*E x* *p* *e* *r* *i* *m* *e* *n* *t* *2*

Implementing user defined scheduling algorithms

Objectives:

* Creating a user-defined scheduler
* User Defined EDF with absolute deadline as dynamic priority
* User defined protocol with relative deadline as the dynamic priority

**TABLE OF CONTENTS**

* 1. Objective
  2. Creating user defined Scheduler
     1. User defined EDF with absolute deadline as dynamic priority
     2. User defined EDF with relative deadline as dynamic priority
  3. Theoretical Backgroun
  4. Pre lab Questions
  5. In-Lab Experimental Procedure
  6. Outcomes

Appendix A : In-Lab Report

* 1. Objective

This experiment aims to creating a user defined scheduler for EDF with absolute and relative deadline as dynamic priority

* 1. References

1. Liu, Jane W.S., Real Time Systems, Pearson Education, 2000
2. Cheddae Release 3.x user’s guide
   1. Theoretical Background

Earliest Deadline First (EDF) is an optimal dynamic priority scheduling algorithm used

in real-time systems. It can be used for both static and dynamic real-time scheduling.EDF uses priorities to the jobs for scheduling.

It assigns priorities to the task according to the absolute deadline. The task whose deadline is closest gets the highest priority. The priorities are assigned and changed in a dynamic fashion. EDF is very efficient as compared to other scheduling algorithms in real-time systems. It can make the CPU utilization to about 100% while still guaranteeing the deadlines of all the tasks.

EDF scheduling algorithm does not need the tasks or processes to be periodic and also the tasks or processes require a fixed CPU burst time. In EDF, any executing task can be preempted if any other periodic instance with an earlier deadline is ready for execution and becomes active. Preemption is allowed in the Earliest Deadline First scheduling algorithm.

* 1. .Pre Lab Assignment

Q1: What are the various Priority driven algorithms

Q2. What is relative deadline and absolute deadline

Q3: What do you understand by the term phase of the task. Explain

Q4: What is effective release time and effective deadline. Explain with example

* 1. In-Lab Experimental Procedure

## Introduction

The code for the scheduling algorithm is written in ADA language. Any user defined scheduling algorithm will have three sections as shown if the fig:

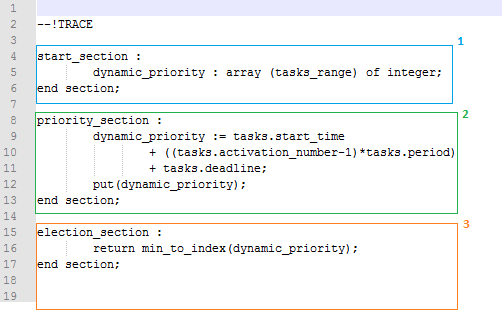


Fig 1: 3 sections of a scheduling algorithm

**1Start Section:** Variables required to schedule the task are declared in this section with their type (excluding the predefined variables). Two types are possible: scalar and arrays. Scalar includes integer, Boolean, double, string and an array is a type which stores one scalar data per task, message, buffer or shared resource. This section is invoked only once, at the start of execution.

**2Priority Section:** This section is called every time a scheduling decision is to be made and this section consists of the code to compute task priorities.

**3Election Section:** This section contains of a return statement which tells which task should be executed for the next unit of time.

This mimics the job done by an on-line scheduler – when a task is generated (this is an event) the scheduler calculates the priority of the task, and then selects which task is to be scheduled. This done every time task is generated and it is completed – the same is in case of this simulated dispatcher.

From Figure 1, the start section consists of a user defined variable dynamic\_priority which is of type integer array.

In the priority section, the dynamic\_priority (absolute deadline) for each task is calculated. The absolute deadline of the task is calculated depending on the values of tasks.ready , tasks.rest\_of\_capacity and using the values of tasks.start\_time, tasks.activation\_number, tasks.period and tasks.deadline.

**tasks.ready** gives the state of the task whether it is ready to run i.e. it is not waiting for any resource or precedency constraint.

**tasks.rest\_of\_capacity** gives the remaining no of units of execution.

**tasks.start\_time, tasks.period, tasks.deadline** are the start time, period and relative deadline specified by the user at the time of task definition.

