

ECEN 5623
REAL-TIME EMBEDDED
SYSTEMS

EXERCISE-2
SERVICE SCHEDULING
FEASIBILITY

SUBMITTED BY:
MOUNIKA REDDY EDULA
SOWMYA
RAMAKRISHNAN

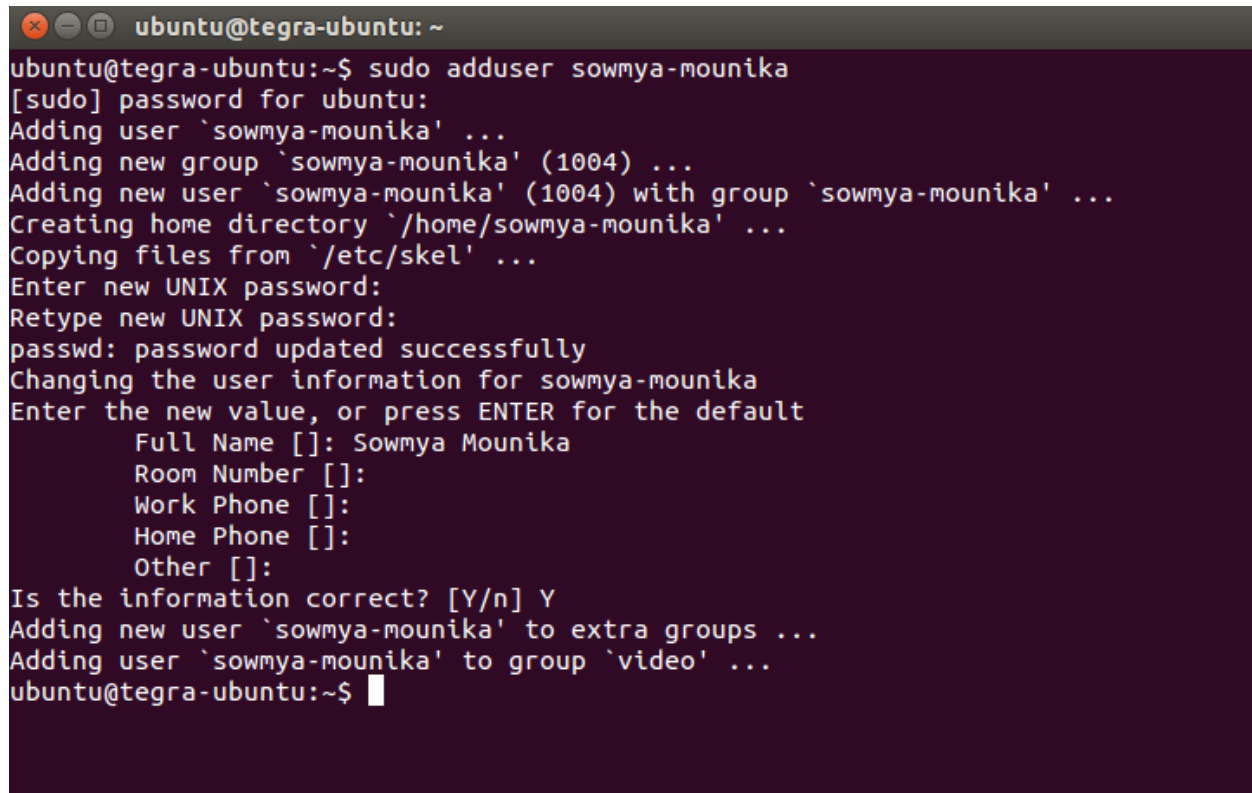
1.

Make yourself an account on your Dev Kit. To do this, use the reset button if the system is locked, use your password to login, and then use “sudo adduser”, enter a password, and enter user information as you see fit. Add your new user account as a “sudoer” using “visudo” right below root with the same privileges (if you need help with “vi”, here’s a [quick reference](#) or [reference card](#)– use arrows to position cursor, below root hit Esc, “i” for insert, type username and privileges as above, and when done, Esc, “:”, “wq”). The old [unix vi editor](#) was one of the first full-screen visual editors – it still has the advantage of being found on virtually any Unix system in existence, but is otherwise cryptic – along with [Emacs](#) it is still widely used in IT, by developers and systems engineers, so it’s good to know the basics. If you really don’t like vi or Emacs, your next best bet is “nano” for Unix systems. Do a quick “sudo whoami” to demonstrate success. Logout of Linux and test your login, then logout. Use Alt+Print-Screen to capture your desktop and save as proof you set up your account. Note that you can always get a terminal with Ctrl+Alt+t key combination. If you don’t like the desktop, you can try “GNOME Flashback” and please play around with customizing your account as you wish.

Answer:

An account “sowmya-mounika” was created on the Jetson TK1 Dev Kit.

