CS 422/522 Design & Implementation of Operating Systems

Real-Time Systems

MAN-KI YOON
DEPT. OF COMPUTER SCIENCE
YALE UNIVERSITY

Part of the slides are based on UIUC CS 431 Lecture Notes
What is a Real-Time System?
What is a Real-Time System?
What is a Real-Time System?
What is a Real-Time System?

[Definition of Real-Time System]

1. of or relating to applications in which the computer must respond as rapidly as required by the user or necessitated by the process being controlled.
What is a Real-Time System?
What is a Real-Time System?

Real-Time

Correctness depends on both functional and temporal aspects
Real-Time Systems vs General-Purpose Systems

Real-Time Systems

Meeting timing requirements

General-Purpose Systems

Optimizing average performance
Typical Real-Time Systems

Avionics and automotive systems
Radar systems
Factory process control
Robotics
Multi-media systems
…
Tasks and Jobs

**Task**: A sequence of the same type of jobs (e.g., process or thread)

**Job**: A unit of computation, e.g.,
- Reading sensor values
- Computing control commands
Periodic Task Model

A task is said to be periodic if its inter-arrival time (i.e., period) is a constant
Periodic Task Model

A task is said to be periodic if its inter-arrival time (i.e., period) is a constant.
Periodic Task Model

A task is said to be *periodic* if its inter-arrival time (i.e., period) is a constant.
Periodic Task Model

A task is said to be periodic if its inter-arrival time (i.e., period) is a constant.
Periodic Task Model

A task is said to be **periodic** if its inter-arrival time (i.e., period) is a constant.
Periodic Task Model

A task is said to be periodic if its inter-arrival time (i.e., period) is a constant.

Scheduled if all jobs meet the relative deadlines.
## Periodic Task Model

![Table of Periodic Task Model](image)

Source: Generic Avionics Software Specification
Priority and Criticality

• Priority: the **order** we execute ready jobs
  - Fixed-priority vs Dynamic-priority

• Criticality: the **penalty** if a task misses its deadline
  - Usually qualitative

• How do we assign priorities to tasks or jobs?
Priority and Criticality

Task 1

Task 2

Critical task
(CPSC422 Assignment)

Non-critical task
(Instagram)

Should we give a higher priority to Task 1? Or Task 2?
Priority and Criticality

Case 1: Priority(Task 1) > Priority(Task 2)

Critical task
(CPSC422 Assignment)

Non-critical task
(Instagram)

Deadline miss!
Priority and Criticality

Case 2: \( \text{Priority(Task 1)} < \text{Priority(Task 2)} \)

Both tasks are schedulable!
Priority and Criticality

• Importance (i.e., criticality) *may or may not* correspond to scheduling priority.
  • Priority is derived from timing requirements

• Importance matters *only when* tasks can be scheduled without missing deadlines.
Notations

Task Utilization: \( U_i = \frac{C_i}{p_i} \)

- Period: \( p_i \)
- Task: \( \tau_i \)
- Job: \( J_{i,j} \)
- Worst-case Execution Time: \( C_i \)
- (Relative) Deadline: \( D_i \)
Real-Time Scheduling Algorithms

• **Rate-Monotonic (RM)**
  • Assign higher priority to *tasks* that have higher-rate (=shorter period)
  • Optimal fixed-priority scheduling

• **Earliest Deadline First (EDF)**
  • Assign higher priority to *jobs* that have earlier relative deadline
  • Optimal dynamic-priority scheduling
Real-Time Scheduling Algorithms

• **Rate-Monotonic (RM)**
  - Assign higher priority to *tasks* that have higher rate (= shorter period)
  - Optimal fixed-priority scheduling

• **Earliest Deadline First (EDF)**
  - Assign higher priority to *jobs* that have earlier relative deadline
  - Optimal dynamic-priority scheduling

What does it mean by ‘optimal’ scheduling?
Real-Time Scheduling Algorithms

- **Rate-Monotonic (RM)**
  - Assign higher priority to *tasks* that have higher rate (=shorter period)
  - Optimal fixed-priority scheduling

- **Earliest Deadline First (EDF)**
  - Assign higher priority to *jobs* that have earlier relative deadline
  - Optimal dynamic-priority scheduling

What does it mean by ‘optimal’ scheduling?

