Real Time Operating Systems

Shared Resources

Luca Abeni

luca.abeni@unitn.it
Interacting Tasks

- Until now, only independent tasks...
  - A job never blocks or suspends
  - A task only blocks on job termination

- In real world, jobs might block for various reasons:
  - Tasks exchange data through shared memory → mutual exclusion
  - A task might need to synchronize with other tasks while waiting for some data
  - A job might need a hardware resource which is currently not available
Interacting Tasks - Example

- Example: control application composed by three periodic tasks
  - $\tau_1$ reads the data from the sensors and applies a filter. The results are stored in memory
  - $\tau_2$ reads the filtered data and computes some control law (updating the state and the outputs); both the state and the outputs are stored in memory
  - $\tau_3$ reads the outputs and writes on an actuator
- All of the three tasks access data in shared memory
- Conflicts on accessing this data concurrently
- $\Rightarrow$ The data structures can become inconsistent
How to handle interactions between tasks?

- Private Resources → Client / Server paradigm
- Shared Resources

Something like “processes vs threads”
Let’s start with processes...

Private Resources

- A Resource Manager (server task) per resource
- Tasks needing to access a resource send a message to its manager
- Interaction via IPC
- Example: the X server
Task Interaction Paradigms - Shared Resources

- What about threads?
- **Shared Resources**
  - Must be accessed in *mutual exclusion*
  - Interaction via mutexes, semaphores, condition variables, . . .
- Real-Time analysis presente here: will focus on shared resources
  - We will use mutexes, not semaphores
  - Extensions to IPC based communication are possible
Resources and Critical Sections

- Shared data structure representing a resource (hw or sw)
- Piece of code accessing the data structure: critical section
  - Critical sections on the same resource must be executed in mutual exclusion
  - Therefore, each data structure should be protected by a mutual exclusion mechanism;
  - This is ok for enforcing data consistency...
  - ...But what is the effect on real-time performance?
    - Assume that resources are protected by mutual exclusion semaphores (mutexes)
    - Why Mutexes and not semaphores? ...
Remember... - Some Definitions

- **Task**
  - Schedulable entity (thread or process)
  - Flow of execution
    - Object Oriented terminology: task $\equiv$ active object
    - Informally speaking: task $\equiv$ active entity that can perform operations on private or shared data

- Now, we need to model the “private or shared data”...
  - As said, focus on **shared** data
Key Concepts - Protected Objects

- Shared data: protected by mutexes ⇒ protected objects
- Protected Objects
  - Encapsulate shared information (Resources)
  - Passive objects (data) shared between different tasks
  - Operations on protected objects are mutually exclusive (this is why they are “protected”!)
- As said, protected by mutexes
  - Locking a mutex, a task “owns” the associate resource...
  - ...So, I can ask: “who is the owner of this resource”?
Shared Resources - Definition

- **Shared Resource** $S_i$
  - Used by multiple tasks
  - Protected by a *mutex* (*mutual exclusion semaphore*)
  - 1 ↔ 1 relationship between resources and mutexes
    - Convention: $S_i$ can be used to indicate either the resource or the mutex
  - The system model must be extended according to this definition
    - Now, the system is not limited to a set of tasks...
Shared Resources - System Model

- System / Application:
  - Set $\mathcal{T}$ of $N$ periodic (or sporadic) tasks:
    \[ \mathcal{T} = \{ \tau_i : 1 \leq i \leq N \} \]
  - Set $S$ of $M$ shared resources:
    \[ S = \{ S_i : 1 \leq i \leq M \} \]
  - Task $\tau_i$ uses resource $S_j$ if it accesses the resource (in a critical section)
  - $k$-th critical section of $\tau_i$ on $S_j$: $cs_{i,j}^k$
  - Length of the longest critical section of $\tau_i$ on $S_j$: $\xi_{i,j}$
```c
pthread_mutex_t m;
...
pthread_mutex_init(&m, NULL);
...
void *tau1(void * arg) {
    pthread_mutex_lock(&m);
    <critical section>
    pthread_mutex_unlock(&m);
}
...

void *tau2(void * arg) {
    pthread_mutex_lock(&m);
    <critical section>
    pthread_mutex_unlock(&m);
}
```