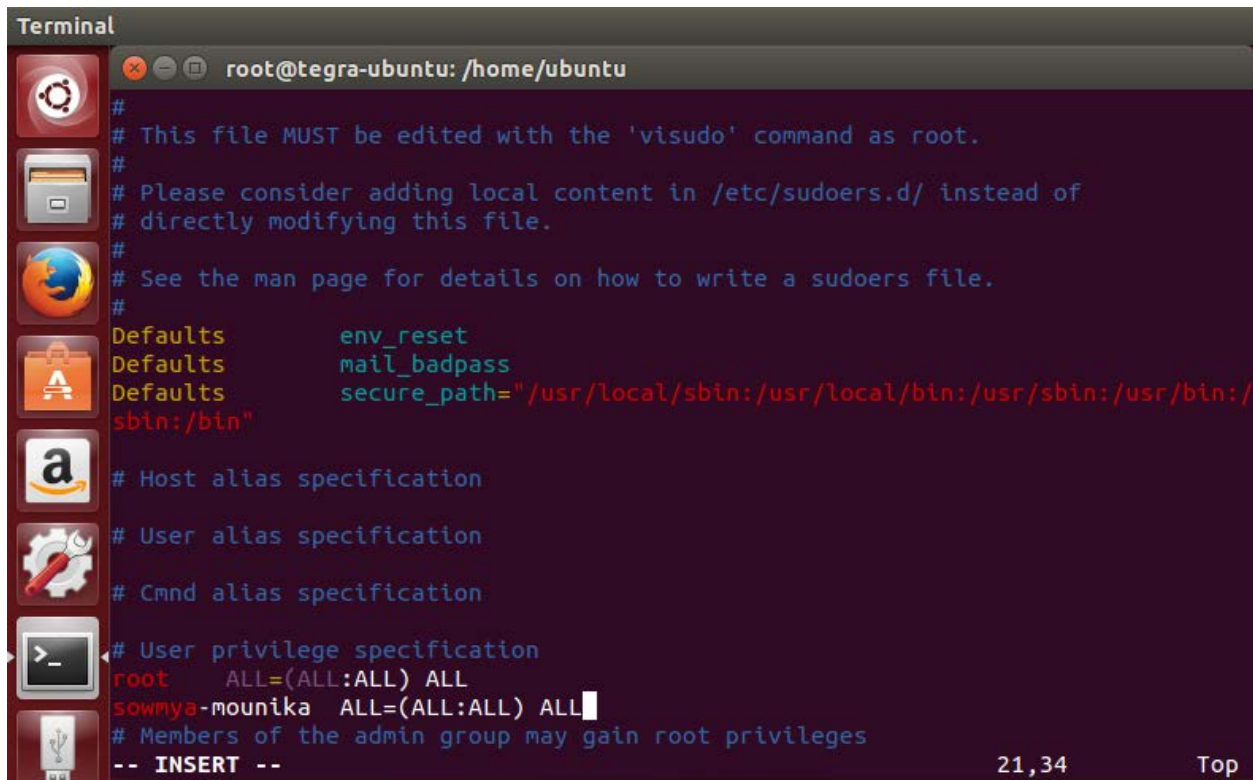
The command “sudo adduser sowmya-mounika” was used to create the account. A new UNIX password was entered., as were the account details [Name: Sowmya Mounika].



```
ubuntu@tegra-ubuntu: ~  
ubuntu@tegra-ubuntu:~$ sudo adduser sowmya-mounika  
[sudo] password for ubuntu:  
Adding user `sowmya-mounika' ...  
Adding new group `sowmya-mounika' (1004) ...  
Adding new user `sowmya-mounika' (1004) with group `sowmya-mounika' ...  
Creating home directory `/home/sowmya-mounika' ...  
Copying files from `/etc/skel' ...  
Enter new UNIX password:  
Retype new UNIX password:  
passwd: password updated successfully  
Changing the user information for sowmya-mounika  
Enter the new value, or press ENTER for the default  
    Full Name []: Sowmya Mounika  
    Room Number []:  
    Work Phone []:  
    Home Phone []:  
    Other []:  
Is the information correct? [Y/n] Y  
Adding new user `sowmya-mounika' to extra groups ...  
Adding user `sowmya-mounika' to group `video' ...  
ubuntu@tegra-ubuntu:~$
```

The account was added as a “sudoer” by editing the “visudo” file.

