Shortest Remaining Time (SRT)

- Preemptive version of shortest process next policy
- Must estimate processing time

Response Time and Ratio

- Response Ratio $R$ is
  - total time spent waiting and executing normalized to the execution time
  - $w$: waiting time (waiting for a processor)
  - $s$: expected service (execution) time

\[ R = \frac{w + s}{s} \]

- Note: In scheduling theory response time is called flow time $F_i = C_i - r_i$
  - i.e., completion time minus ready time
  - this is the sum of waiting and processing times

<table>
<thead>
<tr>
<th>Process</th>
<th>Arrival Time</th>
<th>Service Time</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>
Highest Response Ratio Next (HRRN)

• Choose next process with the greatest response ratio

Feedback

• SPN, SRT and HRRN require that something is known about the execution times
  – e.g., expected execution time
• Alternative policies
  – give preference to shorter tasks by penalizing tasks that have been running longer
Use multiple queues, pushing tasks to the next queue after each preemption

Feedback

- Potential problems
  - starvation
  - low response times for longer tasks
  - many solutions exists, e.g.,
    - use fixed quantum
      - \( q = 1 \)
    - use different quantum in consequent queues
      - \( q = 2^i \) for queue \( i \)
      - starvation still possible though
        - solution: “promote” jobs to higher queue after some time
• Don’t know remaining time process needs to execute

Table 9.4 Process Scheduling Example

<table>
<thead>
<tr>
<th>Process</th>
<th>Arrival Time</th>
<th>Service Time</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>

Table 9.3 Characteristics of Various Scheduling Policies

<table>
<thead>
<tr>
<th>Selection Function</th>
<th>Decision Mode</th>
<th>Throughput</th>
<th>Response Time</th>
<th>Overhead</th>
<th>Effect on Processes</th>
<th>Starvation</th>
</tr>
</thead>
<tbody>
<tr>
<td>FCFS</td>
<td>max([w])</td>
<td>Nonpreemptive</td>
<td>Not emphasized</td>
<td>May be high, especially if there is a large variance in process execution times</td>
<td>Minimum</td>
<td>Penalizes short processes, penalizes IO bound processes</td>
</tr>
<tr>
<td>Round Robin</td>
<td>constant</td>
<td>Preemptive (at time quantum)</td>
<td>May be low if quantum is too small</td>
<td>Provides good response time for short processes</td>
<td>Minimum</td>
<td>Fair treatment</td>
</tr>
<tr>
<td>SPN</td>
<td>min([z])</td>
<td>Nonpreemptive</td>
<td>High</td>
<td>Provides good response time for short processes</td>
<td>Can be high</td>
<td>Penalizes long processes</td>
</tr>
<tr>
<td>SRT</td>
<td>min([z - \phi])</td>
<td>Preemptive (at arrival)</td>
<td>High</td>
<td>Provides good response time</td>
<td>Can be high</td>
<td>Penalizes long processes</td>
</tr>
<tr>
<td>HRRN</td>
<td>max(\frac{w + \phi}{z})</td>
<td>Nonpreemptive</td>
<td>High</td>
<td>Provides good response time</td>
<td>Can be high</td>
<td>Good balance</td>
</tr>
<tr>
<td>Feedback</td>
<td>(see text)</td>
<td>Preemptive (at time quantum)</td>
<td>Not emphasized</td>
<td>Not emphasized</td>
<td>Can be high</td>
<td>May favor I/O bound processes</td>
</tr>
</tbody>
</table>

\(w\) = time spent waiting  
\(\phi\) = time spent in execution so far  
\(z\) = total service time required by the process, including \(\phi\)
### Table 9.5 A Comparison of Scheduling Policies

<table>
<thead>
<tr>
<th>Process</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arrival Time</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>Service Time (T_s)</td>
<td>1.00</td>
<td>1.17</td>
<td>2.25</td>
<td>2.40</td>
<td>6.00</td>
<td>2.56</td>
</tr>
</tbody>
</table>

- **FCFS**: First In, First Out
  - Finish Time: 3, 9, 13, 18, 20
  - Turnaround Time (T_s): 1.00, 1.17, 2.25, 2.40, 6.00
  - Average Turnaround Time: 8.60

