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)
  - $2 (n + m) b \times K = 64 \times 20$ nsec = 1.280 usec
  - 1.28 usec = 1280 machine instructions on a 1 GHz CPU

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</td>
<td>6</td>
</tr>
<tr>
<td>C</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>D</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>E</td>
<td>8</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 (... Next)
  - 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

Non-Preemptive Scheduling

FCFS (FIFO)

0 5 10 15 20

First-Come-First-Served

Gantt Chart

SJF

Shortest Job First

CPU Job Performance Parameters

- T: Observation period
- D: Number of departures in the interval [0,T]
- B: Busy period
- t(i): Turnaround time of the ith departure
  - Time job departed - Time job arrived to CPU
- 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 \( t = \frac{t(+)}{D} \)
- Average Service Time \( s = \frac{s(+)}{D} \)
- Average Waiting Time \( w = \frac{w(+)}{D} = t - s \)
- Throughput (Departure Rate) \( r = \frac{D}{T} \)
- Utilization \( u = \frac{B}{T} \)
Performance of FCFS and SJF

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

FCFS versus Round-Robin

<table>
<thead>
<tr>
<th>Processes</th>
<th>Time</th>
<th>Avg. Context Switches</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service</td>
<td>3 6 4 5 2</td>
<td>4.0</td>
</tr>
<tr>
<td>Turnaround</td>
<td>3 7 9 12 12</td>
<td>8.6</td>
</tr>
<tr>
<td>Waiting</td>
<td>0 1 5 7 10</td>
<td>4.6</td>
</tr>
<tr>
<td>FCFS(FIFO)</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>RR(q=1)</td>
<td>19</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 shortest estimated running time job
  - Estimate based on past behavior (aging or exponential averaging)
  - Estimate: \( T'(\text{new}) = aT + (1-a) \times T', \quad 0 \leq a \leq 1 \)
    - \( T' \): Previous estimate based on aging formula
    - \( T \): New measured usage
  - Easy to implement when \( a = \frac{1}{2} \): \( T'(\text{new}) = \frac{T + T'}{2} \geq 1 \)
  - Small \( a \) -> Past behavior is more important than current

### Multilevel Feedback Queue

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

### Exponential Average Example

- Data: 20, 19, ..., 11, 10, 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
## 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 \((q = 1 \text{ sec})\) 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

## 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 \(i\)