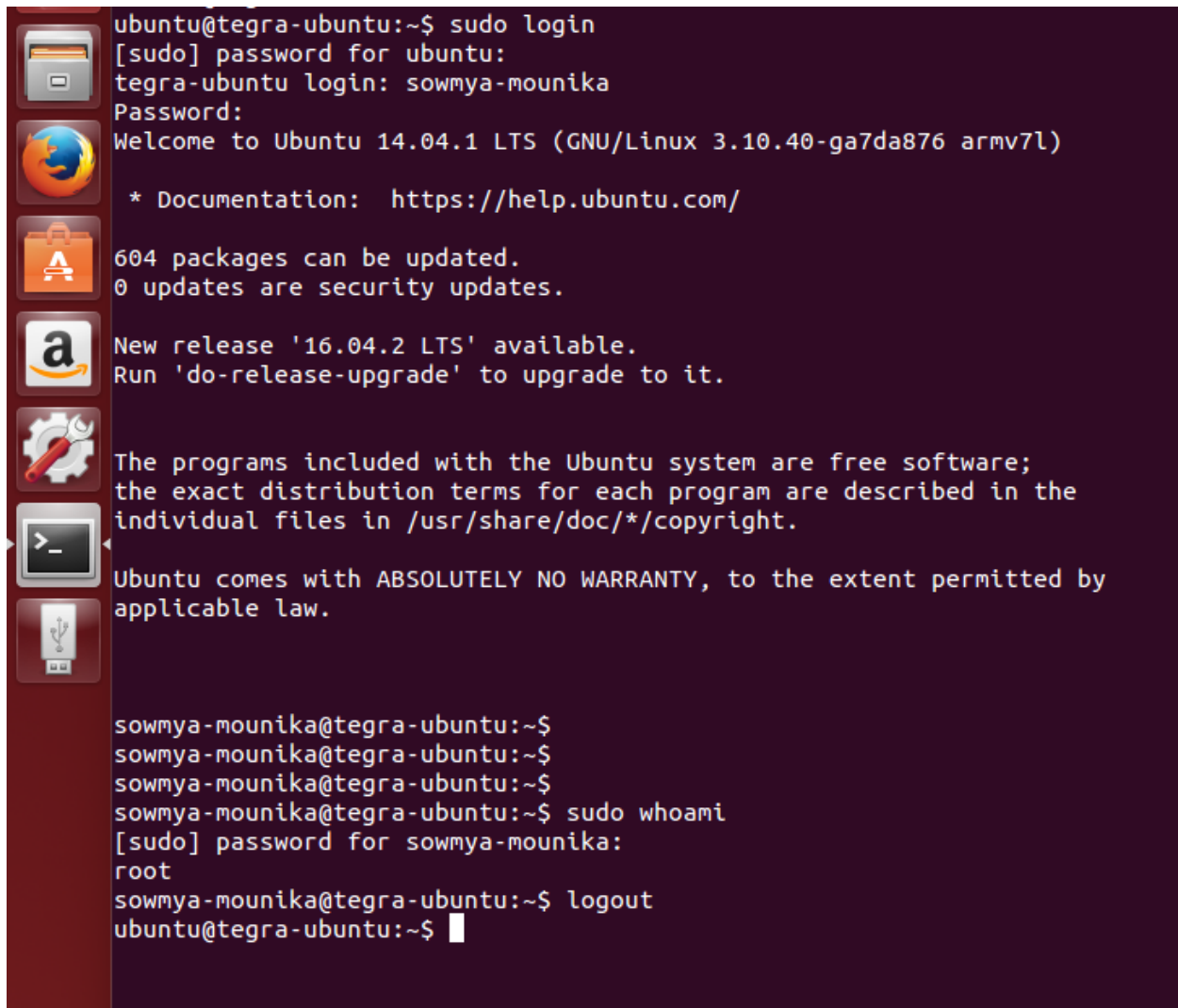
The command “sowmya-mounika ALL=(ALL:ALL) ALL” was used below the command for root, thus giving the account the same root privileges.

A terminal window titled "Terminal" with a dark background and light text. The window shows the content of the /etc/sudoers file. The text is as follows:

```
root@tegra-ubuntu: /home/ubuntu
#
# This file MUST be edited with the 'visudo' command as root.
#
# Please consider adding local content in /etc/sudoers.d/ instead of
# directly modifying this file.
#
# See the man page for details on how to write a sudoers file.
#
Defaults        env_reset
Defaults        mail_badpass
Defaults        secure_path="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/
sbin:/bin"
# Host alias specification
# User alias specification
# Cmnd alias specification
# User privilege specification
root    ALL=(ALL:ALL) ALL
sowmya-mounika  ALL=(ALL:ALL) ALL
# Members of the admin group may gain root privileges
-- INSERT --
```

On the left side of the terminal window, there is a vertical sidebar with several icons: a gear, a folder, a globe, a briefcase, an Amazon logo, a wrench and screwdriver, a terminal icon, and a USB icon. The terminal icon is currently selected. In the bottom right corner of the terminal window, the text "21,34" and "Top" are visible.

