Real-Time Systems II

Alan Burns
University of York, UK
Simple Process Model

- The application is assumed to consist of a fixed set of processes
- All processes are periodic, with known periods
- The processes are completely independent of each other
- All system's overheads, context-switching times and so on are ignored (i.e., assumed to have zero cost)
- All processes have a deadline equal to their period (that is, each process must complete before it is next released)
- All processes have a fixed worst-case execution time
Standard Notation

B  Worst-case blocking time
C  Worst-case computation time (WCET)
D  Deadline of the process
I  The interference time of the process
J  Release jitter of the process
N  Number of processes in the system
P  Priority assigned to the process (if applicable)
R  Worst-case response time of the process
T  Minimum time between process releases
U  The utilization of each process (equal to C/T)
Fixed-Priority Scheduling (FPS)

- This is the most widely used approach and is the main focus of this course.

- Each process has a fixed, *static*, priority which is determined computer pre-run-time.

- The runnable processes are executed in the order determined by their priority.

- In real-time systems, the “priority” of a process is derived from its temporal requirements, not its importance to the correct functioning of the system or its integrity.
Each process is assigned a (unique) priority based on its period; the shorter the period, the higher the priority.

i.e., for two processes $i$ and $j$,

$$T_i < T_j \Rightarrow P_i > P_j$$

This assignment is optimal in the sense that if any process set can be scheduled (using pre-emptive priority-based scheduling) with a fixed-priority assignment scheme, then the given process set can also be scheduled with a rate monotonic assignment scheme.
Liu and Layland

- Proved rate monotonic optimality
- Showed that the worst-case situation is when all processes are released at the same time
- Showed that the first release is the worst
- Derived the utilisation-based test
Example Priority Assignment

<table>
<thead>
<tr>
<th>Process</th>
<th>Period, T</th>
<th>Priority, P</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>25</td>
<td>5</td>
</tr>
<tr>
<td>b</td>
<td>60</td>
<td>3</td>
</tr>
<tr>
<td>c</td>
<td>42</td>
<td>4</td>
</tr>
<tr>
<td>d</td>
<td>105</td>
<td>1</td>
</tr>
<tr>
<td>e</td>
<td>75</td>
<td>2</td>
</tr>
</tbody>
</table>
Utilisation-Based Analysis

- For D=T task sets only
- A simple sufficient but not necessary schedulability test exists

\[ U \equiv \sum_{i=1}^{N} \frac{C_i}{T_i} \leq N \left(2^{1/N} - 1\right) \]

\[ U \leq 0.69 \quad \text{as} \quad N \rightarrow \infty \]
## Utilization Bounds

<table>
<thead>
<tr>
<th>N</th>
<th>Utilization bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100.0%</td>
</tr>
<tr>
<td>2</td>
<td>82.8%</td>
</tr>
<tr>
<td>3</td>
<td>78.0%</td>
</tr>
<tr>
<td>4</td>
<td>75.7%</td>
</tr>
<tr>
<td>5</td>
<td>74.3%</td>
</tr>
<tr>
<td>10</td>
<td>71.8%</td>
</tr>
</tbody>
</table>

Approaches 69.3% asymptotically
## Process Set A

<table>
<thead>
<tr>
<th>Process</th>
<th>Period T</th>
<th>Computation Time C</th>
<th>Priority P</th>
<th>Utilization U</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>80</td>
<td>40</td>
<td>1</td>
<td>0.50</td>
</tr>
<tr>
<td>b</td>
<td>40</td>
<td>10</td>
<td>2</td>
<td>0.25</td>
</tr>
<tr>
<td>c</td>
<td>20</td>
<td>5</td>
<td>3</td>
<td>0.25</td>
</tr>
</tbody>
</table>

- The combined utilization is 1.0
- This is above the threshold for three processes (0.78) but the process set will meet all its deadlines
Time-line for Process Set A

Process

a

b

c

Time

0 10 20 30 40 50 60 70 80
Criticism of Utilisation-based Tests

- Not exact
- Not general
- BUT it is $O(N)$

The test is said to be sufficient but not necessary.
Response-Time Analysis

- Here task $i$'s worst-case response time, $R_i$, is calculated first and then checked (trivially) with its deadline

$$R_i \leq D_i$$

$$R_i = C_i + I_i$$

Where $I$ is the interference from higher priority tasks
Calculating R

During R, each higher priority task \( j \) will execute a number of times:

\[
\text{Number of Releases} = \left\lceil \frac{R_i}{T_j} \right\rceil
\]

The ceiling function \( \lceil \cdot \rceil \) gives the smallest integer greater than the fractional number on which it acts. So the ceiling of 1/3 is 1, of 6/5 is 2, and of 6/3 is 2.