**task.activation\_number** stores the instance of a task i.e. every time a periodic task is activated, this variable is incremented.

These are the variables that are required for computing priority – these essentially will be input to the dispatcher(scheduler)

A list of all available predefined variables can be found in

<http://beru.univ-brest.fr/~singhoff/cheddar/ug/ug_v3/cheddar-r3.html#Ref6.5>

Creating a user-defined scheduler

### User Defined EDF with absolute deadline as dynamic priority

* Write your scheduler code in any editor (preferably Notepad++) and save the file with the extension of .sc
* The .sc file should be in the same folder as that of the cheddar application.
* Launch the application by clicking on cheddar.exe from Cheddar-3.0-win32-bin folder.

Refer to the experiment 1 to create the project.

Following changes must be made to invoke a user defined scheduler:

* While adding the core, define the ***Scheduler type***as Pipeline User Defined Protocol and under ***User defined scheduler file name*** specify the name of the .sc file as shown in the figure.

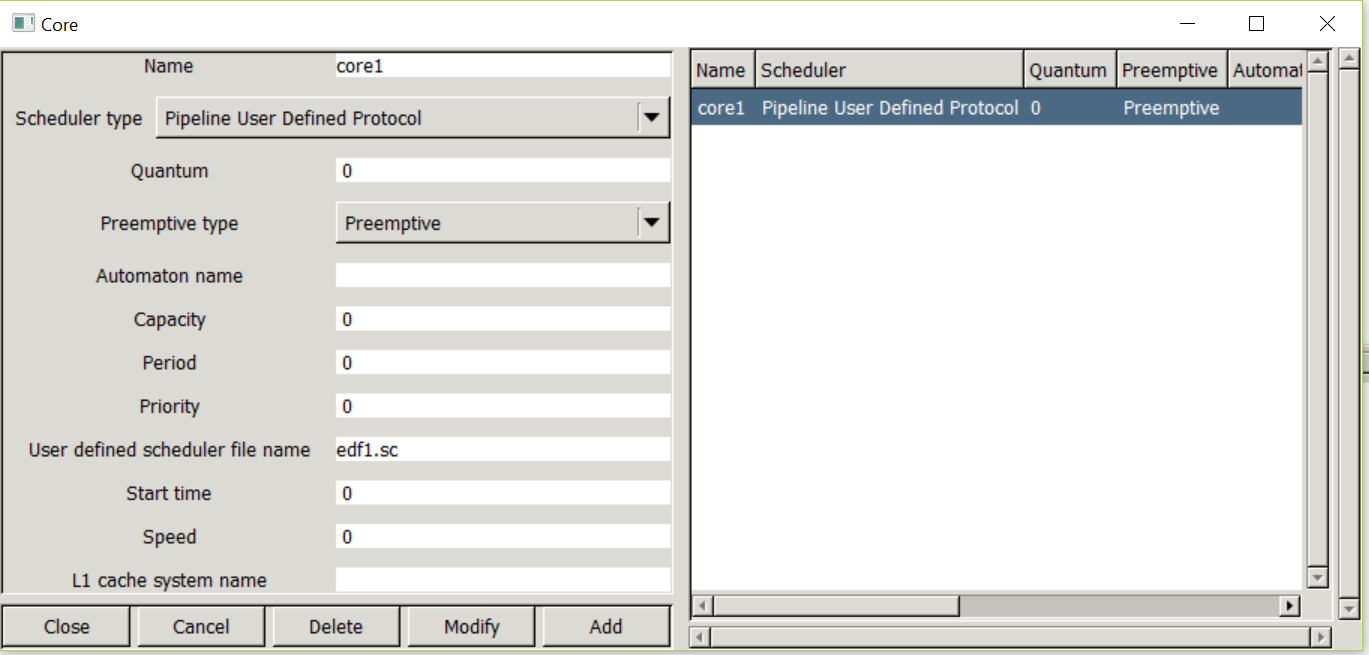


Fig 2: Adding a user-defined core

The .sc file of the scheduler will be as follows:

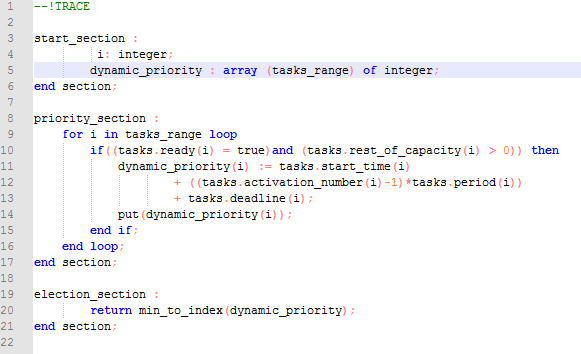


Fig 3: User-defined EDF algorithm

From the above figure, lines 9 -13 are added to compute the dynamic deadline for all the tasks in the given task set.

As cheddar considers relative deadline, the absolute deadline is computed by adding the start time (of each instance – that is why start time is added with instance release time [given as instance no. \* period] )to the relative deadlines which will be the dynamic\_priority in case of EDF.

The following task set has been considered as an example:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Tasks | R | e | d | p |
| T1 | 2 | 3 | 8 | 8 |
| T2 | 0 | 2 | 4 | 4 |
| T3 | 4 | 2 | 14 | 16 |

***Please note that the deadline(d) is absolute.***

* Adding the task set in cheddar

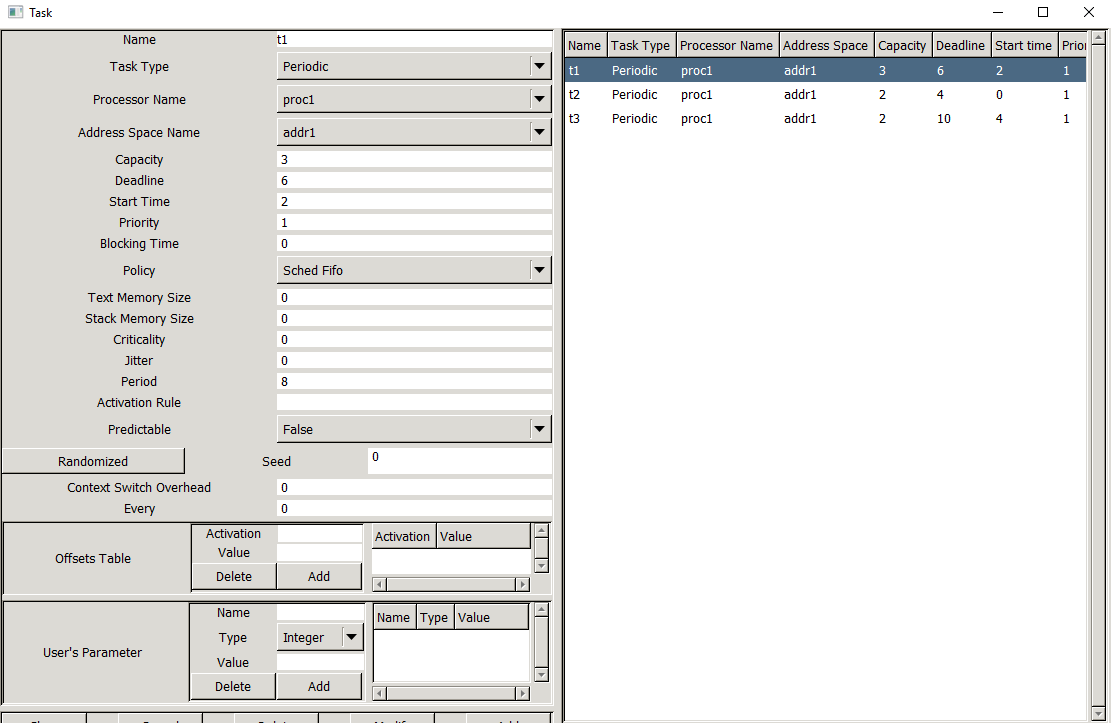


Fig 4: Task set

User defined protocol with relative deadline as the dynamic priority

In the previous case, dynamic priority was decided based on the absolute deadline whereas here, it’s decided based on the relative deadline(tasks.deadline).