Successful logout of Ubuntu and login into new account was done, and “sudo whoami” command was executed to demonstrate success. (The new account was established as root)

A terminal window with a dark purple background and a vertical sidebar on the left containing icons for file manager, web browser, application store, and system settings. The terminal text shows the process of logging in as 'ubuntu' and then switching to the 'sowmya-mounika' user. The 'sudo whoami' command is executed, resulting in 'root' being printed, indicating successful privilege escalation. The session ends with a 'logout' command.

```
ubuntu@tegra-ubuntu:~$ sudo login
[sudo] password for ubuntu:
tegra-ubuntu login: sowmya-mounika
Password:
Welcome to Ubuntu 14.04.1 LTS (GNU/Linux 3.10.40-ga7da876 armv7l)

 * Documentation:  https://help.ubuntu.com/

604 packages can be updated.
0 updates are security updates.

New release '16.04.2 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

sowmya-mounika@tegra-ubuntu:~$
sowmya-mounika@tegra-ubuntu:~$
sowmya-mounika@tegra-ubuntu:~$
sowmya-mounika@tegra-ubuntu:~$ sudo whoami
[sudo] password for sowmya-mounika:
root
sowmya-mounika@tegra-ubuntu:~$ logout
ubuntu@tegra-ubuntu:~$
```

2.

Read the paper "[Architecture of the Space Shuttle Primary Avionics Software System](#)" [available on D2L], by Gene Carlow and provide an explanation and critique of the frequency executive architecture. What advantages and disadvantages does the frequency executive have compared to the real-time threading and tasking implementation methods for real-time software systems? Please be specific about the advantages and disadvantages and provide at least 3 advantages as well as 3 disadvantages.

Answer:

Explanation and critique of the frequency executive architecture

The architecture of NASA Space Shuttle's flight control software system (PASS-Primary Avionics Software System) is described and detailed in this paper. This is an example where successful accommodation of diverse requirements requires and leads to importance on system architecture. The final system is a result of various layers of development. The various layers range from the base of the operating system till the user interface layer. This system provides a high level of abstraction. Since the system is mission-critical, it had many redundancies built into it. The reasons for various design decisions made during the design of system software are detailed in the paper.

Due to the constraint on main memory (Less amount of memory), the full code could not be loaded onto it. An arrangement was made so code and data could be moved into the GPC main memory only on the initiation of the OPS-this is very much like the concept of modern day virtual memory. This arrangement was the segmentation of the software system into eight different parts called OPS. The segmentation was based on individual functions and their requirements in time. Each OPS would be loaded onto the main memory only when required, and each OPS code would otherwise be stored in memory outside of main memory.

The system software's operational sequence structure has three different modes to communicate with the computers to monitor the avionic system.

A major mode divides the primary processing within the OPS into major steps, a specialist function(SPEC) is initiated upon keyboard key entry which executes concurrently with other processes to perform background functions and a display function monitors processing results of major mode functions and the specialist function.

In flight software systems, the design is usually synchronous-each application process is dispatched at the correct and exact instant at which it is to be executed.

Thus, an executive architecture is sufficient for this purpose as it gives the benefit of repeatability. However, there is very limited flexibility for changes. Since space shuttle computers perform many different applications and face many requirement changes, the PASS system adopted a non-synchronous approach along with the decision of isolating application processes from external I/O.

A user interface is present for communication between systems and application software. The dispatching of each application process is timed to always occur at a specific point relative to the start and completion of the different application processes.

The Flight Control OS(FCOS) primarily performs process management to control allocation of internal computer resources and uses a multitasking priority queue to schedule and allocate CPU resources. This queue structure ensures that the highest priority process with work to perform is given CPU resources to accomplish the task.

Apart from this, FCOS also performs input/output management to control allocation of input/output processor resources. The on-board software has been developed to perform three important functions of Guidance, Navigation and Control (GN&C), vehicle systems management and vehicle checkout. These are the Primary Avionics Peripheral Subsystems which use the hard real-time services where the frequency executive design is used in the software. The on-board software is a cyclic closed loop application which performs various functions with tight timing and phasing. Different applications are performed at different frequencies depending on their priorities and need to be serviced. There is a cyclic process (executive) that uses a dispatching design technique which is table-driven to control and manage various services and tasks.

The high frequency executive structure has been given higher priorities than the low frequency executive structures. The scheduling is takes the following order: high frequency structure > medium frequency structure > low frequency executives. This is actually not a completely feasible technique because the deadlines are not considered while the scheduling is done. If high frequency executives only get executed, there will be no implementation a long time for the low frequency executives.

Critical flight control processing must be dispatched at 25Hz to complete within 40-millisecond cycle and there is initiation of all principle vehicle control processes. There are mid-frequency and low-frequency executives too whose dispatching rates go from 6.25Hz to 0.25Hz. Cyclic loops are essentially implemented as a main loop with an invariant loop body. This loop may contain polling for events to determine when to dispatch functions and those functions that need to run more frequently will be called more times in the loop. This executive can be extended to handle asynchronous events with interrupts rather than only performing loop-based polling of inputs.

Overall, the paper gives a good description of a complex software architecture.

A large number of good software constructs and practices, including cyclic executives, have been implemented to maintain functionality and get the correct timing in a mission-critical system.

