Queued Tasks

Real Time Systems

Georgia Giannopoulou • georgia.giannopoulou@tik.ee.ethz.ch
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
Definitions: Performance Metrics

- **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>Task</th>
<th>Arrival Time</th>
<th>1&lt;sup&gt;st&lt;/sup&gt; Exec. time</th>
<th>2&lt;sup&gt;nd&lt;/sup&gt; Exec. time</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*
## Task 2: First Come, First Serve-I

<table>
<thead>
<tr>
<th>Task</th>
<th>Arrival Time</th>
<th>1&lt;sup&gt;st&lt;/sup&gt; exec. time</th>
<th>2&lt;sup&gt;nd&lt;/sup&gt; exec. time</th>
<th>3&lt;sup&gt;rd&lt;/sup&gt; exec. time</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

```
A  B  C  D
```

### Ready Queue

```
A  B  B  C  B  B  C  D  C  D
```
### Task 2: First Come, First Serve-II

<table>
<thead>
<tr>
<th>Task</th>
<th>Arrival time</th>
<th>1st exec. time</th>
<th>2nd exec. time</th>
<th>3rd exec. time</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:**

- **Ready Queue:**
  - A1
  - B1
  - B1
  - C1
  - C1
  - D1
  - D1
  - A2
  - D1
  - A2
  - B2
  - D2
  - A3
  - B3

- **Notes:**
  - C does not queue again!
  - It finished all executions.
Task 3: Shortest Job First

<table>
<thead>
<tr>
<th>Task</th>
<th>Arrival time</th>
<th>1&lt;sup&gt;st&lt;/sup&gt; exec. time</th>
<th>2&lt;sup&gt;nd&lt;/sup&gt; exec. time</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
# Task 3: Shortest Job First

<table>
<thead>
<tr>
<th>Task</th>
<th>Arrival time</th>
<th>1\textsuperscript{st} exec. time</th>
<th>2\textsuperscript{nd} exec. time</th>
<th>3\textsuperscript{rd} exec. time</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**

- **A**
- **B**
- **C**
- **D**

**Ready Queue**

- **A**
- **B**
- **B**
- **C**
- **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?

\[
\begin{align*}
&T_1 \quad T_2 \\
&0 \quad C_{T_1} \quad C_{T_1} + C_{T_2} \quad (a) \\
&T_2 \quad T_1 \\
&0 \quad C_{T_2} \quad C_{T_1} + C_{T_2} \quad (b)
\end{align*}
\]

\[
\frac{C_{T_1}}{2} > \frac{C_{T_2}}{2}
\]

**Catch:** Scheduler needs to know *apriori* the execution times of all tasks.
Task 4: Shortest Remaining Time Next

<table>
<thead>
<tr>
<th>Task</th>
<th>Arrival time</th>
<th>1st exec. time</th>
<th>2nd exec. time</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>

- **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>Task</th>
<th>Arrival time</th>
<th>1&lt;sup&gt;st&lt;/sup&gt; exec. time</th>
<th>2&lt;sup&gt;nd&lt;/sup&gt; exec. time</th>
<th>3&lt;sup&gt;rd&lt;/sup&gt; exec. time</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 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  22  24  26  28  30  32  34  36

\[ r_A^1 = 2^* \]
\[ r_B^1 = 8 \]
\[ r_C^1 = 1^* \]
\[ r_B^1 = 8 \]
\[ r_C^1 = 2 \]

\[ r_A^2 = 4 \]
\[ r_B^1 = 8 \]
\[ r_C^2 = 2^* \]
\[ r_B^1 = 8 \]
\[ r_C^2 = 2^* \]

\[ r_A^3 = 4 \]
\[ r_B^3 = 8 \]
\[ r_C^3 = 0 \]
\[ r_D^3 = 1^* \]

\[ r_A^2 = 4 \]
\[ r_B^1 = 8 \]
\[ r_C^2 = 1^* \]
\[ r_D^1 = 2 \]

\[ r_A^3 = 4 \]
\[ r_B^3 = 8 \]

Legend

Exec #

\[ r_A^3 = 4^* \]

Task

Task with * is selected for execution

Remaining Time

Real Time Systems

Spring 2016
Task 5: Round Robin Scheduling

Time Quantum: 2, Context switching time: 0

<table>
<thead>
<tr>
<th>Task</th>
<th>Arrival time</th>
<th>Exec. time</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 *time quantum*.
  - Or when it gets done.

- Is it a fair scheduling algorithm?
Task 5: Round Robin Scheduling

<table>
<thead>
<tr>
<th>Task</th>
<th>Arrival time</th>
<th>Exec. 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

0 2 4 6 8 10 12 14 16 18

A D C B

Ready Queue (FIFO)

A D C D D A D D C B C C C
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, *apriori*? Would it be enough?
- What are other challenges?