CPU Scheduling

- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
  - FCFS, SJF, RR
- Exponential Averaging
- Multi-level Queue Scheduling
- Performance Evaluation

Scheduling Terminology

- Context switch (dispatch) - the actions taken to make the selected process run on the CPU - restore CPU state to that of the selected process when it last ran (all stored in the PCB - registers, cc, PC)
- Degree of multiprogramming - the number of active processes (i.e. in the ready and running states.)
- Swapping - the actions taken to remove a process from memory to reduce the degree of multiprogramming.

Scheduling/Process State Transition

Why/When Processes Switch

- Key Data Structures
  - Process Control Block (PCB)
  - Ready Queue
  - Blocked Queue

Objectives for Schedulers

- GOOD PERFORMANCE FOR THE USER (based on some metric)
  - high throughput - number of jobs completed per time unit
  - low turnaround time - elapsed time from submission to completion, averaged over a set of jobs.
  - low weighted avg. turnaround time - same as above but certain jobs are given more weight.
  - low response time - elapsed time from submission to some response (timesharing).
Objectives for Schedulers (cont’d)

- **GOOD PERFORMANCE FOR THE SYSTEM**
  - CPU Utilization - percent of time the CPU is engaged in useful work over some time frame.
  - FAIRNESS
    - No process should get postponed indefinitely (priorities)
    - STABILITY
      - predictability
      - graceful degradation

CPU burst and I/O Burst

- 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

Types of Schedulers

- **PREEMPTIVE versus NON-PREEMPTIVE**
  - Preemptive - can interrupt a process during its CPU burst and schedule a different process.
  - Non-preemptive - each process completes its CPU burst before next process is scheduled.
    (Can't be used for timesharing.)
- **PRIORITIES (STATIC vs. DYNAMIC)**
  - Static - fixed for the life of the process; dynamic - changes over the life of the process.
  - Determined by characteristics of the process.
  - Aging - dynamically changing the priority over time (to give jobs that have waiting a

Alternating Sequence of CPU And I/O Bursts

- Selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them.
- CPU scheduling decides which process to switch from ready state to running state.
CPU Scheduling Algorithms

- First Come First Served (FCFS)
- Shortest Job First (SJF)
- Round Robin (RR)
- Multi-level Queues
- Multi-level with Feedback
- (There are many others)

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: \( P_1, P_2, P_3 \)

- Waiting time for \( P_1 \) = 0; \( P_2 \) = 24; \( P_3 \) = 27

Average waiting time: \( (0 + 24 + 27)/3 = 17 \)

FCFS Scheduling (Cont.)

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

- The Gantt chart for the schedule is:

  \[ \begin{array}{c|c|c|c|c}
  & P_2 & P_3 & P_1 \\
  \hline
  0 & 2 & 4 & 8 \\
  2 & 4 & 8 & 12 \\
  4 & 8 & 12 & 16 \\
  \end{array} \]

- 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 (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 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

Process | Arrival Time | Burst Time
--- | --- | ---
\( P_1 \) | 0.0 | 7
\( P_2 \) | 2.0 | 4
\( P_3 \) | 4.0 | 1
\( P_4 \) | 5.0 | 4

- SJF

\[ \begin{array}{c|c|c|c|c}
  & P_1 & P_2 & P_3 & P_4 \\
  \hline
  0 & 3 & 7 & 10 & 16 \\
  \end{array} \]

- Average waiting time = \( (0 + 6 + 3 + 7)/4 - 4 \)

Example of Preemptive SJF

Process | Arrival Time | Burst Time
--- | --- | ---
\( P_1 \) | 0.0 | 7
\( P_2 \) | 2.0 | 4
\( P_3 \) | 4.0 | 1
\( P_4 \) | 5.0 | 4

- SJF (preemptive)

\[ \begin{array}{c|c|c|c|c|c|c|c|c|c|c|c|c|c|c}
  & P_1 & P_2 & P_3 & P_4 & P_5 & P_6 \\
  \hline
  0 & 2 & 4 & 6 & 7 & 11 & 16 \\
  \end{array} \]

- Average waiting time = \( (9 + 1 + 0 + 2)/4 - 3 \)
**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).
- 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.

**Determining Length of Next CPU Burst for SJF**

- 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. \( T_{n+1} \) = predicted value for the next CPU burst
3. \( b \) = constant
4. Define: \( T_{n+1} = bT_n + (1 - b)T_n \)

**Prediction of the Length of the Next CPU Burst**

**Examples of Exponential Averaging**

- If \( a = 0 \), \( T_{n+1} = T_n \), recent history does not count.
- If \( a = 1 \), \( T_{n+1} = T_n \), only the actual last CPU burst counts.
- If we expand the formula, we get:
  \[
  T_{n+1} = bT_n + (1 - b)T_n + (1 - b)T_n + \cdots + (1 - b)^nT_0
  \]
- Since both \( b \) and \( (1 - b) \) are less than or equal to 1, each successive term has less weight than its predecessor.

**Exponential Averaging**

**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:
  - If \( q \) large, FIFO
  - If \( q \) small, \( q \) must be large with respect to context switch, otherwise overhead is too high.
Introduction to Scheduling

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>

- Typically, higher average turnaround than SJF, but better response.

Time Quantum and Context Switch Time

Turnaround Time Varies With The 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

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

- Queuing Theory Analysis
  - Single-server Queue:
    - Arrivals
    - Queueing time
    - Service time
    - Departures

Queuing Theory Analysis (cont’d)

- Queuing Theory Analysis (cont’d)
  - Multiple-server Queue:
    - Arrivals
    - Service
    - Departures

Performance for Schedulers

- Queuing Theory Analysis - uses well-established mathematical models and techniques.
- Simulation - create a model of the system and simulate its performance using simulation software.
- Empirical Experiments - implement and test the algorithms in a real system.
Queuing Theory Analysis (cont’d)
- Inputs:
  - arrival rate - from a probability distribution (usually Poisson which implies random arrivals)
  - service time - from a probability distribution (often exponential)
  - scheduling discipline/algorithm

Queuing Theory Analysis (cont’d)
- Outputs:
  - Items waiting
  - Waiting time
  - Items queued
  - Queuing time

Simulation Analysis
- Discrete-event Simulation
  - Often uses models similar to queueing analysis
  - More detailed or more realistic parameters (e.g. trace driven)
  - Simulates events step by step and gathers statistics rather than using mathematical formulas

Evaluation of CPU Schedulers by Simulation

Simulation Analysis (cont’d)

Empirical Experiments
- Properties:
  - Costly and time-consuming
  - Sometimes not possible
  - More realistic

Introduction to Scheduling
Solaris 2 Scheduling

Windows 2000 Priorities

<table>
<thead>
<tr>
<th></th>
<th>high</th>
<th>block</th>
<th>normal</th>
<th>below normal</th>
<th>idle</th>
</tr>
</thead>
<tbody>
<tr>
<td>time-critical</td>
<td>25</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>high</td>
<td>25</td>
<td>15</td>
<td>12</td>
<td>10</td>
<td>6</td>
</tr>
<tr>
<td>normal</td>
<td>14</td>
<td>11</td>
<td>8</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>below normal</td>
<td>24</td>
<td>13</td>
<td>10</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>normal</td>
<td>23</td>
<td>12</td>
<td>7</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>below normal</td>
<td>22</td>
<td>11</td>
<td>6</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>idle</td>
<td>10</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>