Exercise 4.1: Scheduling

The following process set with safety-critical hard deadlines is given:

<table>
<thead>
<tr>
<th>Process</th>
<th>T (ms)</th>
<th>C (ms)</th>
<th>Degree of Importance</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>100</td>
<td>30</td>
<td>crucial</td>
</tr>
<tr>
<td>Q</td>
<td>6</td>
<td>1</td>
<td>substantial</td>
</tr>
<tr>
<td>S</td>
<td>25</td>
<td>5</td>
<td>least important</td>
</tr>
</tbody>
</table>

a. Draw the timeline to determine whether the process set is schedulable or not assuming that the priority is based on degree of importance and preemption is not allowed.

b. What is the processor utilisation by these processes? Is there a scheduling strategy that allows these processes to meet their deadlines? If so, name the strategy and draw the timeline.

c. A fourth process R is added to the above process set. Failure of this process to meet its (soft) deadline will not lead to safety being undermined. R has a period of 50, but has a processing requirement that is data dependent and varies from 5 to 17 milliseconds. Discuss how process R can be integrated with the other processes.

Exercise 4.2: Schedulability Analysis

The following process set is given:

<table>
<thead>
<tr>
<th>Process</th>
<th>T (ms)</th>
<th>C (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>40</td>
<td>10</td>
</tr>
<tr>
<td>Q</td>
<td>35</td>
<td>10</td>
</tr>
<tr>
<td>S</td>
<td>30</td>
<td>9</td>
</tr>
</tbody>
</table>

a. Using the utilisation-based schedulability test, evaluate the possibility of scheduling the process set.
b. Using response time analysis for rate-monotonic priority order (RMPO), determine if it is possible to schedule the process set.

c. Assume that the following deadlines are now assigned to the processes: \( D_P = 14 \), \( D_Q = 21 \), and \( D_S = 29 \). Using response time analysis for deadline monotonic priority order (DMPO), determine if it is possible to schedule the process set. Does the answer remain the same if \( D_S = 28 \)?

**Exercise 4.3: Concurrent Execution and Drift**

To achieve periodic execution but eliminate the need for busy-waits, most programming languages provide some type of `delay` primitive. Consider the following two Ada program fragments:

```
Start := Ada.Real_Time.Clock;
First_Action;
delay until Start + 10.0;
Second_Action;
```

```
Start := Ada.Real_Time.Clock;
First_Action;
delay (Start + 10.0) - Ada.Real_Time.Clock;
Second_Action;
```

`First_Action` and `Second_Action` are calls to procedures that execute sequentially. Both program fragments are executed in a concurrent environment.

a. Do both program fragments have the same behaviour? If not, use an example to explain how their behaviour differs.

b. What are the time overruns associated with the delays called, and what are their causes? Do they occur for relative or absolute delays, or both? Can they be eliminated? Justify your answers.

c. For the following three Ada program fragments, state which type of drift applies, assuming concurrent execution:

**Fragment one:**
```
Start := Ada.Real_Time.Clock;
loop
    delay 10.0 - (Ada.Real_Time.Clock - Start);
    Start := Ada.Real_Time.Clock;
    First_Action;
end loop;
```

**Fragment two:**
```
Next_Time := Ada.Real_Time.Clock;
loop
    delay until Next_Time;
    Next_Time := Next_Time + 10.0;
    First_Action;
end loop;
```

**Fragment three:**
```
Next_Time := Ada.Real_Time.Clock;
loop
    delay until Next_Time;
    Next_Time := Start + 10.0;
    First_Action;
end loop;
```

2