Chapter 6: CPU Scheduling

- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
- Multiple-Processor Scheduling
- Real-Time Scheduling
- Algorithm Evaluation
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
Alternating Sequence of CPU And I/O Bursts

```
... load store
    add store
    read from file
    wait for I/O
    store increment
    index
    write to file
    wait for I/O
    load store
    add store
    read from file
    wait for I/O
    ... CPU burst
    I/O burst
    CPU burst
    I/O burst
    CPU burst
    I/O burst
    ...```

Operating System Concepts
Histogram of CPU-burst Times
CPU (Short-term) 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
  - Process retains CPU until it releases it
  - Windows 3.1, MAC OS
- All other scheduling is preemptive.
Issues with Preemptive Scheduling

- New mechanisms needed to ensure shared data is not in an inconsistent state (partially updated)
- System calls may change important kernel parameters
  - What happens if process preempted
- Unix (most versions) wait for system call to complete or i/o block to take place
- Also interrupts must be guarded from simultaneous use
  - Interrupts disabled at entry, reenabled at exit
- These are bad features for real time or multiprocessor systems
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
- 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)
Optimization Criteria

- Max CPU utilization
- Max throughput
- Min turnaround time
- Min waiting time
- Min response time
- In theory minimize variance in response time
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: $P_1, P_2, P_3$

The Gantt Chart for the schedule is:

- Waiting time for $P_1 = 0; P_2 = 24; P_3 = 27$
- Average waiting time: \( (0 + 24 + 27)/3 = 17 \)
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>
<tr>
<td>30</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
Shortest-Job-First (SJR) 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 completes its CPU burst.
  - preemptive — if a new process arrives with CPU burst length less than remaining time of current executing process, preempt. This scheme is known as the Shortest-Remaining-Time-First (SRTF).

- SJF is optimal — gives minimum average waiting time for a given set of processes.
Example of Non-Preemptive SJF

<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>7</td>
</tr>
<tr>
<td>$P_2$</td>
<td>2.0</td>
<td>4</td>
</tr>
<tr>
<td>$P_3$</td>
<td>4.0</td>
<td>1</td>
</tr>
<tr>
<td>$P_4$</td>
<td>5.0</td>
<td>4</td>
</tr>
</tbody>
</table>

- SJF (non-preemptive)

- Average waiting time = $(0 + 6 + 3 + 7)/4 = 4$
### Example of Preemptive SJF

<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>7</td>
</tr>
<tr>
<td>$P_2$</td>
<td>2.0</td>
<td>4</td>
</tr>
<tr>
<td>$P_3$</td>
<td>4.0</td>
<td>1</td>
</tr>
<tr>
<td>$P_4$</td>
<td>5.0</td>
<td>4</td>
</tr>
</tbody>
</table>

- **SJF (preemptive)**

- Average waiting time $= \frac{9 + 1 + 0 + 2}{4} = 3$
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

<table>
<thead>
<tr>
<th>CPU burst ($t_j$)</th>
<th>6</th>
<th>4</th>
<th>6</th>
<th>4</th>
<th>13</th>
<th>13</th>
<th>13</th>
<th>…</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;guess&quot; ($\tau_j$)</td>
<td>10</td>
<td>8</td>
<td>6</td>
<td>6</td>
<td>5</td>
<td>9</td>
<td>11</td>
<td>12</td>
</tr>
</tbody>
</table>
Examples of Exponential Averaging

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

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

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

- 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 = highest priority!! maybe).
  - Preemptive
  - nonpreemptive
- SJF is a priority scheduling where priority is the inverse of the predicted next CPU burst time.
- Problem = Starvation (indefinite postponement) – 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), 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 = 20

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P₁</td>
<td>53</td>
</tr>
<tr>
<td>P₂</td>
<td>17</td>
</tr>
<tr>
<td>P₃</td>
<td>68</td>
</tr>
<tr>
<td>P₄</td>
<td>24</td>
</tr>
</tbody>
</table>

The Gantt chart is:

```
P₁  P₂  P₃  P₄  P₁  P₃  P₄  P₁  P₃  P₃
0   20  37  57  77  97  117 121 134 154 162
```

Typically, higher average turnaround than SJF, but better response.