Advantages of frequency executives over threads/tasks for Real-time software systems:

- The frequency executive design is simpler compared to RTOS and all the scheduling algorithms that go with them. This is because it can be considered to be a form of

cooperative multitasking that has only a single task and hence dispatching of the task is efficient and straightforward.

- There is little context switching involved in cyclic executives as it is a form of non-pre-emptive tasking. Context switching is a major contributor to latencies involved in the system that hamper the ability to service a request on time and cyclic executive are spared from that overhead.
- Cyclic executives are highly deterministic. The schedule has to be only stored in a table which can be read by the executive. Hence, the frequency for deadlines to miss is less. A highly predictable system is obtained that can be validated and tested.
- The frequency executive prevents overflow runs at process levels or system levels. High repeatability is achieved because of this phenomenon because the deadlines are met more frequently. Thus, stability with less bugs and malfunctioning is achieved.
- Having a table for frequencies of each task avoids the overhead that may be present in a dynamic priority scheduler which computes next task that must be executed. Lesser chance of corruption of data as a sequence of and time taken for execution of tasks is fixed.
- There are no unknown interrupts and hence improve the efficiency because the flow of code is not interrupted. There are also no interferences thereby giving a high output efficiency.

Disadvantages of frequency executives over threads/tasks for Real-time software systems:

- One of the major disadvantages of cyclic schedules is the limited flexibility – Cyclic Executive cannot deal with run-time changes. Also, any new service requires a new entry in the table to be computed and that can be cumbersome and inefficient. If there is a structural change in the executives, the task may go out of preemption.
- Since it is a non-pre-emptive style of tasking, any long but low priority task may prevent a higher priority task from running for a long time – this is a dangerous situation for critical missions.
- Splitting big tasks into subtasks and determining the scheduling blocks for each tasks is a time-consuming process. It is also difficult and there is a possibility of introducing errors into the process. This is in contrast to most modern-day OS's using pre-emptive scheduling and splitting a big problem into a number of threads or tasks.
- Unable to handle asynchronous requests might lead to delayed response to any interrupt, ultimately may or may not lead to missing deadlines in the worst case.
- Harmonic tasks get to have some hidden costs and so there is not full utilization of resources.
- Tasks may not get enough time to execute due to the longest period being greater than the required time to process the same. This may alter the preemptions and the tasks may become non-feasible.

3.

Read the paper “Building Safety-Critical Real-Time Systems with Re-useable Cyclic Executives”, available from [http://dx.doi.org/10.1016/S0967-0661\(97\)00088-9](http://dx.doi.org/10.1016/S0967-0661(97)00088-9). In other embedded systems classes you built ISR (Interrupt Service Routine) processing software and polling/control loops to control for example stepper motors – describe the concept of the Cyclic Executive and how this compares to the Linux POSIX RT threading and RTOS approaches we have discussed.

Answer:

Many real-time systems have strict safety requirements, and concurrent processes cannot be used in their development. These safety-critical systems are developed using synchronous architectures based on cyclic scheduling. The paper describes a reusable cyclic executive implementation in Ada 95(programming language), based on a generic architecture for synchronous real-time systems. This generic architecture is described by means of an object-oriented design notation. New Ada characteristics, as well as exceptions and generics, are used to build the component blocks of this generic architecture. In order to develop real-time systems, guidelines for using these reusable components are also provided.

The paper articulates the requirements for generic cyclic executives, then goes on to describe structure of the cyclic scheduler, including exceptions, and then describes Cyclic Executive as a generic architecture for synchronous real-time systems, following which the Ada 95 implementation is discussed.

Cyclic Executive

Cyclic Executive is an alternative to a Real-time operating system. A repeating sequence of activities is cycled through at a set frequency. It is a form of co-operative multitasking.

Cyclic Executive consists of a table having the entire execution pattern of a system. It consists of major cycles, made up of serial minor cycles, which further consist of a variable number of frames of variable duration. It is an iterative procedure that allows explicit multiplexing of a periodic process set in one processor. The major schedule defines the process sequence for a major cycle and this is repeated cyclically, and will continue till the operating mode of the system encounters changes.

The minor and major schedules have a synchronized clock of minor and major frames. Each frame within a minor cycle allows only one process of the sequence of the minor schedule to execute. The user, though, can define new exceptions, and predefined exceptions can be explicitly activated.

Example: Overruns which occur in a real time system.

When an overrun occurs, an exception is raised and serviced. Thus, using this mechanism, the main loop structure can be coded in a clean and simple way.

Advantages:

- Unexpected interrupts do not cause much overhead.
- Less jitter.
- Possible to predict the flow of the system (Without considering exceptions).
- No individual task can be preempted.

