue

- A process can move between the various queues; aging can be implemented this way.
- Multilevel-feedback-queue scheduler defined by the following parameters:
  - number of queues
  - scheduling algorithms for each queue
  - method used to determine when to upgrade a process
  - method used to determine when to demote a process
  - method used to determine which queue the process will enter when it needs service.

Multilevel Feedback Queue

Example of Multilevel Feedback Queue

- Three queues:
  - Q0 – RR with time quantum 8 ms
  - Q1 – RR time quantum 16 ms
  - Q2 – FCFS

- Scheduling:
  - A new job enters queue Q0 which is served RR.
    - When it gains CPU, job receives 8 ms.
    - If it does not finish in 8 ms, job is moved to queue Q1.
  - At Q1, job is again served RR and receives 16 additional ms.
  - If it still does not complete, it is preempted and moved to queue Q2 served in FCFS.

Multiple-Processor Scheduling

- CPU scheduling more complex when multiple CPUs are available.
- Homogeneous processors within a multiprocessor.
- Asymmetric multiprocessing – only one processor accesses the system data structures, alleviating the need for data sharing.
- Symmetric multiprocessing (SMP) – each processor is self-scheduling.
- Processor affinity
- Load sharing/balancing
- Symmetric multithreading (SMT)

Thread Scheduling

- User-level and kernel-level threads
- Contention scope:
  - Process-contention scope (PCS) – local
    The threads library decides which thread to put onto an available LWP.
  - System-contention scope (SCS) – global
    Kernel decides which kernel thread to run next.

Solaris Scheduling

- Processors:
  - Idle quantum
  - Ready quantum
  - Time quantum
  - Stack quantum
- Threads:
  - User-level
  - Kernel-level
  - Dispatch table for interactive and time-sharing threads.
Windows XP: Priority-based Scheduling

- Priority-based
  - Variable class (priorities 1-15)
  - Real-time class (priorities 15-32)
  - Idle thread (priority 0): memory management

<table>
<thead>
<tr>
<th>Task class</th>
<th>Critical</th>
<th>High</th>
<th>Above normal</th>
<th>Normal</th>
<th>Below normal</th>
<th>Idle</th>
</tr>
</thead>
<tbody>
<tr>
<td>time-critical</td>
<td>31</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>highest</td>
<td>26</td>
<td>15</td>
<td>12</td>
<td>10</td>
<td>9</td>
<td>6</td>
</tr>
<tr>
<td>above normal</td>
<td>20</td>
<td>14</td>
<td>11</td>
<td>9</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>normal</td>
<td>24</td>
<td>13</td>
<td>10</td>
<td>8</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>below normal</td>
<td>23</td>
<td>12</td>
<td>9</td>
<td>7</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>lowest</td>
<td>22</td>
<td>11</td>
<td>8</td>
<td>6</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>idle</td>
<td>16</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Linux Scheduling

- Preemptive, priority based scheduling with two priority ranges:
  - Real-time (0-99)
  - Other (100-140)
- Complexity: O(1)

Algorithm Evaluation

- Deterministic modeling:
  Takes a particular predetermined workload and defines the performance of each algorithm for that workload.
- Queueing models – mathematical formulation
- Simulation models – based on system trace
- Implementation – evaluation in real situation

Recommended Reading and Exercises

- Reading:
  - Chapter 5 (SGG7)
  - Chapter 6 (sixth edition)
- Exercises:
  - All
- No project in Chapter 5