Total interference is given by:

\[
\left\lceil \frac{R_i}{T_j} \right\rceil C_j
\]
Response Time Equation

\[ R_i = C_i + \sum_{j \in hp(i)} \left[ \frac{R_i}{T_j} \right] C_j \]

Where \( hp(i) \) is the set of tasks with priority higher than task \( i \)

Solve by forming a recurrence relationship:

\[ w_i^{n+1} = C_i + \sum_{j \in hp(i)} \left[ \frac{w_i^n}{T_j} \right] C_j \]

The set of values \( w_i^0, w_i^1, w_i^2, \ldots, w_i^n, \ldots \) is monotonically non-decreasing. When \( w_i^n = w_i^{n+1} \) the solution to the equation has been found; \( w_i^0 \) must not be greater than \( R_i \) (e.g. 0 or \( C_i \))
### Process Set B

<table>
<thead>
<tr>
<th>Process</th>
<th>Period</th>
<th>ComputationTime</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>7</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>b</td>
<td>12</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>c</td>
<td>20</td>
<td>5</td>
<td>1</td>
</tr>
</tbody>
</table>

\[
R_a = 3
\]

\[
R_b = 6
\]

\[
w^0_b = 3
\]

\[
w^1_b = 3 + \left[ \frac{3}{7} \right] 3 = 6
\]

\[
w^2_b = 3 + \left[ \frac{6}{7} \right] 3 = 6
\]

\[
R_b = 6
\]
\[ w_c^0 = 5 \]
\[ w_c^1 = 5 + \left[ \frac{5}{7} \right] 3 + \left[ \frac{5}{12} \right] 3 = 11 \]
\[ w_c^2 = 5 + \left[ \frac{11}{7} \right] 3 + \left[ \frac{11}{12} \right] 3 = 14 \]
\[ w_c^3 = 5 + \left[ \frac{14}{7} \right] 3 + \left[ \frac{14}{12} \right] 3 = 17 \]
\[ w_c^4 = 5 + \left[ \frac{17}{7} \right] 3 + \left[ \frac{17}{12} \right] 3 = 20 \]
\[ w_c^5 = 5 + \left[ \frac{20}{7} \right] 3 + \left[ \frac{20}{12} \right] 3 = 20 \]
\[ R_c = 20 \]
### Revisit: Process Set A

<table>
<thead>
<tr>
<th>Process</th>
<th>Period</th>
<th>Computation Time</th>
<th>Priority</th>
<th>Response time</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>80</td>
<td>40</td>
<td>1</td>
<td>80</td>
</tr>
<tr>
<td>b</td>
<td>40</td>
<td>10</td>
<td>2</td>
<td>15</td>
</tr>
<tr>
<td>c</td>
<td>20</td>
<td>5</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>

- **The combined utilization is 1.0**
- **This was above the utilization threshold for three processes (0.78), therefore it failed the test**
- **The response time analysis shows that the process set will meet all its deadlines**
Response Time Analysis

- **Is sufficient and necessary**

- If the process set passes the test they will meet all their deadlines; if they fail the test then, at run-time, a process will miss its deadline (unless the computation time estimations themselves turn out to be pessimistic)
Process Interactions / Blocking

- If a process is suspended waiting for a lower-priority process to complete some required computation then the priority model is, in some sense, being undermined

- It is said to suffer priority inversion

- If a process is waiting for a lower-priority process, it is said to be blocked
Priority Inversion

To illustrate an extreme example of priority inversion, consider the executions of four periodic processes: \(a\), \(b\), \(c\) and \(d\); and two resources: \(Q\) and \(V\)

<table>
<thead>
<tr>
<th>Process</th>
<th>Priority</th>
<th>Execution Sequence</th>
<th>Release Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td>EQQQE</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>2</td>
<td>EE</td>
<td>2</td>
</tr>
<tr>
<td>c</td>
<td>3</td>
<td>EVVE</td>
<td>2</td>
</tr>
<tr>
<td>d</td>
<td>4</td>
<td>EEQVE</td>
<td>4</td>
</tr>
</tbody>
</table>
Example of Priority Inversion

Process

- **d**: Executing with V locked
- **c**: Executing with Q locked
- **b**
- **a**: Executing

Timeline:
- 0 to 2: Executing
- 2 to 4: Preempted
- 4 to 6: Executing with V locked
- 6 to 8: Executing
- 8 to 10: Executing with Q locked
- 10 to 12: Executing
- 12 to 14: Executing with V locked
- 14 to 16: Executing
- 16 to 18: Preempted
Priority Inheritance

- If process $p$ is blocking process $q$, then $q$ runs with $p$'s priority
Mars Path-Finder

- A problem due to priority inversion nearly caused the lose of the Mars Path-finder mission

- As a shared bus got heavily loaded critical data was not been transferred

- Time-out on this data was used as an indication of failure and lead to re-boot

- Solution was a patch that turned on priority inheritance, this solved the problem
Response Time and Blocking

\[ R_i = C_i + B_i + I_i \]

\[ R_i = C_i + B_i + \sum_{j \in hp(i)} \left[ \frac{R_i}{T_j} \right] C_j \]

\[ w_i^{n+1} = C_i + B_i + \sum_{j \in hp(i)} \left[ \frac{w_i^n}{T_j} \right] C_j \]
Priority Ceiling Protocols

Two forms

- Original ceiling priority protocol
- Immediate ceiling priority protocol
On a Single Processor

- A high-priority process can be blocked at most once during its execution by lower-priority processes.
- Deadlocks are prevented.
- Transitive blocking is prevented.
- Mutual exclusive access to resources is ensured (by the protocol itself).
ICPP

- Each process has a static default priority assigned (perhaps by the deadline monotonic scheme)

- Each resource has a static ceiling value defined, this is the maximum priority of the processes that use it

- A process has a dynamic priority that is the maximum of its own static priority and the ceiling values of any resources it has locked

- As a consequence, a process will only suffer a block at the very beginning of its execution

- Once the process starts actually executing, all the resources it needs must be free; if they were not, then some process would have an equal or higher priority and the process's execution would be postponed
ICPP Inheritance

Process

- d
- c
- b
- a
Summary

- Rate monotonic analysis
- Response Time analysis
- Priority inversion and inheritance
- Priority ceiling protocols