If a task set is not schedulable by the optimal scheduling algorithm, no other scheduling algorithms can schedule the task set.
Rate-Monotonic (RM)

\[ \tau_1 := (p_1 = 5, C_1 = 2) \]

\[ \tau_2 := (p_2 = 9, C_2 = 4) \]
Rate-Monotonic (RM)

\[ \tau_1 := (p_1 = 5, C_1 = 2) \]
\[ \tau_2 := (p_2 = 9, C_2 = 4) \]
Rate-Monotonic (RM)

\[ \tau_1 := (p_1 = 5, C_1 = 2) \]
\[ \tau_2 := (p_2 = 9, C_2 = 4) \]
Rate-Monotonic (RM)

\[ \tau_1 := (p_1 = 5, C_1 = 2) \]
\[ \tau_2 := (p_2 = 9, C_2 = 4) \]
Rate-Monotonic (RM)

$\tau_1 := (p_1 = 5, C_1 = 2)$
$\tau_2 := (p_2 = 9, C_2 = 4)$
Rate-Monotonic (RM)

\[ \tau_1 := (p_1 = 5, C_1 = 2) \]
\[ \tau_2 := (p_2 = 9, C_2 = 4) \]
Rate-Monotonic (RM)

\[ \tau_1 := (p_1 = 5, C_1 = 2) \]
\[ \tau_2 := (p_2 = 9, C_2 = 4) \]

If response time <= deadline, the job is *schedulable*.
Earliest Deadline First (EDF)

$\tau_1 := (p_1 = 5, C_1 = 2)$

$\tau_2 := (p_2 = 9, C_2 = 4)$
Earliest Deadline First (EDF)

\[ \tau_1 := (p_1 = 5, C_1 = 2) \]
\[ \tau_2 := (p_2 = 9, C_2 = 4) \]
Earliest Deadline First (EDF)

$\tau_1 := (p_1 = 5, C_1 = 2)$

$\tau_2 := (p_2 = 9, C_2 = 4)$

Q: What happens next?
Earliest Deadline First (EDF)

\[
\tau_1 := (p_1 = 5, C_1 = 2)
\]

\[
\tau_2 := (p_2 = 9, C_2 = 4)
\]
Earliest Deadline First (EDF)

\[ \tau_1 := (p_1 = 5, C_1 = 2) \]
\[ \tau_2 := (p_2 = 9, C_2 = 4) \]
Earliest Deadline First (EDF)

\[ \tau_1 := (p_1 = 5, C_1 = 2) \]
\[ \tau_2 := (p_2 = 9, C_2 = 4) \]
Earliest Deadline First (EDF)

\[ \tau_1 := (p_1 = 5, C_1 = 2) \]

\[ \tau_2 := (p_2 = 9, C_2 = 4) \]
Earliest Deadline First (EDF)

\[ \tau_1 := (p_1 = 5, C_1 = 2) \]
\[ \tau_2 := (p_2 = 9, C_2 = 4) \]
Earliest Deadline First (EDF)

\[ \tau_1 := (p_1 = 5, C_1 = 2) \]
\[ \tau_2 := (p_2 = 9, C_2 = 4) \]
Schedulability Analysis

• How can we know if a set of periodic tasks is schedulable?
  • Exact test
  • Utilization bound test
Exact Test

• A.k.a. Response time analysis

• For fixed-priority scheduling algorithms

• A task is said to be schedulable if and only if its **worst-case response time** is not greater than its deadline

• When is the worst-case?
Exact Test

• A.k.a. Response time analysis

• For fixed-priority scheduling algorithms

• A task is said to be schedulable if and only if its **worst-case response time** is not greater than its deadline

• When is the worst-case?
  • When all higher-priority tasks are released at the same time (‘**Critical instant theorem**’ [Liu73])

Exact Test

\[ r_{i}^{k+1} = C_i + \sum_{j=1}^{i-1} \left[ \frac{r_{i}^{k}}{p_{j}} \right] C_{j} \]

where \( r_{i}^{0} = \sum_{j=1}^{i} C_{j} \)

- Iterative method
- Tasks are ordered according to their priority; \( \tau_1 \) has the highest priority
- If \( r_{i}^{k+1} > D_i \) -> Unschedulable
- If \( r_{i}^{k+1} = r_{i}^{k} \leq D_i \) for some \( k \) -> Schedulable
- Test task-by-task. If any task fails the exact test, the task set is unschedulable
Exact Test

\[ r_{i}^{k+1} = C_i + \sum_{j=1}^{i-1} \left[ \frac{r_j^k}{p_j} \right] C_j \quad \text{and} \quad r_i^0 = \sum_{j=1}^{i} C_j \]

\( \tau_1 := (p_1 = 10, c_1 = 4) \)
\( \tau_2 := (p_2 = 15, c_2 = 4) \)
\( \tau_3 := (p_3 = 35, c_3 = 10) \)