- **RR q = 3**
  - Finish Time: 4, 18, 37, 20, 15
  - Turnaround Time (T_s): 2.67, 3.25, 2.80, 3.50, 2.71

- **RR q = 4**
  - Finish Time: 3, 17, 11, 20, 19
  - Turnaround Time (T_s): 2.5, 1.75, 2.80, 5.50, 2.71

- **SPN**
  - Finish Time: 3, 9, 15, 20, 11
  - Turnaround Time (T_s): 1.00, 1.17, 2.75, 2.00, 1.50
  - Average Turnaround Time: 7.60

- **SRT**
  - Finish Time: 3, 15, 8, 20, 10
  - Turnaround Time (T_s): 3, 13, 4, 14, 2
  - Average Turnaround Time: 7.20

- **HRN**
  - Finish Time: 3, 9, 13, 20, 15
  - Turnaround Time (T_s): 3, 9, 14, 7, 8.00
  - Average Turnaround Time: 8.00

- **FB q = 1**
  - Finish Time: 4, 20, 16, 19, 11
  - Turnaround Time (T_s): 1.33, 3.00, 3.00, 2.60, 1.5
  - Average Turnaround Time: 10.00

- **FB q = 2**
  - Finish Time: 4, 17, 18, 20, 14
  - Turnaround Time (T_s): 1.33, 2.50, 3.50, 2.80, 3.00
  - Average Turnaround Time: 2.63

### Table 9.6 Formulas for Single-Server Queues with Two Priority Categories

**Assumptions:**
1. Poisson arrival rate.
2. Priority 1 items are serviced before priority 2 items.
3. First-in-first-out dispatching for items of equal priority.
4. No item is interrupted while being served.
5. No items leave the queue (lost calls delayed).

#### (a) General Formulas
- \( \lambda = \lambda_1 + \lambda_2 \)  arrival rate
- \( \rho_1 = \lambda_1T_{s1} \)  utilization
- \( \rho = \rho_1 + \rho_2 \)
- \( T_s = \frac{\lambda}{\lambda_2} T_{s1} + \frac{\lambda_2}{\lambda_1} T_{s2} \)  average service time
- \( T_r = \frac{\lambda}{\lambda_2} T_{r_1} + \frac{\lambda_2}{\lambda_1} T_{r_2} \)  turnaround time

#### (b) No interrupts; exponential service times
- \( T_{s1} = T_{s1} + \frac{\rho T_{s1} + \rho_2 T_{s2}}{1 - \rho} \)
- \( T_{r1} = T_{r1} + T_{s1} - T_{s2} \)

#### (c) Preemptive-resume queuing discipline;
 exponential service times
- \( T_{s1} = T_{s1} + \frac{\rho T_{s1}}{1 - \rho_1} \)
- \( T_{r2} = T_{r2} + \frac{1}{1 - \rho} \left( \rho_1 T_{r1} + \rho_2 T_{r2} \right) \)
Fair-Share Scheduling

- All previous approaches treat collection of ready processes as single pool
- User’s application runs as a collection of processes (threads)
  - concern about the performance of the application, not single process; (this changes the game)
  - need to make scheduling decisions based on process sets
Fair-Share Scheduling

• Philosophy can be extended to groups
  – e.g. time-sharing system,
    • all users from one department treated as group
    • the performance of that group should not affect other groups significantly
      – e.g. as many people from the group log in performance degradation should be primarily felt in that group

Fair-Share Scheduling

• Fair share
  – each user is assigned a weight that corresponds to the fraction of total use of the resources
  – scheme should operate approximately linear
    • e.g. if user A has twice the weight of user B, then (in the long run), user A should do twice the work than B.
Traditional UNIX Scheduling

- Multilevel feedback using round robin within each of the priority queues
- If a running process does not block or complete within 1 second, it is preempted
- Priorities are recomputed once per second
- Base priority divides all processes into fixed *bands* of priority levels

Bands

- Decreasing order of priority
  - Swapper
  - Block I/O device control
  - File manipulation
  - Character I/O device control
  - User processes