Linux POSIX RT

Linux POSIX RT is an RTOS microkernel that runs as a fully preemptive process. It implements a POSIX API for thread creation and manipulation. Thread attributes can be set. Another key point in RT Linux is that the real time tasks have direct access to memory and do not use any virtual memory. It has multiple scheduling policies like SCHED_FIFO and SCHED_RR which are preemptive in nature.

Advantages:

- The major advantage over Cyclic Executive is the predictability of tasks with respect to the application's external requirements, even when the requirements are changing.
- More stability is present in the system because time constraints can be pre-identified. Higher priority tasks will continue to run if the utilization has not exceeded the time constraints.
- The frequencies of various tasks do not require to be harmonic. Thus, it is possible to use periodic tasks with their natural frequencies without performance degradation.
- There is no need to break the tasks to meet the frame length limitations. Each task can be executed for as long as necessary; they are constrained by their own utilization specifications.

Comparison between Cyclic Executive and RTOS

- 1) Cyclic executive is a synchronous architecture and is best when safety is a critical issue since there is no arbitrary interleaving of operations. The behavior of the system is completely predictable which is not possible in Linux POSIX RT threading, as the threads will interleave in the middle and synchronization issues for shared resources, deadlocks make them hard to be reliable.
- 2) Disadvantage of cyclic executive is the low abstraction level of the resulting software which makes it difficult and costly to develop and maintain but in Linux, POSIX RT takes care of abstraction level for different architectures.
- 3) The Cyclic Executive architecture sounds very complex, but in Linux POSIX RT threading it is simple and easy because all that matters is how many services are there and if any shared resources are present among them.
- 4) The concept of recovery group- where, if a process fails, then the whole group is considered to have failed, and a recovery process is started - provides degraded

functionality of group. This concept of recovery process is not there for user level programming but if it is an internal error then Linux will take care of it.

- 5) Concept of exceptions exist in cyclic executive and Linux POSIX RT threading. But in cyclic executive they have to deal with additional exceptions of minor cycle overrun, frame overrun.

4.

Download [Feasibility example code](#) and build it on a Jetson, DE1-SoC or TIVA or Virtual Box and execute the code. Compare the tests provided to analysis using Cheddar for the first 4 examples. Now, implement the remaining examples [5 more] that we reviewed in class ([found here](#)). Complete analysis for all three policies using Cheddar (RM, EDF, LLF). In cases where RM fails, but EDF or LLF succeeds, explain why. Cheddar uses both service simulations over the LCM of the periods as well as feasibility analysis based on the RM LUB and scheduling-point/completion-test algorithms, referred to as “Worst Case Analysis”. Does your modified Feasibility code agree with Cheddar analysis in all 5 additional cases? Why or why not?

Answer:

The Feasibility example code (Dr. Siewart’s) for Rate Monotonic Scheduling was downloaded and built on Linux. This code was for Examples 0, 1, 2 and 3. The code was then extended to implement Examples 4, 5, 6, 7 and 8 and was executed. Hence, feasibility results were obtained for Examples 0 to 8 for both Scheduling—Point as well as Completion tests.

This code runs the completion time test and scheduling point test for Rate monotonic policy for the given examples. The code has functions for completion time and scheduling point tests and arrays of WCETs (worst case execution times), Time periods and Deadlines of tasks are passed as parameters to return whether the given example is feasible or not.

In Scheduling Point test tests, if a set of services can be shown to meet all deadlines from the critical instant up to the longest deadline of all tasks in the set, then the set is feasible. The completion time test requires the total cumulative demand from all higher priority tasks, up to some time t , to be less than or equal to the deadline for a given service, for that service to be feasible. If all the services in the set are feasible, the set is feasible.