Released at time 0
Exact Test

\[ r_i^{k+1} = C_i + \sum_{j=1}^{i-1} \left[ \frac{r_i^k}{p_j} \right] C_j, \quad r_i^0 = \sum_{j=1}^{i} C_j \]

\( r_0 = \tau_1 = (p_1 = 10, c_1 = 4) \)

\( \tau_1 := (p_1 = 10, c_1 = 4) \)

\( \tau_2 := (p_2 = 15, c_2 = 4) \)

\( \tau_3 := (p_3 = 35, c_3 = 10) \)

\[ r_3^0 = 4 + 4 + 10 = 18 \]

Time

\[ 0 \quad 10 \quad 15 \quad \text{Time} \]
Exact Test

\[ r_{i}^{k+1} = C_i + \sum_{j=1}^{i-1} \left[ \frac{r_{i}^{k}}{p_j} \right] C_j \quad , \quad r_{i}^{0} = \sum_{j=1}^{i} C_j \]

\( \tau_1 := (p_1 = 10, c_1 = 4) \)

\( \tau_2 := (p_2 = 15, c_2 = 4) \)

\( \tau_3 := (p_3 = 35, c_3 = 10) \)

New jobs of Task 1 and 2 arrive before Task 3’s job finishes. -> Additional preemptions
Exact Test

\[
\tau_1 := (p_1 = 10, c_1 = 4)
\]

\[
\tau_2 := (p_2 = 15, c_2 = 4)
\]

\[
\tau_3 := (p_3 = 35, c_3 = 10)
\]

\[
r_{i+1} = C_i + \sum_{j=1}^{i-1} \left\lfloor \frac{r^k_i}{p_j} \right\rfloor C_j
\]

\[
r_i^0 = \sum_{j=1}^{i} C_j
\]

\[
r_3^1 = 10 + \sum_{j=1}^{2} \left\lfloor \frac{r_3^0}{p_j} \right\rfloor C_j = 10 + \left\lfloor \frac{18}{10} \right\rfloor 4 + \left\lfloor \frac{18}{15} \right\rfloor 4 = 10 + 2 \cdot 4 + 2 \cdot 4 = 10 + 8 + 8 = 26
\]
Exact Test

\[ r_i^{k+1} = C_i + \sum_{j=1}^{i-1} \left[ \frac{r_i^k}{p_j} \right] C_j, \quad r_i^0 = \sum_{j=1}^{i} C_j \]

\[ \tau_1 := (p_1 = 10, c_1 = 4) \]

\[ \tau_2 := (p_2 = 15, c_2 = 4) \]

\[ \tau_3 := (p_3 = 35, c_3 = 10) \]

New job of Task 1 arrives before Task 3’s job finishes. -> Additional preemption
**Exact Test**

\[ r_i^{k+1} = C_i + \sum_{j=1}^{i-1} \left[ \frac{r_i^k}{p_j} \right] C_j, \quad r_i^0 = \sum_{j=1}^{i} C_j \]

\( \tau_1 := (p_1 = 10, c_1 = 4) \)

\( \tau_2 := (p_2 = 15, c_2 = 4) \)

\( \tau_3 := (p_3 = 35, c_3 = 10) \)

\( r_3^1 = 26 \)

**Q:** Compute \( r_3^2 \)
Exact Test

\[ r_i^{k+1} = C_i + \sum_{j=1}^{i-1} \left\lfloor \frac{r_i^k}{p_j} \right\rfloor C_j \quad , \quad r_i^0 = \sum_{j=1}^{i} C_j \]

\[ r_1^2 = 10 + \sum_{j=1}^{2} \left\lfloor \frac{10}{p_j} \right\rfloor C_j = 10 + \left\lfloor \frac{10}{10} \right\rfloor 4 + \left\lfloor \frac{26}{15} \right\rfloor 4 = 10 + 4 + \frac{26}{4} - 3 \cdot 4 + 2 \cdot 4 = 10 + 12 + 8 = 30 \]

\( \tau_1 := (p_1 = 10, c_1 = 4) \)

\( \tau_2 := (p_2 = 15, c_2 = 4) \)