• Mutual exclusion on a shared resource can cause *blocking time*

• When task \( \tau_i \) tries to access a resource \( S \) already held from task \( \tau_j \), \( \tau_i \) blocks

• Blocking time: time between the instant when \( \tau_i \) tries to access \( S \) (and blocks) and the instant when \( \tau_j \) releases \( S \) (and \( \tau_i \) unblocks)

• This is needed for implementing mutual exclusion, and cannot be avoided

• The problem is that this blocking time might become unpredictable/too large...
Blocking times can be particularly bad in priority scheduling if a high priority task wants to access a resource that is held by a lower priority task:

- A low priority task executes, while a high priority one is blocked...
- ...Schedulability guarantees can be compromised!

- Schedulability tests must account for blocking times!
- Blocking times must be deterministic (and not too large!!!)
Consider the following example, where $p_1 > p_2$.

From time 4 to 7, task $\tau_1$ is blocked by a lower priority task $\tau_2$; this is a priority inversion.

This priority inversion is not avoidable; in fact, $\tau_1$ must wait for $\tau_2$ to leave the critical section.

However, in some cases, the priority inversion could be too large.
Example of Priority Inversion

- Consider the following example, with $p_1 > p_2 > p_3$.

- Here, priority inversion is very large: from 4 to 12.
- Problem: while $\tau_1$ is blocked, $\tau_2$ arrives and preempts $\tau_3$ before it can leave the critical section.
- Other medium priority tasks could preempt $\tau_3$ as well...
What Happened on Mars?

- Not only a theoretical problem; it happened for real
- Most (in)famous example: Mars Pathfinder

A small robot, the Sojourner rover, was sent to Mars to explore the martian environment and collect useful information. The on-board control software consisted of many software threads, scheduled by a fixed priority scheduler. A high priority thread and a low priority thread were using the same software data structure (an “information bus”) protected by a mutex. The mutex was actually used by a library that provided high level communication mechanisms among threads, namely the `pipe()` mechanism. At some instant, it happened that the low priority thread was interrupted by a medium priority thread while blocking the high priority thread on the mutex.

At the time of the Mars Pathfinder mission, the problem was already known. The first accounts of the problem and possible solutions date back to early ’70s. However, the problem became widely known in the real-time community since the seminal paper of Sha, Rajkumar and Lehoczky, who proposed the Priority Inheritance Protocol and the Priority Ceiling Protocol to bound the time a real-time task can be blocked on a mutex.
A more complete (but maybe biased) description of the incident can be found here:
http://www.cs.cmu.edu/~rajkumar/mars.html
Dealing with Priority Inversion

- Priority inversion can be reduced...
  - ...But how?
  - By introducing an appropriate resource sharing protocol (concurrency protocol)

- Provides an upper bound for the blocking time
  - Non Preemptive Protocol (NPP) / Highest Locking Priority (HLP)
  - Priority Inheritance Protocol (PI)
  - Priority Ceiling Protocol (PC)
  - Immediate Priority Ceiling Protocol (Part of the OSEK and POSIX standards)

- mutexes (not generic semaphores) must be used
Non Preemptive Protocol (NPP)

- The idea is very simple *inhibit preemption when in a critical section*. How would you implement that?

- Advantages: *simplicity*
- Drawbacks: tasks which are not involved in a critical section suffer blocking
Non Preemptive Protocol (NPP)

- The idea is very simple *inhibit preemption when in a critical section*. How would you implement that?
- Raise the task’s priority to the maximum available priority when entering a critical section

- Advantages: *simplicity*
- Drawbacks: tasks which are not involved in a critical section suffer blocking
Remember the previous example...