The .sc file for the same is as shown in the figure:

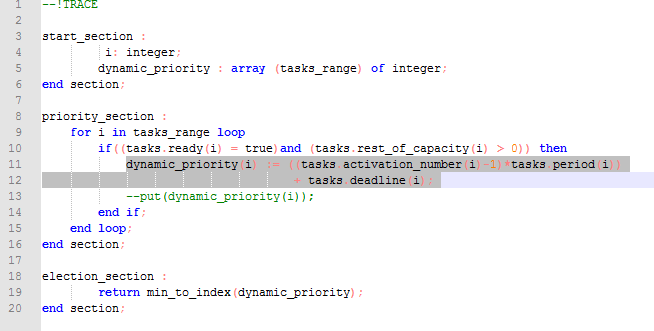


Fig 6: .sc of user defined algorithm with relative deadline as the dynamic priority

Follow the same procedure to add the core, processor, address space and tasks(same task set is considered).

The simulation results for the same are as follows:

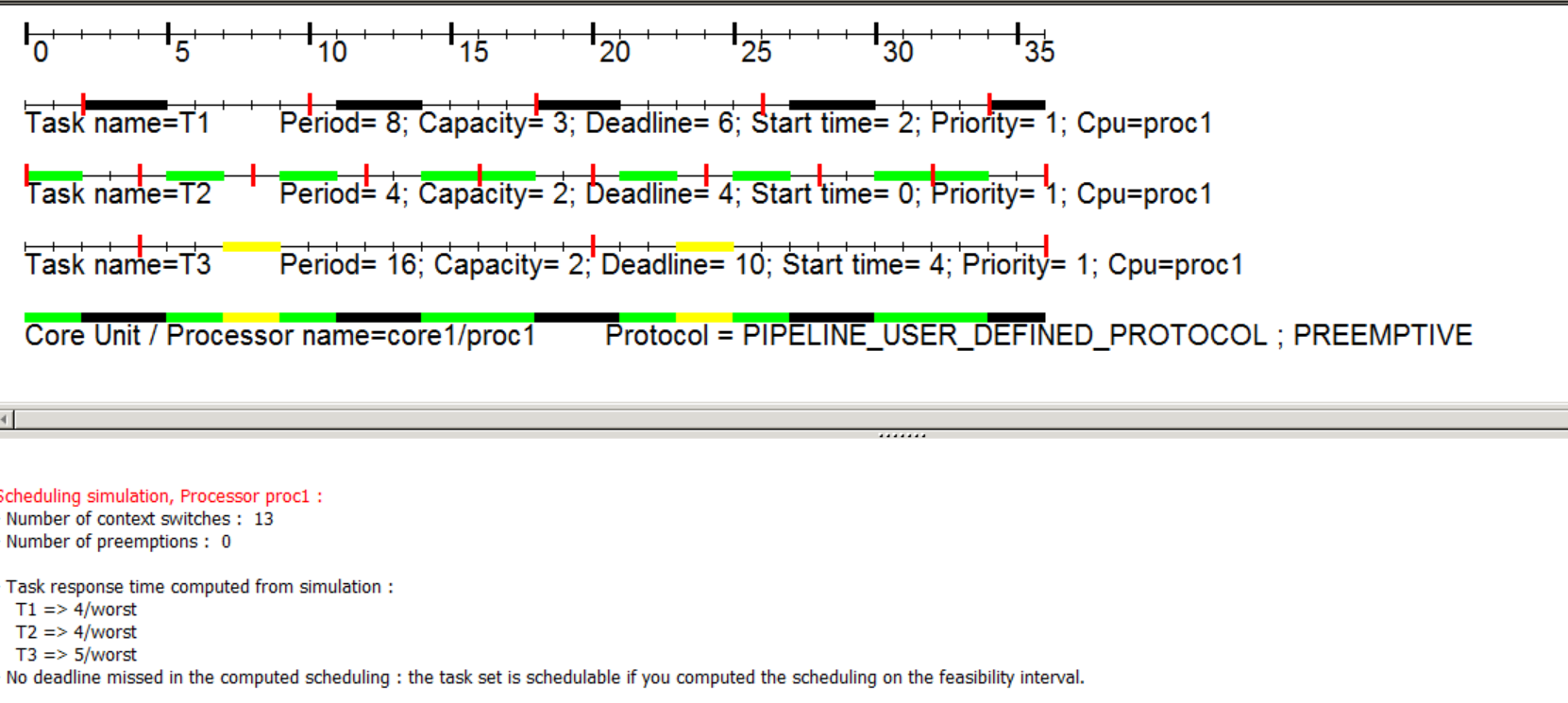


Fig 7: Simulation results

At time unit 7, T3 starts executing and has a relative deadline of 12. At time unit 8, third instance of T2 will arrive and has a deadline of 12 as well. Hence T3 will continue its execution and will not