\( \tau_3 := (p_3 = 35, c_3 = 10) \)
Exact Test

\[
\begin{align*}
 r_{i}^{k+1} &= C_{i} + \sum_{j=1}^{i-1} \left[ \frac{r_{i}^{k}}{p_{j}} \right] C_{j} \\
 r_{i}^{0} &= \sum_{j=1}^{i} C_{j}
\end{align*}
\]

\[
\begin{align*}
 r_{1}^{3} &= 10 + \sum_{j=1}^{2} \left[ \frac{r_{1}^{3}}{p_{j}} \right] C_{j} = 10 + \left[ \frac{26}{10} \right] 4 + \left[ \frac{26}{15} \right] 4 = 10 + 2.6 + 1.73 = 10 + 3 \cdot 4 + 2 \cdot 4 = 10 + 12 + 8 = 30 \\
 r_{2}^{3} &= 10 + \sum_{j=1}^{2} \left[ \frac{r_{2}^{3}}{p_{j}} \right] C_{j} = 10 + \left[ \frac{30}{10} \right] 4 + \left[ \frac{30}{15} \right] 4 = 10 + 3 \cdot 4 + 2 \cdot 4 = 10 + 12 + 8 = 30
\end{align*}
\]

\[
\begin{align*}
 \tau_{1} := (p_{1} = 10, c_{1} = 4) \\
 \tau_{2} := (p_{2} = 15, c_{2} = 4) \\
 \tau_{3} := (p_{3} = 35, c_{3} = 10)
\end{align*}
\]
Exact Test

\[ r_i^{k+1} = C_i + \sum_{j=1}^{i-1} \left[ \frac{r_i^k}{p_j} \right] C_j, \quad r_i^0 = \sum_{j=1}^{i} C_j \]

\( \tau_1 := (p_1 = 10, c_1 = 4) \)

\( \tau_2 := (p_2 = 15, c_2 = 4) \)

\( \tau_3 := (p_3 = 35, c_3 = 10) \)

\[ r_3^2 = 10 + \sum_{j=1}^{2} \left[ \frac{r_3^1}{p_j} \right] C_j = 10 + \frac{26}{10} 4 + \frac{26}{15} 4 = 10 + 3 \cdot 4 + 2 \cdot 4 = 10 + 12 + 8 = 30 \]

\[ r_3^3 = 10 + \sum_{j=1}^{2} \left[ \frac{r_3^2}{p_j} \right] C_j = 10 + \frac{30}{10} 4 + \frac{30}{15} 4 = 10 + 3 \cdot 4 + 2 \cdot 4 = 10 + 12 + 8 = 30 \]

Worst-case Response Time (=30) < Deadline (=35)
Utilization Bound Test

Task Utilization

\[ U_i = \frac{C_i}{p_i} \]

Processor Utilization \((n=\text{number of tasks})\)

\[ U = \sum_{i=1}^{n} U_i = \sum_{i=1}^{n} \frac{C_i}{p_i} \]

Utilization Bound \((U_b)\)

Any task \( \tau_i \in \{\tau_1, \tau_2, \ldots, \tau_n\} \) is guaranteed to be schedulable if \( U \leq U_b \)

\( U_b \) depends on the scheduling algorithm, # of tasks, availability on timing information, ...
RM Utilization Bound

A set of $n$ tasks is schedulable under RM scheduling if (see [Liu73] for proof)

$$U \leq U_{RM}(n) = n(2^{1/n} - 1)$$

Example

<table>
<thead>
<tr>
<th>Task</th>
<th>$C_i$ (Execution Time)</th>
<th>$p_i$ (Period)</th>
<th>$U_i$ (Utilization)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task 1</td>
<td>20</td>
<td>100</td>
<td>?</td>
</tr>
<tr>
<td>Task 2</td>
<td>40</td>
<td>150</td>
<td>?</td>
</tr>
<tr>
<td>Task 3</td>
<td>100</td>
<td>350</td>
<td>?</td>
</tr>
</tbody>
</table>

RM Utilization Bound

A set of $n$ tasks is schedulable under RM scheduling if (see [Liu73] for proof)

$$U \leq U_{RM}(n) = n(2^{1/n} - 1)$$

Example

<table>
<thead>
<tr>
<th>C_i (Execution Time)</th>
<th>p_i (Period)</th>
<th>U_i (Utilization)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task 1</td>
<td>20</td>
<td>100</td>
</tr>
<tr>
<td>Task 2</td>
<td>40</td>
<td>150</td>
</tr>
<tr>
<td>Task 3</td>
<td>100</td>
<td>350</td>
</tr>
</tbody>
</table>

1) Check the schedulability of {task 1}:

$$U_1 = 0.2 < U_{RM}(1) = 1$$
RM Utilization Bound

A set of $n$ tasks is schedulable under RM scheduling if (see [Liu73] for proof)

$$U \leq U_{RM}(n) = n(2^{1/n} - 1)$$

Example

<table>
<thead>
<tr>
<th></th>
<th>$C_i$ (Execution Time)</th>
<th>$p_i$ (Period)</th>
<th>$U_i$ (Utilization)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task 1</td>
<td>20</td>
<td>100</td>
<td>0.200</td>
</tr>
<tr>
<td>Task 2</td>
<td>40</td>
<td>150</td>
<td>0.267</td>
</tr>
<tr>
<td>Task 3</td>
<td>100</td>
<td>350</td>
<td>0.286</td>
</tr>
</tbody>
</table>

