Uniprocessor Scheduling

Chapter 9
Goals of Scheduling

- Quick response time
- Fast throughput
- Processor efficiency
Type of Scheduling

✓ **Long-term**
   - performed when new process is created

✓ **Medium-term**
   - swapping

✓ **Short-term**
   - decision as to which ready process will be executed by the processor

✓ **I/O**
   - decision as to which process’s pending I/O request shall be handled by available I/O device
Scheduling and Process State Transition

- New
  - Ready, suspend
    - Blocked, suspend
    - Ready
      - Running
        - Exit
  - Ready, suspend
    - Medium-term scheduling
  - Blocked
    - Medium-term scheduling
    - Short-term scheduling

Long-term scheduling

Short-term scheduling

Medium-term scheduling
Queuing Diagram for Scheduling

- Batch jobs
- Interactive users
- Long-term scheduling
  - Ready Queue
  - Medium-term scheduling
    - Ready, Suspend Queue
    - Blocked, Suspend Queue
    - Blocked Queue
  - Time-out
    - Event Wait
    - Medium-term scheduling
      - Release
Long-Term Scheduling

✓ Determines which programs are admitted to the system for processing
✓ Controls the degree of multiprogramming
✓ More processes, smaller percentage of time each process is executed
Medium-Term Scheduling

- Swapping

- Based on the need to manage multiprogramming (it is hard to foresee the CPU and memory requirements of processes in the long-term scheduling phase).
Short-Term Scheduling

✓ Performed by the *dispatcher*

✓ Invoked when an event occurs, e.g.
  - clock interrupt
  - I/O interrupt
  - operating system call
  - signal (e.g., when software events occur)
Short-Tem Scheduling Criteria

✓ User-oriented
  ▪ Response Time
    o Elapsed time between the submission of a request until there is output.

✓ System-oriented
  ▪ effective and efficient utilization of the processor

✓ Performance-related
  ▪ response time and throughput

✓ Not performance related
  ▪ predictability (e.g., fairness, no starvation)
Priorities

✓ Scheduler will always choose a process of higher priority over one of lower priority
✓ Have multiple ready queues to represent each level of priority
✓ Lower-priority may suffer starvation
  ▪ allow a process to change its priority based on its age or execution history
Priority Queuing

- Admit
- RQ0
- RQ1
- ... (multiple RQn)
- Blocked Queue
- Event occurs
- Preemption
- Dispatch
- Processor
- Release
- Event Wait
Decision Mode

✔ **Nonpreemptive**
  - Once a process is in the running state, it will continue until it terminates or blocks itself for I/O

✔ **Preemptive**
  - Currently running process may be interrupted and moved to the Ready state by the operating system
  - Allows for better service since no process can monopolize the processor for very long
An Example

<table>
<thead>
<tr>
<th>Process</th>
<th>Arrival Time</th>
<th>Service Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>2</td>
</tr>
</tbody>
</table>
First-Come-First-Served (FCFS)

Each process joins the Ready queue

When the current process ceases to execute, the oldest process in the Ready queue is selected
First-Come-First-Served (FCFS)

✓ A short process may have to wait a very long time before it can execute
✓ Favors CPU-bound processes
  ▪ I/O processes have to wait until CPU-bound process completes
Round-Robin

✓ Uses preemption based on a clock
✓ An amount of time is determined that allows each process to use the processor for that length of time
Shortest Process Next

- Nonpreemptive policy
- Process with shortest expected processing time is selected next
- Short process jumps ahead of longer processes
Predictability of longer processes is reduced
If estimated time for process not correct, the operating system may abort it
Possibility of starvation for longer processes
**Shortest Remaining Time**

- Preemptive version of shortest process next policy
- Must be able to estimate remaining processing time
Choose next process with the highest ratio of:

\[
\frac{\text{time spent waiting} + \text{expected service time}}{\text{expected service time}}
\]
Feedback

- Penalize jobs that have been running longer
- Can be used when we don’t know the remaining time a process needs to execute
- Implemented in UNIX (see later)
Fair-Share Scheduling

- User’s application runs as a collection of processes (threads)
- User is concerned about the performance of the application
- Need to make scheduling decisions based on groups of processes
- For each process, there must be an a priori upper limit on the waiting time
UNIX Scheduling

✓ Priorities are recomputed once per second
✓ Base priority divides all threads into fixed bands of priority levels
✓ Adjustment factor used to keep process in its assigned band

\[ P(i) = Base + \frac{CPU(i)}{2} + \text{nice} \]

- \( i \) – i-th interval
- CPU(i) – CPU utilization by the thread thus far
- nice – user controllable factor (rare)
- P – priority: lower values mean higher priority
Feedback

✓ Process is demoted to the next lower-priority queue each time it returns to the ready queue
✓ Longer processes drift downward
✓ To avoid starvation, CPU time slices for lower-priority processes are longer