**CPU Scheduling (CSE 422S)**

Ken Wong  
Washington University

kenw@wustl.edu  
www.arl.wustl.edu/~kenw

---

**Process Run States (Simplified)**

- **Basic process run states**
  - New
  - Ready
  - Running
  - Waiting
  - Dispatch
  - I/O or Event Wait
  - Complete
  - Parent Forks
  - Preempt
  - Exit
  - Zombie

- **Other possible states**
  - Stopped: Not terminated, but not to be scheduled
  - Zombie

---

**Short-Term Scheduling**

- **Scheduler**
- **CPU**
- **Resource Manager**
- **READY Queue**
- **Blocked Jobs**
- **New Jobs**
- **Preemption or Yield**
- **Done**

- **Long-term scheduler** decides which processes should be scheduled by **short-term scheduler**

---

**Scheduler**

- **Simple Scheduler Tasks**
  - Determine the order in which active processes should contend for the CPU(s)
  - Context switch between one process and another
    - Save current process' CPU state in its Process Control Block (PCB)
    - Load CPU registers from new process' PCB
  - Run the **Idle Process** if no runnable processes

- **Scheduler Architectures**
  - Separate scheduler process
    - e.g., Unix
    - Scheduler kernel process runs when a process blocks, is interrupted (e.g., quantum expires), or is awakened
  - Embedded scheduler function in each thread
    - e.g., Windows 2000
    - Thread enters kernel mode and runs scheduler function to switch context
Multiprocessor Scheduling

- Scheduling is two-dimensional
  - Which process to run?
  - Which CPU should run the process?

- Difficulties
  - Potential contention for scheduling data structures
  - A process holding a spin lock loses the CPU and blocks other processes.
  - The cache of a CPU that has run a process for a long time often has useful data.
  - The TLB of a CPU ... same as above ...
  - A group of related processes would finish faster if scheduled together
    - e.g., Parallel make command

Batched Workload Example

<table>
<thead>
<tr>
<th>Process</th>
<th>Arrival Time</th>
<th>Service Demand</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>B</td>
<td>2 + ε</td>
<td>6</td>
</tr>
<tr>
<td>C</td>
<td>4 + ε</td>
<td>4</td>
</tr>
<tr>
<td>D</td>
<td>6 + ε</td>
<td>5</td>
</tr>
<tr>
<td>E</td>
<td>8 + ε</td>
<td>2</td>
</tr>
</tbody>
</table>

CPU Scheduling Policies

- Non-Preemptive (can't interrupt running process)
  - FCFS (First-Come-First-Served)
  - SJF (Shortest Job First) or SJN (... Next)
  - External Priority
    - Static: Priority is assigned once
    - Dynamic: Priority can change during CPU usage
  - EDF (Earliest Deadline First)

- Preemptive (can interrupt running process)
  - Round-Robin
    - Equitably distribute CPU time among all processes by giving a time slice (quantum) to each READY process
  - Others: SJF or SJN, Priority, EDF

Non-Preemptive Scheduling

<table>
<thead>
<tr>
<th>Process</th>
<th>Arrival Time</th>
<th>Service Demand</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

First-Come-First-Served (smallest arrival time first)

Shortest Job First (smallest service demand)
**CPU Job Performance Parameters**

- **T**: Observation period
- **D**: Number of departures in the interval [0, T]
- **B**: Busy period
- **d(i)**: Service demand of ith arrival
- **t(i)**: Turnaround time of the ith departure
  - Time job departed - Time job arrived to CPU
  - Interactive jobs: response time
- **s(i)**: Accumulated service time of ith departure
  - Total time job was in the RUN state (using the CPU)
- **w(i)**: Waiting (Queueing) time of the ith departure
  - Total time job spent in the READY queue

**Average Performance Metrics**

Notation: \( x(+) = \sum_{i=1}^{n} x(i) \) when there are n jobs

- Average Turnaround Time \( t = t(+) / D \)
- Average Service Time \( s = s(+) / D \)
- Average Waiting Time \( w = w(+) / D = t - s \)
- Throughput (Departure Rate) \( r = D / T \)
- Utilization \( u = B / T \)