Using NPP, we have:
Some Observations

- The blocking (priority inversion) is bounded by the length of the critical section of task \( \tau_3 \).
- Medium priority tasks (\( \tau_2 \)) cannot delay \( \tau_1 \).
- \( \tau_2 \) experiences some blocking, but it does not use any resource.
  
  - *Indirect blocking:* \( \tau_2 \) is in the middle between a higher priority task \( \tau_1 \) and a lower priority task \( \tau_3 \) which use the same resource.
  
  - Must be computed and taken into account in the admission test as any other blocking time.

- What’s the maximum blocking time \( B_i \) for \( \tau_i \)?
A Problem with NPP

- Consider the following example, with $p_0 > p_1 > p_2 > p_3$.

- $\tau_0$ misses its deadline (suffers a blocking time equal to 3) even though it does not use any resource!!

- Solution: raise $\tau_3$ priority to the maximum between tasks accessing the shared resource ($\tau_1$' priority)
So....

This time, everyone is happy

Problem: we must know in advance which task will access the resource
NPP introduces a blocking time on all tasks bounded by the maximum length of a critical section used by lower priority tasks.

How does blocking time affect the response times?

Response Time Computation:

\[ R_i = C_i + B_i + \sum_{j=1}^{i-1} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \]

- \( B_i \) is the blocking time from lower priority tasks
- \( \sum_{h=1}^{i-1} \left\lfloor \frac{R_i}{T_h} \right\rfloor C_h \) is the interference from higher priority tasks
## Response Time Computation - I

<table>
<thead>
<tr>
<th>Task</th>
<th>$C_i$</th>
<th>$T_i$</th>
<th>$\xi_{i,1}$</th>
<th>$D_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td>20</td>
<td>70</td>
<td>0</td>
<td>30</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>20</td>
<td>80</td>
<td>1</td>
<td>45</td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>35</td>
<td>200</td>
<td>2</td>
<td>130</td>
</tr>
</tbody>
</table>
## Response Time Computation - II

<table>
<thead>
<tr>
<th>Task $\tau_i$</th>
<th>$C_i$</th>
<th>$T_i$</th>
<th>$\xi_{i,1}$</th>
<th>$D_i$</th>
<th>$B_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td>20</td>
<td>70</td>
<td>0</td>
<td>30</td>
<td>2</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>20</td>
<td>80</td>
<td>1</td>
<td>45</td>
<td>2</td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>35</td>
<td>200</td>
<td>2</td>
<td>130</td>
<td>0</td>
</tr>
<tr>
<td>Task</td>
<td>$C_i$</td>
<td>$T_i$</td>
<td>$\xi_{i,1}$</td>
<td>$D_i$</td>
<td>$B_i$</td>
</tr>
<tr>
<td>------</td>
<td>-------</td>
<td>-------</td>
<td>-------------</td>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td>$T_1$</td>
<td>20</td>
<td>70</td>
<td>0</td>
<td>30</td>
<td>2</td>
</tr>
<tr>
<td>$T_2$</td>
<td>20</td>
<td>80</td>
<td>1</td>
<td>45</td>
<td>2</td>
</tr>
<tr>
<td>$T_3$</td>
<td>35</td>
<td>200</td>
<td>2</td>
<td>130</td>
<td>0</td>
</tr>
</tbody>
</table>
Another possible solution to the priority inversion:

- A low priority task $\tau_3$ blocking an higher priority task $\tau_1$ inherits its priority
- $\rightarrow$ medium priority tasks cannot preempt $\tau_3$
Another possible solution to the priority inversion:

- a low priority task $\tau_3$ blocking an higher priority task $\tau_1$ *inherits* its priority
- medium priority tasks cannot preempt $\tau_3$
Another possible solution to the priority inversion:
- a low priority task $\tau_3$ blocking an higher priority task $\tau_1$ inherits its priority
- medium priority tasks cannot preempt $\tau_3$
The Priority Inheritance protocol