be pre-empted. The no. of context switches and pre-emptions have been reduced in comparison to the previous case.

* 1. Outcomes

The simulation results for the user defined EDF with Absolute deadline as dynamic priority are as follows:

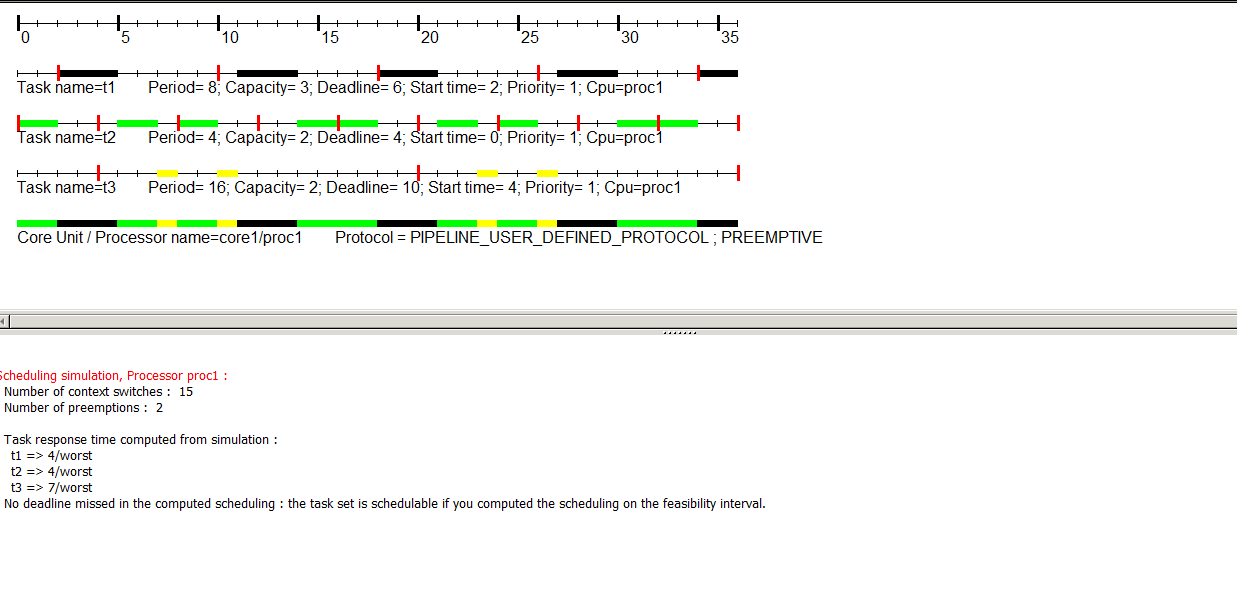


Fig 5: Simulation results for user-defined EDF with absolute deadline dynamic priority

At time unit 7, T3 will start execution and has a deadline of 16. At 8, the next instance of T2 will start and has a deadline of 12. As the absolute deadline of T2 is lesser than that of T3,T3 will be pre-empted by T2. This will repeat at 16 and 24.

