Lecture Topics

- Today: Uniprocessor Scheduling  
  (Stallings, chapter 9.1-9.3)  
- Next: continued

Announcements

- Self-Study Exercise #9  
- Project #8 (due 11/15)  
- Project #9 (due 11/29)
Processor Scheduling

- Operating system responsible for allocating resources among competing processes
- OS must schedule processes to use processor cycles
- Scheduling must balance multiple objectives, including effective utilization of the CPU and fairness to all processes

Stallings Chapter 9 and Chapter 10:

- Uniprocessor scheduling
- Multiprocessor scheduling
- Real-time scheduling

We'll start with uniprocessor scheduling, then move on to the others
Uniprocessor Scheduling

- Goal: allocate CPU cycles to processes in a way that meets system objectives, such as fast response time, high throughput, and high processor efficiency

- Three types of scheduling:
  - Long-term
  - Medium-term
  - Short-term

Types of Scheduling

- Long-term: decision to add a process to the set of processes which are in the system

- Medium-term: decision to add a process to the set of processes that are at least partially resident in memory (not suspended)

- Short-term: decision to dispatch a process from the Ready pool
Types of Scheduling

Diagram showing the process flow from New to Ready to Running to Exit, with subcategories for Long-term scheduling, Medium-term scheduling, and Short-term scheduling. The diagram includes transitions from Ready to Blocked and Suspend, and from Blocked to Ready.
## Long-Term Scheduling

- Decides when to admit new processes (influences degree of multiprogramming)
- Places new process in Ready state or Ready-Suspended state
- Batch jobs: pool of jobs on disk, OS selects one to admit (FCFS? Priority?)
- Interactive users: automatically admitted up to some limit

## Medium-Term Scheduling

- Decides when to "unsuspend" a process (influences degree of multiprogramming)
- Processes were placed in one of the two Suspended states to balance resource utilization (ex: too many processes in RAM)
- Suspended processes can be moved to the Ready state or the Blocked state when resources are available
Short-Term Scheduling

- Decides which process should be moved from the Ready state to the Running state
- Short-term scheduler invoked whenever the OS has control of the CPU:
  - System call
  - I/O interrupt
  - Timer interrupt
Frequency

- Long-term scheduling: low frequency (decision to admit new process)
- Medium-term scheduling: moderate frequency (decision to swap in process)
- Short-term scheduling: high frequency (whenever a process is to be dispatched)

Short-term Scheduling Metrics

- Turnaround time: time from submission to completion of a batch job (time executing plus time waiting for resources, including CPU)
- Response time: time it takes system to respond to an interactive command (from end of input to start of response)
- Throughput: number of processes completed per unit of time
Short-term Scheduling Criteria

- **Goal:** allocate processor cycles to optimize system behavior
- **Examples:**
  - minimize response time
  - maximize throughput
  - treat all processes fairly
- **Criteria are interdependent and sometimes conflicting**

User-oriented Criteria

Criteria that are of primary interest to users:

- Turnaround time
- Response time
- Predictability
- Ability to meet deadlines
System-oriented Criteria

Criteria that are systems oriented:

- Throughput
- Processor utilization
- Resource balancing
- Ability to enforce priorities

Review: typical execution pattern

Execution of a process is usually a series: CPU burst followed by I/O burst (end with CPU burst)
Scheduling Policies

- Selection function based on characteristics
  - process priority
  - waiting time (so far)
  - execution time (so far)
  - total service time (stated by user)

- Example: FCFS (first-come-first-served) selects the process which has been waiting for the longest time

Scheduling Policies

- Non-preemptive decision mode -- process in Running state keeps control of CPU until:
  - process terminates
  - process blocks itself

- Example: current process uses system call to initiate I/O operation, so OS moves that process from Running to Blocked and then dispatches a different process
Scheduling Policies

- Preemptive decision mode -- OS may move process from Running to Ready when:
  - interrupt from timer
  - interrupt from I/O device

- Example: I/O operation initiated by different process completes, so OS moves that process from Blocked to Ready; OS moves current process from Running to Ready and then dispatches some other process.