- Another possible solution to the priority inversion:
  - a low priority task $\tau_3$ blocking an higher priority task $\tau_1$ inherits its priority
  - medium priority tasks cannot preempt $\tau_3$

- Task $\tau_3$ inherits the priority of $\tau_1$
The Priority Inheritance protocol

- Another possible solution to the priority inversion:
  - a low priority task $\tau_3$ blocking an higher priority task $\tau_1$ *inherits* its priority
  - medium priority tasks cannot preempt $\tau_3$

- Task $\tau_3$ inherits the priority of $\tau_1$
- Task $\tau_2$ cannot preempt $\tau_3$ ($p_2 < p_1$)
The Priority Inheritance protocol

- Another possible solution to the priority inversion:
  - a low priority task \( \tau_3 \) blocking an higher priority task \( \tau_1 \) *inherits* its priority
  - medium priority tasks cannot preempt \( \tau_3 \)

- Task \( \tau_3 \) inherits the priority of \( \tau_1 \)
- Task \( \tau_2 \) cannot preempt \( \tau_3 \) \((p_2 < p_1)\)
Another possible solution to the priority inversion:

- a low priority task $\tau_3$ blocking an higher priority task $\tau_1$ *inherits* its priority
- medium priority tasks cannot preempt $\tau_3$

Task $\tau_3$ inherits the priority of $\tau_1$
Task $\tau_2$ cannot preempt $\tau_3$ ($p_2 < p_1$)
The Priority Inheritance protocol

- Another possible solution to the priority inversion:
  - a low priority task $\tau_3$ blocking an higher priority task $\tau_1$ inherits its priority
  - medium priority tasks cannot preempt $\tau_3$

- Task $\tau_3$ inherits the priority of $\tau_1$
- Task $\tau_2$ cannot preempt $\tau_3$ ($p_2 < p_1$)
The Priority Inheritance protocol

- Another possible solution to the priority inversion:
  - a low priority task \( \tau_3 \) blocking an higher priority task \( \tau_1 \) *inherits* its priority
  - medium priority tasks cannot preempt \( \tau_3 \)

\[
\begin{align*}
\tau_1 & \quad \text{L(S)} & \quad \text{U(S)} & \quad \text{L(S)} & \quad \text{U(S)} \\
\tau_2 & \quad \text{S} & \quad \text{S} & \quad \text{S} & \quad \text{S} \\
\tau_3 & \quad \text{L(S)} & \quad \text{U(S)} & \quad \text{L(S)} & \quad \text{U(S)}
\end{align*}
\]

- Task \( \tau_3 \) inherits the priority of \( \tau_1 \)
- Task \( \tau_2 \) cannot preempt \( \tau_3 \) \((p_2 < p_1)\)
The Priority Inheritance protocol

- Another possible solution to the priority inversion:
  - a low priority task $\tau_3$ blocking an higher priority task $\tau_1$ inherits its priority
  - medium priority tasks cannot preempt $\tau_3$

- Task $\tau_3$ inherits the priority of $\tau_1$
- Task $\tau_2$ cannot preempt $\tau_3$ ($p_2 < p_1$)
Some PI Properties

- Summarising, the main rules are the following:
  - If a task $\tau_i$ blocks on a resource protected by a mutex $S$, and the resource is locked by task $\tau_j$, then $\tau_j$ inherits the priority of $\tau_i$.
  - If $\tau_j$ itself blocks on another mutex by a task $\tau_k$, then $\tau_k$ inherits the priority of $\tau_i$ (multiple inheritance).
  - If $\tau_k$ is blocked, the chain of blocked tasks is followed until a non-blocked task is found that inherits the priority of $\tau_i$.
  - When a task unlocks a mutex, it returns to the priority it had when locking it.
We only consider *non nested* critical sections...