The simulation results for the user defined EDF with relative deadline dynamic priority are as follows:

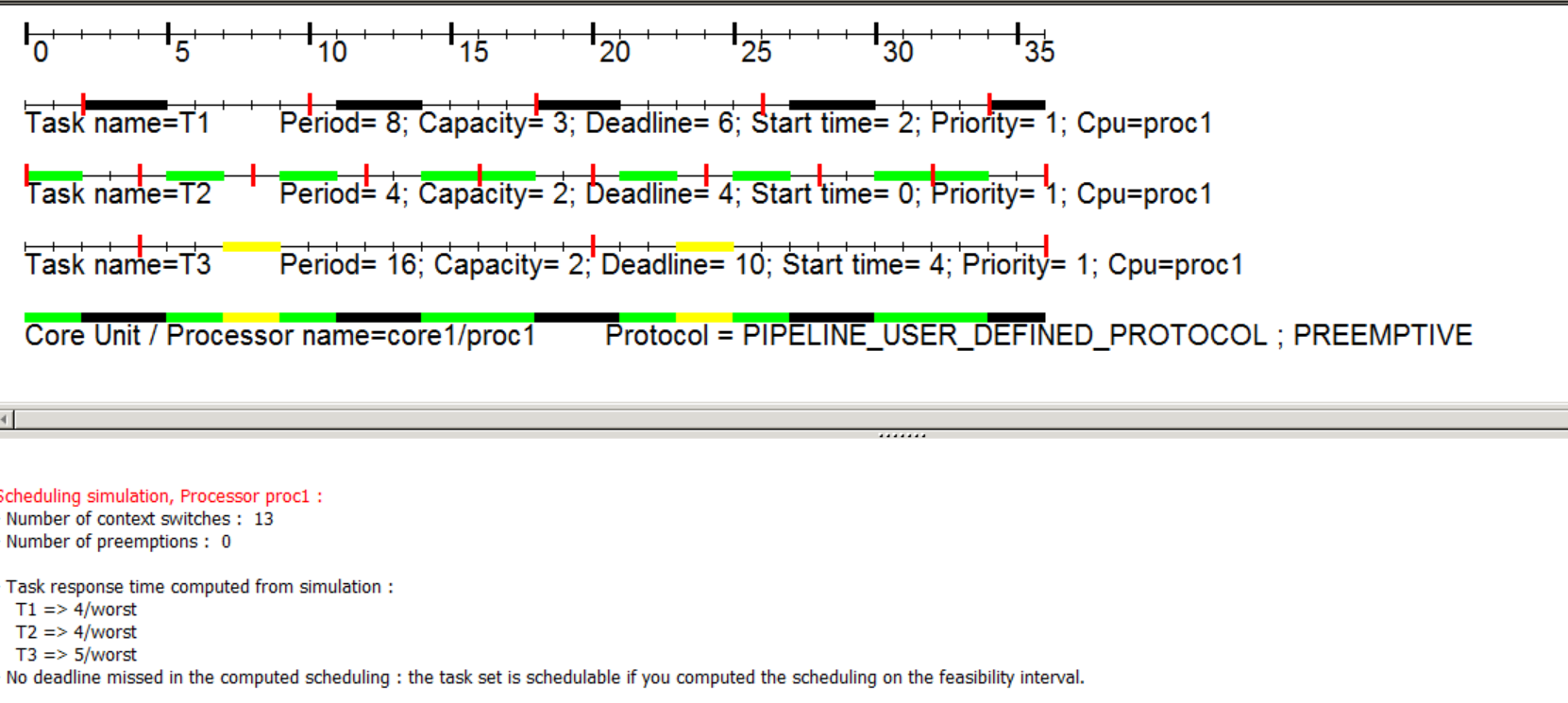


Fig 7: Simulation results

At time unit 7, T3 starts executing and has a relative deadline of 12. At time unit 8, third instance of T2 will arrive and has a deadline of 12 as well. Hence T3 will continue its execution and will not be pre-empted. The no. of context switches and pre-emptions have been reduced in comparison to the previous case.

Appendix A

***Experiment 2***

Implementing user defined scheduling algorithms

**In -Lab Report**

## Exercise 1

Create a scheduler uf.sc that will assign highest static priority to task with the least CPU utilization and compare it against the behaviour of a rate monotonic scheduler for any task set example from class or from your textbook.

## Exercise 2

Create a scheduler critical.sc that will schedule a task that is critical immediately, else it uses RMS and compare it against the behavior of a rate monotonic scheduler for the following task set. If there are multiple critical tasks – then RMS is applied within them. Under what circumstances will this scheduler work better than RMS – show such a task set example.