**CPU Scheduling (CSE 422S)**

Ken Wong  
Washington University

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

---

**Short-Term Scheduling**

Preemption or Yield

New Jobs

Scheduler

CPU

READY Queue

Resource Manager

Request

Blocked Jobs

- **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

---

**Context Switching**

- **Giving CPU to a different process requires a full context switch**
  - Save registers of interrupted process and load registers of next process

- **Full context switch time**
  - $= 2 (n + m) b \times K$
    - $n$ general registers
    - $m$ status registers
    - $b$ memory accesses to save a single register
    - $K$ time units per memory access

- **Example (n=32, m=2, b=1, K=20 nsec)**
  - $2 (n + m) b \times K = 64 \times 20 \text{ nsec} = 1.280 \text{ usec}$
  - $1.28 \text{ usec} = 1280 \text{ machine instructions on a 1 GHz CPU}$
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 processes may be related and 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 (process runs to completion)
  - FCFS (First-Come-First-Served)
  - SJF (Shortest Job First) or SJN (Shortest Next)
  - External Priority
    - Static: Priority is assigned once
    - Dynamic: Priority can change during CPU usage
  - EDF (Earliest Deadline First)
- Preemptive (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

Process Run States (Simplified)

- Basic process run states
  - New
  - Ready
  - Running
  - Waiting
  - Preempt
  - Exit
  - Zombie
- Other possible states
  - Stopped: Not terminated, but not to be scheduled
  - Zombie

process is blocked
Non-Preemptive Scheduling

FCFS (FIFO)

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

- Gantt Chart

SJF

- 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></th>
<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>FCFS</td>
<td>3</td>
<td>6</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>4.0</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Turnaround</td>
<td>SJF</td>
<td>3</td>
<td>7</td>
<td>9</td>
<td>12</td>
<td>12</td>
<td>8.6</td>
<td>5</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>
<td></td>
</tr>
</tbody>
</table>

- CPU Job Performance Parameters
- Average Performance Metrics
- Performance of FCFS and SJF
**FCFS versus Round-Robin**

**FCFS (FIFO)**
- A
- B
- C
- D
- E

**Round-Robin (RR) (q=1)**
- A
- B
- C
- D
- E

**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.</th>
<th>Context Switches</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service</td>
<td>3</td>
<td>6</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td></td>
<td>4.0</td>
<td>5</td>
</tr>
<tr>
<td>FCFS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Turnaround</td>
<td>3</td>
<td>7</td>
<td>9</td>
<td>12</td>
<td>12</td>
<td></td>
<td>8.6</td>
<td></td>
</tr>
<tr>
<td>Waiting</td>
<td>0</td>
<td>1</td>
<td>5</td>
<td>7</td>
<td>10</td>
<td></td>
<td>4.6</td>
<td></td>
</tr>
<tr>
<td>RR(q=1)</td>
<td>3</td>
<td>16</td>
<td>11</td>
<td>14</td>
<td>8</td>
<td></td>
<td>10.4</td>
<td>15</td>
</tr>
<tr>
<td>Turnaround</td>
<td>0</td>
<td>10</td>
<td>7</td>
<td>9</td>
<td>6</td>
<td></td>
<td>6.4</td>
<td></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

**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)
  - Estimate: \( T'(\text{new}) = aT + (1-a) \times T' \), \( 0 \leq a \leq 1 \)
    - \( T' \): Previous estimate based on aging formula
    - \( T \): New measured usage
  - Easy to implement when \( a = \frac{1}{2} \): \( T'(\text{new}) = (T' + T) \gg 1 \)
  - Small \( a \) → Past behavior is more important than current

Exponential Average Example
- Data: 20, 19, ..., 11, 10, 10, ...
- Exponential Average \( T'(\text{new}), a = 1/2 \)
  - \((20 + 0)/2 = 10\)
  - \((19 + 10)/2 = 14.5\)
  - \((18 + 14.5)/2 = 16.25\)
  - ...
  - \((11 + 12.96)/2 = 11.98\)
  - \((10 + 11.98)/2 = 10.99\)
  - \((10 + 10.99)/2 = 10.5\)
  - \((10 + 10.5)/2 = 10.25\)
  - ... 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 where \( w(1) + ... + w(n) = 1 \)
- 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

- Arrivals → Queue 0 → CPU → Departures
- Queue 1 → Timeout
- ... → Queue n
- Low Priority

Lower priority queues have higher quantums

BSD Unix Priority Formulas

- 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