The plan will most likely hold for other exercise sessions too.

- Introduction to Material Relevant to a Task
- *Solve that Task*: Time 15-20 mins
- Discuss the Solution *for that Task*
- Pick next task. Go back to Step 1.
Real Time Systems are Special

• Is a Real Time System related to High Speed System?
  – Does a Real Time System always need to be a High Speed System?
  – What characteristics does a Real Time System need to have?
• Is "Real Time" only a scheduling problem?
Real Time Systems are Special

• Real Time Systems *must* meet given timing constraints
  – Real Time Systems have *Deadlines*
  – Requires Timing Predictability, not necessary for High Speed Systems
• A *Systems* Concept
  – Need well-designed hardware, software and communication network
• **Processor Utilization**: The ratio of busy time of the processor to the total time required for all tasks to finish. Ideally, utilization = 100%

• **Waiting Time**: Time spent by a task in the *ready* queue.

• **Response Time**: The amount of time it takes to *finish* executing a task, from the moment a process is ready to execute.

• **Throughput**: The measure of work done in a unit time interval.

• See also lecture slides 3-4 to 3-6, 3-13.

• **Fair Schedule**: For this exercise, a schedule is fair if every task eventually gets a chance to execute on the processor.
Task 2: First Come, First Serve-I

<table>
<thead>
<tr>
<th>Process</th>
<th>Arrival Time</th>
<th>1st exec. length</th>
<th>2nd exec. length</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

- Tasks added to a First In-First Out (FIFO) data structure
- Non-Preemptive Algorithm
  - If a task needs to execute *repeatedly*, each successive execution treated as a *new* task
  * Queued into the FIFO at the end

---

Real Time Systems

Spring 2014
## Task 2: First Come, First Serve-I

<table>
<thead>
<tr>
<th>Process</th>
<th>Arrival Time</th>
<th>1&lt;sup&gt;st&lt;/sup&gt; exec. length</th>
<th>2&lt;sup&gt;nd&lt;/sup&gt; exec. length</th>
<th>3&lt;sup&gt;rd&lt;/sup&gt; exec. length</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>10</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>3</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>2</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>D</td>
<td>5</td>
<td>1</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

### Task Arrivals

<table>
<thead>
<tr>
<th>Task Arrivals</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
</tr>
</tbody>
</table>

### Ready Queue

<table>
<thead>
<tr>
<th>Ready Queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
### Task 2: First Come, First Serve-II

<table>
<thead>
<tr>
<th>Process</th>
<th>Arrival Time</th>
<th>1&lt;sup&gt;st&lt;/sup&gt; exec. length</th>
<th>2&lt;sup&gt;nd&lt;/sup&gt; exec. length</th>
<th>3&lt;sup&gt;rd&lt;/sup&gt; exec. length</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>10</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>2</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>D</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>–</td>
</tr>
</tbody>
</table>

### Diagram

![Diagram showing the execution of processes A, B, C, and D with arrival times and execution lengths.]
Task 3: Shortest Job First

<table>
<thead>
<tr>
<th>Process</th>
<th>Arrival Time</th>
<th>1st exec. length</th>
<th>2nd exec. length</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

- Non Preemptive Algorithm (Preemptive version: _Shortest Remaining Time Next_)
- Minimizes the _average waiting time_ (How?)
- Scheduler picks up the task with the shortest execution time from the _ready queue_

A's second execution smaller than B's first execution. Thus, A runs before B.
## Task 3: Shortest Job First

<table>
<thead>
<tr>
<th>Process</th>
<th>Arrival Time</th>
<th>1&lt;sup&gt;st&lt;/sup&gt; exec. length</th>
<th>2&lt;sup&gt;nd&lt;/sup&gt; exec. length</th>
<th>3&lt;sup&gt;rd&lt;/sup&gt; exec. length</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>10</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>3</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>2</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>D</td>
<td>5</td>
<td>1</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

### Task Arrivals

- **Task Arrivals:**
  - A
  - B
  - C
  - D

### Ready Queue

- **Ready Queue:**
  - A
  - B
  - C
  - D

---

**Out of B,C,D:**
- Task D has shortest execution time, Pick D to run.

**Out of B,C:**
- Task C has shortest execution time, Pick C to run.
How does $SJF$ minimize Average Waiting Time?

Catch: Scheduler needs to know *apriori* the execution times of all tasks.
• Preemptive Algorithm
  – As a new task arrives, the scheduler determines which of the ready tasks has the smallest execution time, and executes it.

• Attempts to minimize the average waiting time
  – Dynamic Algorithm, so not strictly minimal.
### Task 4: Shortest Remaining Time Next

<table>
<thead>
<tr>
<th>Process</th>
<th>Arrival Time</th>
<th>1st exec. length</th>
<th>2nd exec. length</th>
<th>3rd exec. length</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>8</td>
<td>8</td>
<td>-</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td>D</td>
<td>7</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Remember to carefully work out the remaining times!
Task 4: Shortest Remaining Time Next

Task Arrivals

A
B
C
D

A1
C1
C2
C2
D1
D2
D3
A2
A3
B1

B2

0 2 4 6 8 10 12 14 16 18 20 28 36

Legend

Exec #

\begin{align*}
\tau_A^1 &= 2* \\
\tau_B^1 &= 8 \\
\tau_C^1 &= 1* \\
\tau_B^1 &= 8 \\
\tau_C^1 &= 2 \\
\tau_A^2 &= 4 \\
\tau_B^2 &= 8 \\
\tau_C^2 &= 2* \\
\tau_A^2 &= 4 \\
\tau_B^2 &= 8 \\
\tau_C^2 &= 2* \\
\tau_A^2 &= 4 \\
\tau_B^2 &= 8 \\
\tau_C^2 &= 1* \\
\tau_A^3 &= 4* \\
\tau_B^3 &= 8 \\
\tau_A^3 &= 4* \\
\tau_B^3 &= 8 \\
\end{align*}

Task with * is selected for execution

Remaining Time

Real Time Systems

Spring 2014
Task 5: Round Robin Scheduling

Time Quantum: 2, Context Switching Time: 0

<table>
<thead>
<tr>
<th>Process</th>
<th>Arrival Time</th>
<th>Exec. length</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>5</td>
</tr>
</tbody>
</table>

Preemptive Algorithm
- Task preempted if it exceeds its \textit{time quantum}.
- Or when it gets done.

Is it a fair scheduling algorithm?

- A's quantum exceeded, pre-empted by B.
- A finishes execution. B resumes execution.

Real Time Systems

Spring 2014
**Task 5: Round Robin Scheduling**

<table>
<thead>
<tr>
<th>Process</th>
<th>Arrival Time</th>
<th>Computation Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>B</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>D</td>
<td>1</td>
<td>5</td>
</tr>
</tbody>
</table>

**Task Arrivals**

A D C B

A | C | D | C | D | B | C | D | B | C

0 2 4 6 8 10 12 14 16 18
Task 6: Scheduling Function

- **Lateness of a Task**: The time past its deadline that a task requires to complete execution, i.e., $L = f - d$. $f$: actual finishing time of a task, $d$: deadline of the task.

- **Laxity**: The time difference between time span to deadline and (remaining) execution time which indicates the available flexibility for scheduling of corresponding task.

- **Feasible Schedule**: All tasks meet their deadlines.
Real Time Systems are Special. And Awesome.

But designing these is not trivial.

• What information do you need to design a Real-Time System?
  – What if you knew the running times of each task, \textit{apriori}? Would it be enough?
• What are other challenges?