First-Come-First-Served Scheduling

- Process which has been in the Ready queue for the longest time is selected.
- Non-preemptive strategy (current process must voluntarily give up the CPU)
- Low overhead (minimizes process switches)
- Not practical unless based on priority system (separate FIFO queues for each priority level)
First-Come-First-Served Scheduling

- Throughput may be low (CPU-bound processes will hold CPU for long periods)
- Response time and turnaround time may be high (short processes and I/O-bound processes have to wait for long periods of time while CPU-bound processes hold the CPU)
- Starvation not possible (as long as processes complete)

Example: FCFS

<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>

Diagram showing the timeline of processes A, B, C, D, and E with their arrival times and service times.
Example: FCFS

- Turnaround time: waiting time + service time
- Normalized turnaround time: ratio of turnaround time to service time

<table>
<thead>
<tr>
<th>Process</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arrival Time</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>Service Time ($T_s$)</td>
<td>3</td>
<td>6</td>
<td>4</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td><strong>FCFS</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Finish Time</td>
<td>3</td>
<td>9</td>
<td>13</td>
<td>18</td>
<td>20</td>
</tr>
<tr>
<td>Turnaround Time ($T_r$)</td>
<td>3</td>
<td>7</td>
<td>9</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>$T_r/T_s$</td>
<td>1.00</td>
<td>1.17</td>
<td>2.25</td>
<td>2.40</td>
<td>6.00</td>
</tr>
</tbody>
</table>

Round-Robin Scheduling

- Timer interrupt gives OS control of CPU, selects next process from Ready queue (time slicing)
- Preemptive strategy (current process interrupted and must give up CPU)
- Low overhead (but more than FCFS)
- Effective for general-purpose interactive systems
Round-Robin Scheduling

- Throughput may be low (CPU-bound processes still favored)
- Response time good for short processes
- Starvation not possible
- Some overhead due to frequent timer interrupts and process switches

Round-Robin Time Quantum

- Main issue: length of time quantum
- Short quantum: short processes favored, but more overhead (each process switch)
- Long quantum: degenerates to FCFS
- Ideal: time quantum should be somewhat greater than time required for typical interaction
Round-Robin Time Quantum

Time quantum greater than typical interaction: most processes voluntarily give up CPU before timer interrupt occurs.

Round-Robin Time Quantum

Time quantum less than typical interaction: most processes preempted before timer interrupt occurs.
**Example: RR (q = 1)**

<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>

![Process timeline for RR (q = 1)](image)

**Example: RR (q = 4)**

<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>

![Process timeline for RR (q = 4)](image)
### Example: FCFS and RR (q = 1, q = 4)

<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>0</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>Service Time (Ts)</td>
<td>3</td>
<td>6</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>Mean</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>Finish Time</th>
<th>Turnaround Time (Tr)</th>
<th>Tr/Ts</th>
</tr>
</thead>
<tbody>
<tr>
<td>FCFS</td>
<td>3 9 13 18 20</td>
<td>3 7 9 12 12</td>
<td>1.00 1.17 2.25 2.40 6.00</td>
</tr>
<tr>
<td>RR q = 1</td>
<td>4 18 17 20 15</td>
<td>4 16 13 14 7</td>
<td>1.33 2.67 3.25 2.80 3.50</td>
</tr>
<tr>
<td>RR q = 4</td>
<td>3 17 11 20 19</td>
<td>3 15 7 14 11</td>
<td>1.00 2.5 1.75 2.80 5.50</td>
</tr>
</tbody>
</table>

### Summary: FCFS and RR

- **FCFS**: simple and low overhead, not practical without some priority strategy
- **FCFS** favors CPU-bound processes (hold CPU for long time periods)
- **RR**: relatively simple and relatively low overhead, used in many systems
- **RR** favors CPU-bound processes (placed back in Ready queue sooner)
Virtual Round-Robin Scheduling

- VRR: refinement to RR which does not favor CPU-bound processes
- Two Ready queues: main Ready queue and Auxiliary Ready queue (processes which moved from Blocked to Ready)
- Processes in Auxiliary Ready queue are scheduled before processes in main Ready queue
Priority-based Scheduling

Many systems use priority-based scheduling

- Each process assigned a priority
- Scheduler selects process with highest priority (FCFS within a category)
- Starvation possible for processes with low priority (more complex variations prevent starvation)
Feedback Scheduling

- Give preference to short processes by penalizing processes which have been executing for a long time (lower their priority)
- Preemptive strategy
- Moderate to high overhead
- Effective for general-purpose interactive systems

Feedback Scheduling

- Process which gives up the CPU voluntarily remains at the same level
- Process which uses up its time quantum is moved to a lower level
- Older, longer processes drift to lower levels
- Starvation is possible (processes in lower levels may not get access to CPU)
Feedback Scheduling: Variations

- Use a different quantum for each level (longer quantum in lower queues)
- Allow a process to remain at the same level for several preemptions before moving it to a lower level
- Move a process to a higher level if it has not been scheduled for some period of time (prevents starvation)