- In presence of multiple inheritance, the computation of the blocking time becomes very complex
- Non nested critical sections → multiple inheritance cannot happen, and the computation of the blocking time becomes simpler

The maximum blocking time can be computed based on two important properties

- They provide an upper bound on the number of times a task can block
Two important theorems:

- **Theorem 1** if PI is used, a task block only once on each different critical section
- **Theorem 2** if PI is used, a task can be blocked by another lower priority task for at most the duration of one critical section

⇒ a task can be blocked more than once, but only once per each resource and once by each task
We must build a *resource usage table*

- A task per row, in decreasing order of priority
- A resource per column
- Cell \((i, j)\) contains \(\xi_{i,j}\)
  - i.e. the length of the longest critical section of task \(\tau_i\) on resource \(S_j\), or 0 if the task does not use the resource

Blocking times can be computed based on this table
How to use the resource usage table?

Let’s recall the 2 PI properties...

A task can be blocked only by lower priority tasks:

Then, for each task (row), we must consider only the rows below (tasks with lower priority)

A task block only on resources directly used, or used by higher priority tasks (indirect blocking):

For each task, only consider columns on which it can be blocked (used by itself or by higher priority tasks)
Example - 1

<table>
<thead>
<tr>
<th></th>
<th>$S_1$</th>
<th>$S_2$</th>
<th>$S_3$</th>
<th>$B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>?</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>?</td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>?</td>
</tr>
<tr>
<td>$\tau_4$</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>?</td>
</tr>
<tr>
<td>$\tau_5$</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>?</td>
</tr>
</tbody>
</table>

- Let’s start from $B_1$
- $\tau_1$ can be blocked only on $S_1$. Therefore, we must consider only the first column, and take the maximum, which is 3. Therefore, $B_1 = 3$. 
Example - 2

<table>
<thead>
<tr>
<th></th>
<th>$S_1$</th>
<th>$S_2$</th>
<th>$S_3$</th>
<th>$B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>?</td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>?</td>
</tr>
<tr>
<td>$\tau_4$</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>?</td>
</tr>
<tr>
<td>$\tau_5$</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>?</td>
</tr>
</tbody>
</table>

- $\tau_2$ can be blocked on $S_1$ (*indirect blocking*) and on $S_2$
- Consider all cases where two distinct lower priority tasks in \{$\tau_3, \tau_4, \tau_5$\} access $S_1$ and $S_2$, sum the two contributions, and take the maximum;
  - $\tau_4$ on $S_1$ and $\tau_5$ on $S_2$: $\rightarrow 5$
  - $\tau_4$ on $S_2$ and $\tau_5$ on $S_1$: $\rightarrow 4$
Example - 3

<table>
<thead>
<tr>
<th></th>
<th>$S_1$</th>
<th>$S_2$</th>
<th>$S_3$</th>
<th>$B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>?</td>
</tr>
<tr>
<td>$\tau_4$</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>?</td>
</tr>
<tr>
<td>$\tau_5$</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>?</td>
</tr>
</tbody>
</table>

- $\tau_3$ can be blocked on all 3 resources
  - $\tau_4$ on $S_1$ and $\tau_5$ on $S_2$: $\rightarrow 5$;
  - $\tau_4$ on $S_2$ and $\tau_5$ on $S_1$ or $S_3$: $\rightarrow 4$;
  - $\tau_4$ on $S_3$ and $\tau_5$ on $S_1$: $\rightarrow 2$;
  - $\tau_4$ on $S_3$ and $\tau_5$ on $S_2$ or $S_3$: $\rightarrow 3$;
### Example - 4

<table>
<thead>
<tr>
<th></th>
<th>$S_1$</th>
<th>$S_2$</th>
<th>$S_3$</th>
<th>$B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>$\tau_4$</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>?</td>
</tr>
<tr>
<td>$\tau_5$</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>?</td>
</tr>
</tbody>
</table>