2) Check the schedulability of \{task 1, task 2\}:

$$U_1 + U_2 \approx 0.467 < U_{RM}(2) = 0.828$$
RM Utilization Bound

A set of $n$ tasks is schedulable under RM scheduling if (see [Liu73] for proof)

$$U \leq U_{RM}(n) = n(2^{1/n} - 1)$$

Example

<table>
<thead>
<tr>
<th>Task</th>
<th>$C_i$ (Execution Time)</th>
<th>$p_i$ (Period)</th>
<th>$U_i$ (Utilization)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task 1</td>
<td>20</td>
<td>100</td>
<td>0.200</td>
</tr>
<tr>
<td>Task 2</td>
<td>40</td>
<td>150</td>
<td>0.267</td>
</tr>
<tr>
<td>Task 3</td>
<td>100</td>
<td>350</td>
<td>0.286</td>
</tr>
</tbody>
</table>

3) Check the schedulability of {task 1, task 2, task 3}:

$$U_1 + U_2 + U_3 \approx 0.753 < U_{RM}(3) = 0.780$$
RM Utilization Bound

A set of \( n \) tasks is schedulable under RM scheduling if (see [Liu73] for proof)

\[
U \leq U_{RM}(n) = n\left(2^{1/n} - 1\right)
\]

Example

<table>
<thead>
<tr>
<th></th>
<th>( C_i ) (Execution Time)</th>
<th>( p_i ) (Period)</th>
<th>( U_i ) (Utilization)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task 1</td>
<td>40</td>
<td>100</td>
<td>0.400</td>
</tr>
<tr>
<td>Task 2</td>
<td>40</td>
<td>150</td>
<td>0.267</td>
</tr>
<tr>
<td>Task 3</td>
<td>100</td>
<td>350</td>
<td>0.286</td>
</tr>
</tbody>
</table>

Q: What if \( C_1 = 40 \)?
RM Utilization Bound

A set of $n$ tasks is schedulable under RM scheduling if (see [Liu73] for proof)

$$U \leq U_{RM}(n) = n(2^{1/n} - 1)$$

Example

<table>
<thead>
<tr>
<th></th>
<th>$C_i$ (Execution Time)</th>
<th>$p_i$ (Period)</th>
<th>$U_i$ (Utilization)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task 1</td>
<td>40</td>
<td>100</td>
<td>0.400</td>
</tr>
<tr>
<td>Task 2</td>
<td>40</td>
<td>150</td>
<td>0.267</td>
</tr>
<tr>
<td>Task 3</td>
<td>100</td>
<td>350</td>
<td>0.286</td>
</tr>
</tbody>
</table>

Q: What if $C_1 = 40$?

$$U_1 + U_2 + U_3 \approx 0.953 > U_{RM}(3) = 0.780$$
RM Utilization Bound

A set of \( n \) tasks is schedulable under RM scheduling if (see [Liu73] for proof)

\[
U \leq U_{RM}(n) = n(2^{1/n} - 1)
\]

Example

<table>
<thead>
<tr>
<th>( C_i ) (Execution Time)</th>
<th>( p_i ) (Period)</th>
<th>( U_i ) (Utilization)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task 1</td>
<td>40</td>
<td>100</td>
</tr>
<tr>
<td>Task 2</td>
<td>40</td>
<td>150</td>
</tr>
<tr>
<td>Task 3</td>
<td>100</td>
<td>350</td>
</tr>
</tbody>
</table>

Q: What if \( C_1 = 40 \)?

\[ U_1 + U_2 + U_3 \approx 0.953 > U_{RM}(3) = 0.780 \]

Q: Are the tasks unschedulable?
RM Utilization Bound

A set of \( n \) tasks is schedulable under RM scheduling if (see [Liu73] for proof)

\[
U \leq U_{RM}(n) = n(2^{1/n} - 1)
\]

Example

<table>
<thead>
<tr>
<th>( C_i ) (Execution Time)</th>
<th>( p_i ) (Period)</th>
<th>( U_i ) (Utilization)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task 1</td>
<td>40</td>
<td>100</td>
</tr>
<tr>
<td>Task 2</td>
<td>40</td>
<td>150</td>
</tr>
<tr>
<td>Task 3</td>
<td>100</td>
<td>350</td>
</tr>
</tbody>
</table>

Q: What if \( C_1 = 40 \)?

\[ U_1 + U_2 + U_3 \approx 0.953 > U_{RM}(3) = 0.780 \]

Q: Are the tasks unschedulable?