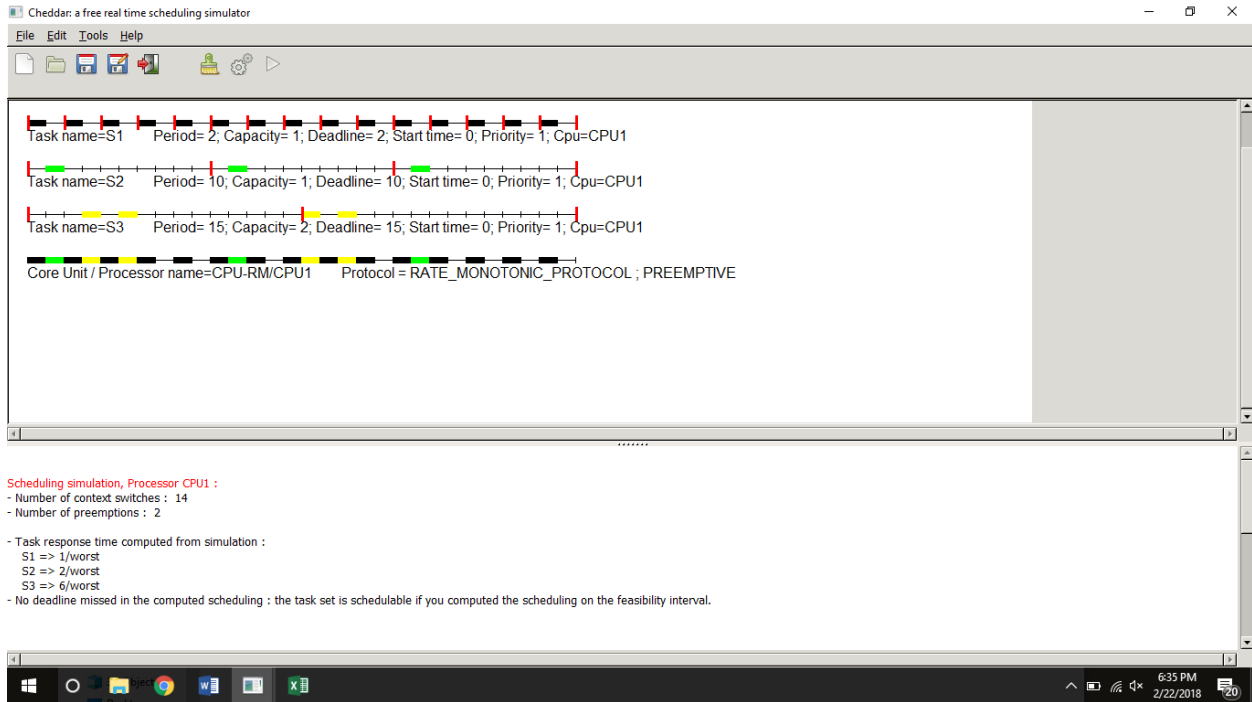
The screenshot of execution for Examples 0 to 8 for both the tests (**Rate Monotonic Scheduling**) is as shown below:

```
sowmya@sowmya-HP-Pavilion-Notebook: ~/Desktop/Feasibility_code
sowmya@sowmya-HP-Pavilion-Notebook:~$ cd Desktop
sowmya@sowmya-HP-Pavilion-Notebook:~/Desktop$ cd Feasibility_code
sowmya@sowmya-HP-Pavilion-Notebook:~/Desktop/Feasibility_code$ make all
gcc -O0 -g -c feasibility_tests.c
gcc -O0 -g -o feasibility_tests feasibility_tests.o -lm
sowmya@sowmya-HP-Pavilion-Notebook:~/Desktop/Feasibility_code$ ./feasibility_tests
***** Completion Test Feasibility Example
Ex-0 U=0.73 (C1=1, C2=1, C3=2; T1=2, T2=10, T3=15; T=D): FEASIBLE
Ex-1 U=0.84 (C1=1, C2=1, C3=2; T1=2, T2=5, T3=7; T=D): INFEASIBLE
Ex-2 U=1.00 (C1=1, C2=1, C3=1, C4=2; T1=2, T2=5, T3=7, T4=13; T=D): INFEASIBLE
Ex-3 U=0.93 (C1=1, C2=2, C3=3; T1=3, T2=5, T3=15; T=D): FEASIBLE
Ex-4 U=1.00 (C1=1, C2=1, C3=4; T1=2, T2=4, T3=16; T=D): FEASIBLE
Ex-5 U=1.00 (C1=1, C2=2, C3=1; T1=2, T2=5, T3=10; T=D): FEASIBLE
Ex-6 U=1.00 (C1=1, C2=1, C3=1, C4=2; T1=2, T2=5, T3=7, T4=13; T=D): INFEASIBLE
Ex-7 U=1.00 (C1=1, C2=2, C3=4; T1=3, T2=5, T3=15; T=D): FEASIBLE
Ex-8 U=1.00 (C1=1, C2=1, C3=1, C4=2; T1=2, T2=5, T3=7, T4=13; T=D): INFEASIBLE
***** Scheduling Point Feasibility Example
Ex-0 U=0.73 (C1=1, C2=1, C3=2; T1=2, T2=10, T3=15; T=D): FEASIBLE
Ex-1 U=0.84 (C1=1, C2=1, C3=2; T1=2, T2=5, T3=7; T=D): INFEASIBLE
Ex-2 U=1.00 (C1=1, C2=1, C3=1, C4=2; T1=2, T2=5, T3=7, T4=13; T=D): INFEASIBLE
Ex-3 U=0.93 (C1=1, C2=2, C3=3; T1=3, T2=5, T3=15; T=D): FEASIBLE
Ex-4 U=1.00 (C1=1, C2=1, C3=4; T1=2, T2=4, T3=16; T=D): FEASIBLE
Ex-5 U=1.00 (C1=1, C2=2, C3=1; T1=2, T2=5, T3=10; T=D): FEASIBLE
Ex-6 U=1.00 (C1=1, C2=1, C3=1, C4=2; T1=2, T2=5, T3=7, T4=13; T=D): INFEASIBLE
Ex-7 U=1.00 (C1=1, C2=2, C3=4; T1=3, T2=5, T3=15; T=D): FEASIBLE
Ex-8 U=1.00 (C1=1, C2=1, C3=1, C4=2; T1=2, T2=5, T3=7, T4=13; T=D): INFEASIBLE
sowmya@sowmya-HP-Pavilion-Notebook:~/Desktop/Feasibility_code$
```