**Performance of FCFS and SJF**

<table>
<thead>
<tr>
<th>Processes</th>
<th>Time</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>Avg.</th>
<th>Context Switches</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service</td>
<td></td>
<td>3</td>
<td>6</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>4.0</td>
<td></td>
</tr>
<tr>
<td>Turnaround</td>
<td></td>
<td>3</td>
<td>7</td>
<td>9</td>
<td>12</td>
<td>12</td>
<td>8.6</td>
<td></td>
</tr>
<tr>
<td>Waiting</td>
<td></td>
<td>0</td>
<td>1</td>
<td>5</td>
<td>7</td>
<td>10</td>
<td>4.6</td>
<td></td>
</tr>
<tr>
<td>SJF</td>
<td></td>
<td>3</td>
<td>7</td>
<td>11</td>
<td>14</td>
<td>3</td>
<td>7.6</td>
<td></td>
</tr>
<tr>
<td>Waiting</td>
<td></td>
<td>0</td>
<td>1</td>
<td>7</td>
<td>9</td>
<td>1</td>
<td>3.6</td>
<td></td>
</tr>
</tbody>
</table>

**FCFS versus Round-Robin**

- **FCFS (FIFO)**
  - A 3
  - B 6
  - C 4
  - D 5
  - E 2

- **RR (q=1)**
  - A 3
  - B 15
  - C 18
  - D 20
  - E 16

First-Come-First-Served

Round-Robin (cycle thru READY queue)
FCFS versus Round-Robin

<table>
<thead>
<tr>
<th>Processes</th>
<th>Time</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>Avg. Context Switches</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service</td>
<td></td>
<td>3</td>
<td>6</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>4.0</td>
</tr>
<tr>
<td>FCFS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>Turnaround</td>
<td></td>
<td>3</td>
<td>7</td>
<td>9</td>
<td>12</td>
<td>12</td>
<td>8.6</td>
</tr>
<tr>
<td>Waiting</td>
<td></td>
<td>0</td>
<td>1</td>
<td>5</td>
<td>7</td>
<td>10</td>
<td>4.6</td>
</tr>
<tr>
<td>RR(q=1)</td>
<td></td>
<td>3</td>
<td>16</td>
<td>11</td>
<td>14</td>
<td>8</td>
<td>10.4</td>
</tr>
<tr>
<td>Turnaround</td>
<td></td>
<td>0</td>
<td>10</td>
<td>7</td>
<td>9</td>
<td>6</td>
<td>6.4</td>
</tr>
</tbody>
</table>

Round-Robin Scheduling

- N processes will get (1/N)th of CPU time
- A new process is placed at the end of the RUN/READY queue
- Effect of context switching
  - C = Context switch overhead
  - Each of N processes will get q seconds of CPU service and incur C seconds of overhead \( N(q+C) \) seconds to serve N processes once
- Implementation
  - Set timer to interrupt every q seconds
  - Timer interrupt handler calls scheduler to start next process

Non-Preemptive SJF Scheduling

- Last job finishes at time 20 (=Sum of demands)
- 100% utilization during the period \([0,20]\)
- Avg service time = \((3+6+4+5+2)/5\)  \( s = 4 \)
- Avg turnaround (wait+service) time \( t = 8.6 \)
  - \( = (3+7+9+12+12)/5 \)
- Avg wait time = \((0+1+5+7+10)/5\)  \( w = 4.6 \)
- Sum \( s(i) \times T(i) = 9+42+36+60+24 = 171 \)
### Alternative Scheduling Policies

