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 state
- Add some
- Read from file
- Wait the I/O
- Start increment
- Index order in file
- Wait the I/O
- Load state
- Add some
- Read from file
- Wait the I/O
CPU Scheduler

- Selects a ready process to run on CPU
- Consider process state changes:
  1. from running to waiting state
  2. from running to ready state
  3. from waiting to ready
  4. terminate

Dispatcher

- gives control of the CPU to selected process, actions:
  - 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 per time unit
- **Turnaround time** – amount of time to execute a process
- **Waiting time** – time a process has been waiting in the ready queue
- **Response time** – time from request submission until the first response is produced (time for output of response not counter)

### Scheduling Algorithms

- First-come first-serve
- Shortest job first
- Priority based
- Round robin

### First-Come, First-Served (FCFS) Scheduling

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P₁</td>
<td>24</td>
</tr>
<tr>
<td>P₂</td>
<td>3</td>
</tr>
<tr>
<td>P₃</td>
<td>3</td>
</tr>
</tbody>
</table>

Processes arrive in the order: P₁, P₂, P₃

The Gantt Chart for the schedule is:

<table>
<thead>
<tr>
<th></th>
<th>P₁</th>
<th>P₂</th>
<th>P₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Waiting time for P₁ = 0; P₂ = 24; P₃ = 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:

```
     P_2     P_3     P_1
   0     3        0
```

- 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

- Determine processes' CPU burst length
- Schedule the process with the shortest length
- Two schemes:
  - non-preemptive
    - running process completes its CPU burst.
  - preemptive
    - new process arrives with CPU burst length less than remaining time of current process preempts
- Name: Shortest-Remaining-Time-First (SRTF)
- SJF is optimal
  - 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)

```
     P_1     P_3     P_2     P_4
   0      3       7       12      16
```

- 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>P₁</td>
<td>0.0</td>
<td>7</td>
</tr>
<tr>
<td>P₂</td>
<td>2.0</td>
<td>4</td>
</tr>
<tr>
<td>P₃</td>
<td>4.0</td>
<td>1</td>
</tr>
<tr>
<td>P₄</td>
<td>5.0</td>
<td>4</td>
</tr>
</tbody>
</table>

- SJF (preemptive)

Average waiting time = (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. \( \tau_n \) = predicted value for the next CPU burst
3. \( 0 < \alpha < 1 \)
4. Define:

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

Prediction of the Length of the Next CPU Burst
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:

$$
\tau_{n+1} = \alpha t_n + (1 - \alpha) \alpha t_{n-1} + \ldots + (1 - \alpha)^{j-1} \alpha t_{n-j} + \ldots + (1 - \alpha)^{n-1} t_0
$$

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

Priority Scheduling

- each process has a priority (integer)
- process with highest priority gets CPU
  - (smallest integer = highest priority)
  - preemptive
  - nonpreemptive
- SJF is a form of priority scheduling,
  - 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 runs for quantum of CPU time,
  - Time quantum: usually 10-100 milliseconds
  - After time has elapsed, process is preempted and added to the end of the ready queue
- for $n$ processes each gets $1/n$ of CPU
  - in chunks of time quantum $q$
- 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:

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

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

### Time Quantum and Context Switch Time

<table>
<thead>
<tr>
<th>Process Time = 10</th>
<th>Quantum</th>
<th>Context Switch Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-10</td>
<td>0-12</td>
<td>0</td>
</tr>
<tr>
<td>1-5</td>
<td>4-6</td>
<td>1</td>
</tr>
<tr>
<td>6-10</td>
<td>9-1</td>
<td>9</td>
</tr>
</tbody>
</table>

### Turnaround Time Varies With The Time Quantum

![Graph showing turnaround time varies with time quantum](image)
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
    - 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 to determine when to upgrade a process
  - method to determine when to demote a process
  - method 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

- CPU scheduling more complex when multiple CPUs are available
- Homogeneous processors within a multiprocessor
- Load sharing
- Asymmetric multiprocessing
  - only one processor accesses the system data structures, alleviating the need for data sharing
Real-Time Scheduling

- **Hard real-time systems**
  - required to complete a critical task within a guaranteed amount of time

- **Soft real-time computing**
  - requires that critical processes receive priority over less fortunate ones
  - More effective with kernel preemption
    - Kernel data structures at risk
    - Conflict resolution
    - Preemption points
    - Priority inheritance

Dispatch Latency

Let competing processes finish kernel data access

Algorithm Evaluation

- **Deterministic modeling**
  - takes a particular predetermined workload and defines the performance of each algorithm for that workload

- **Queuing models**

- **Implementation**

Algorithm Evaluation

- **Deterministic modeling**
  - takes a particular predetermined workload and defines the performance of each algorithm for that workload

- **Queuing models**

- **Implementation**
Evaluation of CPU Schedulers by Simulation

Solaris 2 Scheduling

Windows XP Priorities

- Priority is adjusted
  - after each run: down
  - up after return from wait, more for GUI wait
  - active window gets priority boost
  - foreground process get larger time quantum
Linux

- Time-sharing method
  - Priority-based preemptive scheduling

- Credit-based algorithm
  - Credits = credits / 2 + priority
  - Gives preference to processes doing I/O
  - Process with highest credit runs
  - Each time slice consumes 1 credit
  - Process runs until block, or credits exhausted
  - If no process has credits, credits are recomputed for all