A: Not necessarily. Need to do the exact test!
RM Utilization Bound

A set of \( n \) tasks is schedulable under RM scheduling if (see [Liu73] for proof)

\[
U \leq U_{RM}(n) = n(2^{1/n} - 1)
\]

Example

<table>
<thead>
<tr>
<th>Task</th>
<th>( C_i ) (Execution Time)</th>
<th>( p_i ) (Period)</th>
<th>( U_i ) (Utilization)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task 1</td>
<td>40</td>
<td>100</td>
<td>0.400</td>
</tr>
<tr>
<td>Task 2</td>
<td>40</td>
<td>150</td>
<td>0.267</td>
</tr>
<tr>
<td>Task 3</td>
<td>100</td>
<td>350</td>
<td>0.286</td>
</tr>
</tbody>
</table>

Q: What is the worst-case response time of Task 3?

\[
r_{i}^{k+1} = C_{i} + \sum_{j=1}^{i-1} \left[ \frac{r_{i}^{k}}{p_{j}} \right] C_{j}, \quad r_{i}^{0} = \sum_{j=1}^{i} C_{j}
\]
RM Utilization Bound

Utilization bound test is a \textit{sufficient} condition

- If $U \leq U_{RM}(n)$, the task set is guaranteed to be schedulable by RM.
- $U > U_{RM}(n)$ does not necessarily mean the task set is unschedulable
- Need to perform an exact test

UB for any $n$

$$U_{RM} = \lim_{n \to \infty} U_{RM}(n) = \ln 2 \approx 0.693$$

Q: What does this mean?
RM Utilization Bound

Utilization bound test is a **sufficient** condition
- If $U \leq U_{RM}(n)$, the task set is guaranteed to be schedulable by RM.
- $U > U_{RM}(n)$ does not necessarily mean the task set is unschedulable
  - Need to perform an exact test

**UB for any $n$**

$$U_{RM} = \lim_{n \to \infty} U_{RM}(n) = \ln 2 \approx 0.693$$

- That is, any task set is schedulable if $U \leq U_{RM}$
EDF Utilization Bound

A set of tasks is schedulable under EDF scheduling if and only if

\[ U \leq U_{EDF} = 1 \]

- Sufficient and necessary condition
- Does not depend on # of tasks

<table>
<thead>
<tr>
<th></th>
<th>( C_i ) (Execution Time)</th>
<th>( p_i ) (Period)</th>
<th>( U_i ) (Utilization)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task 1</td>
<td>40</td>
<td>100</td>
<td>0.400</td>
</tr>
<tr>
<td>Task 2</td>
<td>40</td>
<td>150</td>
<td>0.267</td>
</tr>
<tr>
<td>Task 3</td>
<td>100</td>
<td>350</td>
<td>0.286</td>
</tr>
</tbody>
</table>

\[ U_1 + U_2 + U_3 \approx 0.953 < U_{EDF} \]
RM vs EDF

EDF’s utilization bound is 1 while RM’s is less than 1
- RM may not fully utilize the CPU

Why do we need RM?
- Simpler implementation
  - Priorities do not change
  - Some tasks may not have deadlines

- EDF is unpredictable
  - Domino effect during overloaded situation
  - A low critical task which overruns but has an earlier deadline can delay a high critical task.
  - FAA (Federal Aviation Administration) and EASA (European Aviation Safety Agency) forbid the use of EDF
Priority Inversion

So far, tasks are assumed to be independent

What if tasks share data?
  ◦ Synchronization!

  ◦ But it can be a source of priority inversion

```c
semaphore->P();  // critical section goes here
semaphore->V();
```

A few definitions

- Synchronization:
  - using atomic operations to ensure cooperation between threads
- Mutual exclusion:
  - ensuring that only one thread does a particular thing at a time. One thread doing it excludes the other, and vice versa.
- Critical section:
  - piece of code that only one thread can execute at once. Only one thread at a time will get into the section of code.
- Lock: prevents someone from doing something
  - lock before entering critical section, before accessing shared data
  - unlock when leaving, after done accessing shared data
  - wait if locked

How to use semaphores

- Binary semaphores can be used for mutual exclusion:
  - initial value of 1; P() is called before the critical section, and V() is called after the critical section.
```c
semaphore->P();  // critical section goes here
semaphore->V();
```

- Scheduling constraints
  - having one thread to wait for something to happen
    - Example: Thread 2, which must wait for a thread to terminate. By setting the initial value to 0 instead of 1, we can implement waiting on a semaphore.