Time Quantum and Context Switch Time

- Process time = 10
  - Quantum: 12
  - Context switches: 0

  - Quantum: 6
  - Context switches: 1

  - Quantum: 1
  - Context switches: 9
Turnaround Time Varies With The Time Quantum

![Graph showing average turnaround time varying with time quantum]
Multilevel Queue

- 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 Scheduling

- Highest priority: system processes
- Interactive processes
- Interactive editing processes
- Batch processes
- Student processes

Lowest priority
Multilevel Feedback 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 a process will enter when that process needs service
Example of Multilevel Feedback Queue

- Three queues:
  - $Q_0$ – time quantum 8 milliseconds
  - $Q_1$ – time quantum 16 milliseconds
  - $Q_2$ – FCFS

- Scheduling
  - A new job enters queue $Q_0$ which is served FCFS. When it gains CPU, job receives 8 milliseconds. If it does not finish in 8 milliseconds, job is moved to queue $Q_1$.
  - At $Q_1$ job is again served FCFS and receives 16 additional milliseconds. If it still does not complete, it is preempted and moved to queue $Q_2$. 
Multilevel Feedback Queues

quantum = 8

quantum = 16

FCFS
Multiple-Processor Scheduling

- CPU scheduling more complex when multiple CPUs are available.
- Assume:
  - Homogeneous processors within a multiprocessor.
  - Uniform memory access (UMA)
- Load sharing - use common ready queue
  - Symmetric – each processor examines ready queue
- Asymmetric multiprocessing – only one processor accesses the system data structures, alleviating the need for data sharing protection.
Real-Time Scheduling

- **Hard real-time** systems – required to complete a critical task within a guaranteed amount of time.
  - Need special purpose software on dedicated hardware
  - No secondary storage or virtual memory

- **Soft real-time** computing – requires that critical processes receive priority over less fortunate ones.
  - Need priority scheduling
  - Need small dispatch latency – difficult
  - Unix: context switch only when systems calls complete or I/O blocks
  - Can insert preemption points in system calls
  - Or make kernel preemptible
  - Read more on this.
Conflict phase: preempt kernel processes/ release low priority process resources needed by higher priority processes
Algorithm Evaluation

- Deterministic modeling – takes a particular predetermined workload and defines the performance of each algorithm for that workload.
- Queueing models – obtain probability distribution from measured CPU and I/O bursts. Treat computer as network of queues of waiting processes with known arrival and service rates.
- Simulations – represent components by software data structures.
  - Use random number generator to generate data.
  - Use trace tapes.
- Implementation.
Evaluation of CPU Schedulers by Simulation

- Actual process execution
- Trace tape
- CPU 10, I/O 213
- CPU 12, I/O 112
- CPU 2, I/O 147
- CPU 175

Simulation for FCFS:
- Performance statistics

Simulation for SJF:
- Performance statistics

Simulation for RR (Q = 14):
- Performance statistics
Solaris 2 Scheduling

- **Global Priority**: Highest to lowest
- **Scheduling Order**: First to last
- **Class-Specific Priorities**: Real time, system, interactive and time sharing
- **Scheduler Classes**: Kernel threads of real-time LWP, kernel service threads, kernel threads of interactive and time-sharing LWP

---

**Image Description**

- The diagram outlines the scheduling mechanism in Solaris 2, focusing on priority levels, scheduling orders, and specific priority classes.
- It shows how threads are categorized into different classes and scheduled accordingly.
- The diagram illustrates the flow from highest priority to lowest, with specific sections dedicated to real-time, system, and interactive tasks.

---

**Key Points**

- **Real Time**: Highest priority, scheduled first.
- **System**: Medium priority, scheduled next.
- **Interactive and Time Sharing**: Lowest priority, scheduled last.

---

**Technical Details**

- Solaris 2 scheduling is designed to manage threads effectively, ensuring real-time responsiveness and efficient use of system resources.
## Windows 2000 Priorities

<table>
<thead>
<tr>
<th>Priority</th>
<th>Real-time</th>
<th>High</th>
<th>Above normal</th>
<th>Normal</th>
<th>Below normal</th>
<th>Idle priority</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>8</td>
<td>6</td>
</tr>
<tr>
<td>above normal</td>
<td>25</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>