<table>
<thead>
<tr>
<th></th>
<th>FCFS</th>
<th>RR</th>
<th>SJF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job Selection</td>
<td>Min arrival time</td>
<td>Constant</td>
<td>Min (s(i))</td>
</tr>
<tr>
<td>Decision Mode</td>
<td>Nonpreemptive</td>
<td>Preemptive</td>
<td>Nonpreemptive</td>
</tr>
<tr>
<td>Throughput</td>
<td>-</td>
<td>Lower for smaller quantum</td>
<td>High for short jobs</td>
</tr>
<tr>
<td>Response time</td>
<td>High if large variance in (s(i))</td>
<td>Good for short jobs</td>
<td>Good for short jobs</td>
</tr>
<tr>
<td>Overhead</td>
<td>Minimum</td>
<td>Depends on (q)</td>
<td>Can be high</td>
</tr>
<tr>
<td>Fairness</td>
<td>Can penalize short jobs</td>
<td>Fair</td>
<td>Penalizes long jobs</td>
</tr>
<tr>
<td>Starvation</td>
<td>No</td>
<td>No</td>
<td>Possible</td>
</tr>
</tbody>
</table>

### Shortest Process Next (SPN)

- **Extend batched SJF idea to interactive system**
  - SJF has minimum average turnaround time
- **Interactive Process**
  - Wait for command; Execute Command; Wait ...
  - Treat each command as a job and choose shortest first
  - Which one is the shortest job???
- **Select jobs with shortest estimated burst time**
  - Estimate based on past behavior (aging or exponential averaging)
    - \(\hat{T}(\text{new}) = a\hat{T} + (1-a) \times T\), \(0 \leq a \leq 1\)
    - \(\hat{T}\): Previous estimate based on aging formula
    - \(T\): New measured usage
  - Easy to implement when \(a = \frac{1}{2}\): \(\hat{T}(\text{new}) = (\hat{T} + T) \gg 1\)
  - Small \(a\) \(\Rightarrow\) Past behavior is more important than current

### Exponential Average Example

- **Data:** 20, 19, ..., 11, 10, 10, 10, ...
- **Exponential Average** \(\hat{T}(\text{new}), a = 1/2\)
  - \((20 + 0)/2 = 10\)
  - \((19 + 10)/2 = 14.5\)
  - \((18 + 14.5)/2 = 16.25\)
  - ...
  - \((11 + 12.96)/2 = 11.98\)
  - \((10 + 11.98)/2 = 10.99\)
  - \((10 + 10.99)/2 = 10.5\)
  - \((10 + 10.5)/2 = 10.25\)
  - ... Exponential average converges toward 10

### Fairness ???

- **One Definition**
  - If there are \(N\) users, each user gets 1/\(N\) of the CPU
  - Can generalize to giving user \(i\) \(w(i)\) of the CPU
    - \(w(1) + ... + w(n) = 1\)
- **Algorithm 1**
  - User \(i\) gets \(K(i)\) tickets in proportion to \(w(i)\) periodically
  - Each time slice (quantum) is worth \(Q\) tokens
  - A user gets its time slice of the CPU if \(K(i) \geq Q\)
    - \(K(i)\) is reduced by \(Q\) every time user \(i\) uses a quantum
  - Service users in round-robin order
- **Algorithm 2 (statistical version)**
  - Number the tickets
  - Randomly pick a ticket number to give service
Traditional Unix Scheduling

- System V (Release 3), 4.3 BSD
- Target: Interactive, time-sharing system
  - Good response time for interactive users
  - Long running, background jobs do not starve
  - Multilevel feedback with round robin (q = 1 sec) within each priority queue
- Base priority values
  - Divide all processes into fixed bands of priority levels
  - 'nice' values are restricted to prevent movement out of assigned priority band
  - Bands (highest first): Swapper, Block I/O device, File manipulation, Character I/O device, User process
- Hard-clock interrupt every 10 msec
  - Kernel collects usage statistics and can preempt process

Multilevel Feedback Queue

- Priority value of process in time interval i
  - \( P(i) = B + U'(i-1)/2 + \text{nice} \)
    - \( B \): Base priority value of process
    - \( U'(i) \): Exponential average of CPU utilization of process in time interval i
    - \( \text{nice} \): Nice value of process (user-controllable)
      - between -20 and 20
      - normally 0
    - Smallest value is Highest priority; i.e., schedule process with smallest \( P(i) \) first
- Exponentially weighted average utilization of process
  - \( U'(i) = U(i)/2 + U'(i-1)/2 \)
    - \( U(i) \): CPU utilization of process in time interval i

Exponentially weighted average utilization of process

- Lower priority queues have higher quantums