- $\tau_4$ can be blocked on all 3 resources. We must consider all columns; however, it can be blocked only by $\tau_5$.
- The maximum is $B_4 = 2$.
- $\tau_5$ cannot be blocked by any other task (because it is the lower priority task!); $B_5 = 0$.
### Example: Final result

<table>
<thead>
<tr>
<th>( \tau_i )</th>
<th>( S_1 )</th>
<th>( S_2 )</th>
<th>( S_3 )</th>
<th>( B )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \tau_1 )</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>( \tau_2 )</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>( \tau_3 )</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>( \tau_4 )</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>( \tau_5 )</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
An example

- Task
  - 5 Tasks
- Shared resources
  - Results buffer
    - Used by R1 and R2
    - R1 (2 ms) R2 (20 ms)
  - Communication buffer
    - Used by C1 and C3
    - C1 (10 ms) C3 (10 ms)
Example of blocking time computation

<table>
<thead>
<tr>
<th></th>
<th>C</th>
<th>T</th>
<th>D</th>
<th>$\xi_{1,i}$</th>
<th>$\xi_{2,i}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>ES</td>
<td>5</td>
<td>50</td>
<td>6</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>IS</td>
<td>10</td>
<td>100</td>
<td>100</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$\tau_1$</td>
<td>20</td>
<td>100</td>
<td>100</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>40</td>
<td>150</td>
<td>130</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>100</td>
<td>350</td>
<td>350</td>
<td>0</td>
<td>10</td>
</tr>
</tbody>
</table>
## Table of resource usage

<table>
<thead>
<tr>
<th></th>
<th>$\xi_{1,i}$</th>
<th>$\xi_{2,i}$</th>
<th>$B_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>ES</td>
<td>0</td>
<td>0</td>
<td>?</td>
</tr>
<tr>
<td>IS</td>
<td>0</td>
<td>0</td>
<td>?</td>
</tr>
<tr>
<td>$\tau_1$</td>
<td>2</td>
<td>10</td>
<td>?</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>20</td>
<td>0</td>
<td>?</td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>0</td>
<td>10</td>
<td>?</td>
</tr>
</tbody>
</table>
Computation of the blocking time

<table>
<thead>
<tr>
<th></th>
<th>$\xi_{1,i}$</th>
<th>$\xi_{2,i}$</th>
<th>$B_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>ES</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>IS</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$\tau_1$</td>
<td>2</td>
<td>10</td>
<td>?</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>20</td>
<td>0</td>
<td>?</td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>0</td>
<td>10</td>
<td>0</td>
</tr>
</tbody>
</table>

- Task $ES$ and $IS$ do not experience any blocking since neither do they use shared resource (direct blocking) nor are there tasks having higher priority that do so (indirect blocking).
- Task $\tau_3$ does not experience any blocking time either (since it is the one having the lowest priority).
Computation of the blocking time

<table>
<thead>
<tr>
<th></th>
<th>$\xi_{1,i}$</th>
<th>$\xi_{2,i}$</th>
<th>$B_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>ES</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>IS</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$\tau_1$</td>
<td>2</td>
<td>10</td>
<td>30</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>20</td>
<td>0</td>
<td>?</td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>0</td>
<td>10</td>
<td>0</td>
</tr>
</tbody>
</table>

- For task $\tau_1$ we have to consider both columns 1 and 2 since it uses both resources.
- The possibilities are:
  - $\tau_2$ on $S_1$ and $\tau_3$ on $S_2$: $\rightarrow 30$;
Computation of the blocking time

<table>
<thead>
<tr>
<th></th>
<th>$\xi_{1,i}$</th>
<th>$\xi_{2,i}$</th>
<th>$B_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>ES</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>IS</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$\tau_1$</td>
<td>2</td>
<td>10</td>
<td>30</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>20</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>0</td>
<td>10</td>
<td>0</td>
</tr>
</tbody>
</table>