- Controlling access to a finite resource.

Looks familiar? Lecture 6-9
Priority Inversion

When a high priority task is delayed by a low priority task
Priority Inversion

When a high priority task is delayed by a low priority task.
Priority Inversion

When a high priority task is delayed by a low priority task

High Priority

Low Priority

Time
Priority Inversion

When a high priority task is delayed by a low priority task

Diagram:
- High Priority
  - Release
  - P()
  - V()
  - Priority Inversion

- Low Priority
  - Release
  - P()
  - V()

Time
Unbounded Priority Inversion

Time

High Priority

Medium Priority

Medium Priority

Low Priority

Normal Execution

Critical Section
Unbounded Priority Inversion

- **High Priority**
  - Normal Execution
  - Critical Section

- **Medium Priority**
  - Normal Execution

- **Low Priority**
  - Normal Execution

Time
Unbounded Priority Inversion

- **High Priority**
  - Normal Execution
  - Critical Section

- **Medium Priority**
  - Normal Execution
  - Critical Section

- **Low Priority**
  - Normal Execution
  - Critical Section
Unbounded Priority Inversion

- **High Priority**:\[P()\] - Normal Execution
- **Medium Priority**:\[P()\] - Critical Section
- **Low Priority**:\[P()\] - Critical Section

Time
Unbounded Priority Inversion

- **High Priority**
  - $P()$

- **Medium Priority**
  - $P()$

- **Low Priority**
  - $P()$
Unbounded Priority Inversion

- **High Priority**: 
  - P() 
  - V()

- **Medium Priority**: 
  - P() 

- **Low Priority**: 
  - P() 
  - V()
Unbounded Priority Inversion

Priority inversion is unbounded due to preemptions by medium priority tasks.
Unbounded Priority Inversion

It actually happened on Mars!

NASA Mars Pathfinder (1997)

What really happened on Mars?

From: Mike Jones <mjk@msn.com>
Sent: Sunday, December 27, 1997 8:43 pm
Subject: What really happened on Mars?

THE PROBLEM

The Mars Pathfinder mission was widely proclaimed as “flawless” in the early days after its July 4th, 1997 landing on the Martian surface. Successes included its unconventional “landing” — bouncing onto the Martian surface surrounded by airbags, deploying the Sojourner rover, and gathering and transmitting voluminous data back to Earth, including the panoramic pictures that were such a hit on the Web. But a few days into the mission, not long after Pathfinder started gathering meteorological data, the spacecraft began experiencing total system resets, each resulting in losses of data. The press reported these failures in terms such as “software glitches” and “the computer was trying to do too many things at once.”

This week at the IEEE Real-Time Systems Symposium I heard a fascinating keynote address by David Winder, Chief Technical Officer of Wind River Systems. Wind River makes VxWorks, the real-time embedded systems kernel that was used in the Mars Pathfinder mission. In his talk, he explained in detail voluminous data back to Earth, including the panoramic pictures that were such a hit on the Web. But a few days into the mission, not long after Pathfinder started gathering meteorological data, the spacecraft began experiencing total system resets, each resulting in losses of data. The press reported these failures.
Unbounded Priority Inversion

It actually happened on Mars!

What really happened on Mars?

From: Mike Jones <mike@microsoft.com>
Sent: Sunday, December 27, 1997 8:47 pm
Subject: What really happened on Mars?

THE PROBLEM

The Mars Pathfinder mission was widely proclaimed as "flawless" in the early days after its July 4th, 1997 landing on the Martian surface. Successes included its unconventional "landing" — bouncing onto the Martian surface surrounded by airbags, deploying the Sojourner rover, and gathering and transmitting voluminous data back to Earth, including the panoramic pictures that were such a hit on the Web. But a few days into the mission, not long after Pathfinder started gathering meteorological data, the spacecraft began experiencing total system resets, each resulting in losses of data. The press reported these failures in terms such as "software glitches" and "the computer was trying to do too many things at once".

This week I attended the IEEE Real-Time Systems Symposium where I heard a fascinating keynote address by David Winder, Technical Director of Wind River Systems. Wind River makes VxWorks, the real-time operating systems kernel that was used in the Mars Pathfinder mission. In his talk, he explained in detail how the system was designed and why it failed.