- For task $\tau_2$, consider column 2: it represents the only resource used by tasks having both higher and lower priority than $\tau_2$ ($\tau_2$ itself uses resource 1 which is not used by other tasks with lower priority)
- The possibilities are:
  - $\tau_3$ on $S_2$: $\rightarrow 10$;
### The response times

<table>
<thead>
<tr>
<th></th>
<th>C</th>
<th>T</th>
<th>D</th>
<th>$\xi_{1,i}$</th>
<th>$\xi_{2,i}$</th>
<th>$B_i$</th>
<th>$R_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>ES</td>
<td>5</td>
<td>50</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5+0+0=5</td>
</tr>
<tr>
<td>IS</td>
<td>10</td>
<td>100</td>
<td>100</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>10+0+5=15</td>
</tr>
<tr>
<td>$\tau_1$</td>
<td>20</td>
<td>100</td>
<td>100</td>
<td>2</td>
<td>10</td>
<td>30</td>
<td>20+30+20=70</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>40</td>
<td>150</td>
<td>130</td>
<td>20</td>
<td>0</td>
<td>10</td>
<td>40+10+40=90</td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>100</td>
<td>350</td>
<td>350</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>100+0+200=300</td>
</tr>
</tbody>
</table>
Response Time Analysis

- We have seen the schedulability test based on response time analysis

\[ R_i = C_i + B_i + \sum_{h=1}^{i-1} \left\lceil \frac{R_i}{T_h} \right\rceil C_h \]

- There are also other options
- For instance the following sufficient test: The system is schedulable if

\[ \forall i, 1 \leq i \leq n, \sum_{k=1}^{i-1} \frac{C_k}{T_k} + \frac{C_i + B_i}{T_i} \leq i(2^{1/i} - 1) \]
Time Demand Approach

- In a task set $\mathcal{T}$ composed of independent and periodic tasks, $\tau_i$ is schedulable (for all possible phasings) iff

$$\exists 0 \leq t \leq D_i : W_i(0, t) = C_i + \sum_{h=1}^{i-1} \left\lceil \frac{t}{T_h} \right\rceil C_h \leq t$$
Time Demand Approach

- In a task set $\mathcal{T}$ composed of independent and periodic tasks, $\tau_i$ is schedulable (for all possible phasings) **iff**

$$\exists 0 \leq t \leq D_i : W_i(0, t) = C_i + \sum_{h=1}^{i-1} \left\lceil \frac{t}{T_h} \right\rceil C_h \leq t$$

- Introducing blocking times $B_i$, $\tau_i \in \mathcal{T}$ is schedulable **if** exists $0 \leq t \leq D_i$ such that

$$W_i(0, t) = C_i + \sum_{h=1}^{i-1} \left\lceil \frac{t}{T_h} \right\rceil C_h \leq t - B_i$$
As usual, we can define

\[ W_i(t) = C_i + \sum_{h=1}^{i-1} \left\lceil \frac{t}{T_h} \right\rceil C_h \]

\[ L_i(t) = \frac{W_i(t)}{t} \]

\[ L_i = \min_{0 \leq t \leq D_i} L_i(t) + \frac{B_i}{t} \]
As usual, we can define

\[ W_i(t) = C_i + \sum_{h=1}^{i-1} \left\lfloor \frac{t}{T_h} \right\rfloor C_h \]

\[ L_i(t) = \frac{W_i(t)}{t} \]

\[ L_i = \min_{0 \leq t \leq D_i} L_i(t) + \frac{B_i}{t} \]

The task set is schedulable if \( \forall i, L_i \leq 1 \)
As usual, we can define

\[ W_i(t) = C_i + \sum_{h=1}^{i-1} \left\lfloor \frac{t}{T_h} \right\rfloor C_h \]

\[ L_i(t) = \frac{W_i(t)}{t} \]

\[ L_i = \min_{0 \leq t \leq D_i} L_i(t) + \frac{B_i}{t} \]

The task set is schedulable if \( \forall i, L_i \leq 1 \)

Again, we can compute \( L_i \) by only considering the scheduling points