Sufficiently voluminous data back to Earth, including the panoramic pictures that were such a hit on the Web. But a few days into the mission, not long after Pathfinder started gathering meteorological data, the spacecraft began experiencing total system resets, each resulting in losses of data. The press reported these failures...
Unbounded Priority Inversion

It actually happened on Mars!

NASA Mars Pathfinder (1997)

What really happened on Mars?

From: Mike Jones <mike@microsoft.com>
Sent: Sunday, December 07, 1997 6:47 pm
Subject: What really happened on Mars?

THE PROBLEM

The Mars Pathfinder mission was widely proclaimed as "flawless" in the early days after its July 4th, 1997 landing on the Martian surface. Successes included its unconventional "landing" — bouncing onto the Martian surface surrounded by airbags, deploying the Sojourner rover, and gathering and transmitting voluminous data back to Earth, including the panoramic pictures that were such a hit on the Web. But a few days into the mission, not long after Pathfinder started gathering meteorological data, the spacecraft began experiencing total system resets, each resulting in losses of data. The press reported these failures in terms such as "software glitches" and "the computer was trying to do too many things at once".

This week at the IEEE Real-Time Systems Symposium I heard a fascinating keynote address by David Wintle, Head Technical Officer of Wind River Systems. Wind River makes VxWorks, the real-time embedded systems kernel that was used in the Mars Pathfinder mission. In his talk, he explained in detail voluminous data back to Earth, including the panoramic pictures that were such a hit on the Web. But a few days into the mission, not long after Pathfinder started gathering meteorological data, the spacecraft began experiencing total system resets, each resulting in losses of data. The press reported these failures in terms such as "software glitches" and "the computer was trying to do too many things at once".

1) \( L \) is executing, accessing the bus.
2) \( H \) can’t access the bus. It is blocked by \( L \)
3) \( M \) preempts \( L \), so \( H \) is further blocked.
4) Watchdog timer notices that \( H \) has not executed for some time. Hence, it resets the system!
Unbounded Priority Inversion

It actually happened on Mars!

How was the problem corrected?

“Priority Inheritance Protocol (PIP)”
VxWorks had PIP, but it had been turned off for the mutex!


1) **L** is executing, accessing the bus.
2) **H** can’t access the bus. It is blocked by **L**.
3) **M** preempts **L**, so **H** is further blocked.
4) Watchdog timer notices that **H** has not executed for some time. Hence, it resets the system!
Priority Inheritance Protocol

Low priority task inherits the highest priority of all the blocked tasks
- This keeps medium tasks from delaying the low priority task that is in a critical section

Time

Task 1 (High Priority)

Task 2 (Medium Priority)

Task 3 (Low Priority)
Priority Inheritance Protocol

Low priority task inherits the highest priority of all the blocked tasks
• This keeps medium tasks from delaying the low priority task that is in a critical section
Priority Inheritance Protocol

Low priority task inherits the highest priority of all the blocked tasks
- This keeps medium tasks from delaying the low priority task that is in a critical section

Task 1 (High Priority)

Task 2 (Medium Priority)

Task 3 (Low Priority)

Task 2 can’t preempt Task 3!
Priority Inheritance Protocol

Low priority task inherits the highest priority of all the blocked tasks
- This keeps medium tasks from delaying the low priority task that is in a critical section
Priority Inheritance Protocol

Low priority task inherits the highest priority of all the blocked tasks

- This keeps medium tasks from delaying the low priority task that is in a critical section
Priority Inheritance Protocol

A job $J$ can be blocked for at most $\min(n,m)$ times where

- $n =$ number of lower priority jobs that could block $J$
- $m =$ number of distinct semaphores that can be used to block $J$

But chained blocking and deadlock can happen under PIP

- Solution: Priority Ceiling Protocol (PCP)
Priority Ceiling Protocol

**Priority ceiling** of a semaphore
- The priority of the highest priority task that may use the semaphore

**Key Idea**
- A job J is allowed to enter a critical section only if its priority is higher than all priority ceilings of the semaphores currently locked by jobs other than J
  - Thus, it can never be blocked by lower priority jobs until its completion!
- When a job gets a semaphore, PCP guarantees that this job will get all the semaphores that it ever needs.
- Hence, PCP prevents chained blocking and deadlock.

For more information, see
Interested in Research?

- Self-driving cars
- Real-time + machine learning
- Internet of things
- Real-time scheduling
  
  ...  

For research opportunities (for senior project), feel free to contact

- Dr. Man-Ki Yoon (man-ki.yoon@yale.edu)
- Dr. Jung-Eun Kim (jung-eun.kim@yale.edu)
- Prof. Zhong Shao (zhong.shao@yale.edu)