Analysis was done for the Examples using Cheddar tool. Cheddar, a GPL real time scheduling tool, is used to compare and verify results from running the code. Cheddar allows us to model software architectures of real-time systems to check their schedulability or other performance criteria. It does this by scheduling simulations and/or feasibility tests. We use cheddar to generate scheduling simulations and also perform feasibility tests for all examples, for RM, EDF and LLF scheduling policies. The results obtained from cheddar are similar to those obtained by running feasibility tests in code.

The screenshots of Cheddar implementation for **Rate Monotonic Scheduling** of Examples 0 to 8 are as shown below.

Example 0 – Cheddar



Example 0 – Feasibility

Scheduling feasibility, Processor CPU1 :

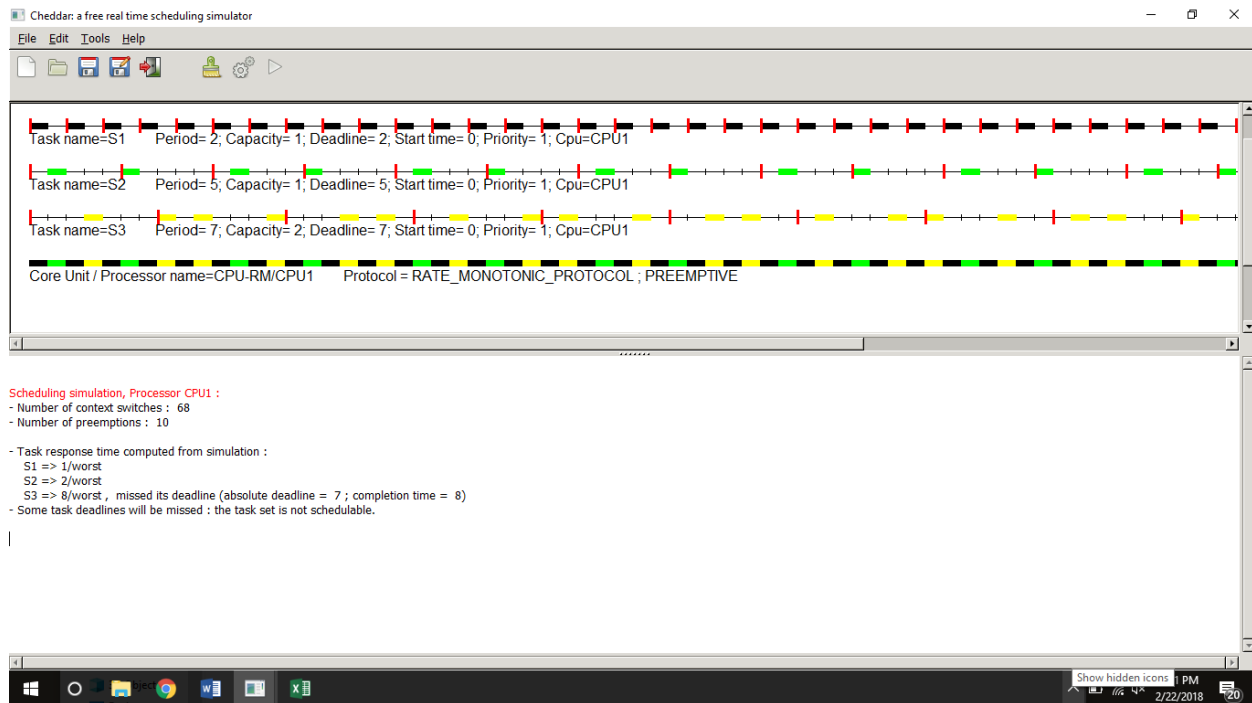
1) Feasibility test based on the processor utilization factor :

- The hyperperiod is 30 (see [18], page 5).
- 8 units of time are unused in the hyperperiod.
- Processor utilization factor with deadline is 0.73333 (see [1], page 6).
- Processor utilization factor with period is 0.73333 (see [1], page 6).
- In the preemptive case, with RM, the task set is schedulable because the processor utilization factor 0.73333 is equal or less than 0.77976 (see [1], page 16, theorem 8).

2) Feasibility test based on worst case response time for periodic tasks :

- Worst Case task response time : (see [2], page 3, equation 4).
 - S3 => 6
 - S2 => 2
 - S1 => 1
- All task deadlines will be met : the task set is schedulable.

Example 1 – Cheddar



Example 1 – Feasibility

Scheduling feasibility, Processor CPU1 :

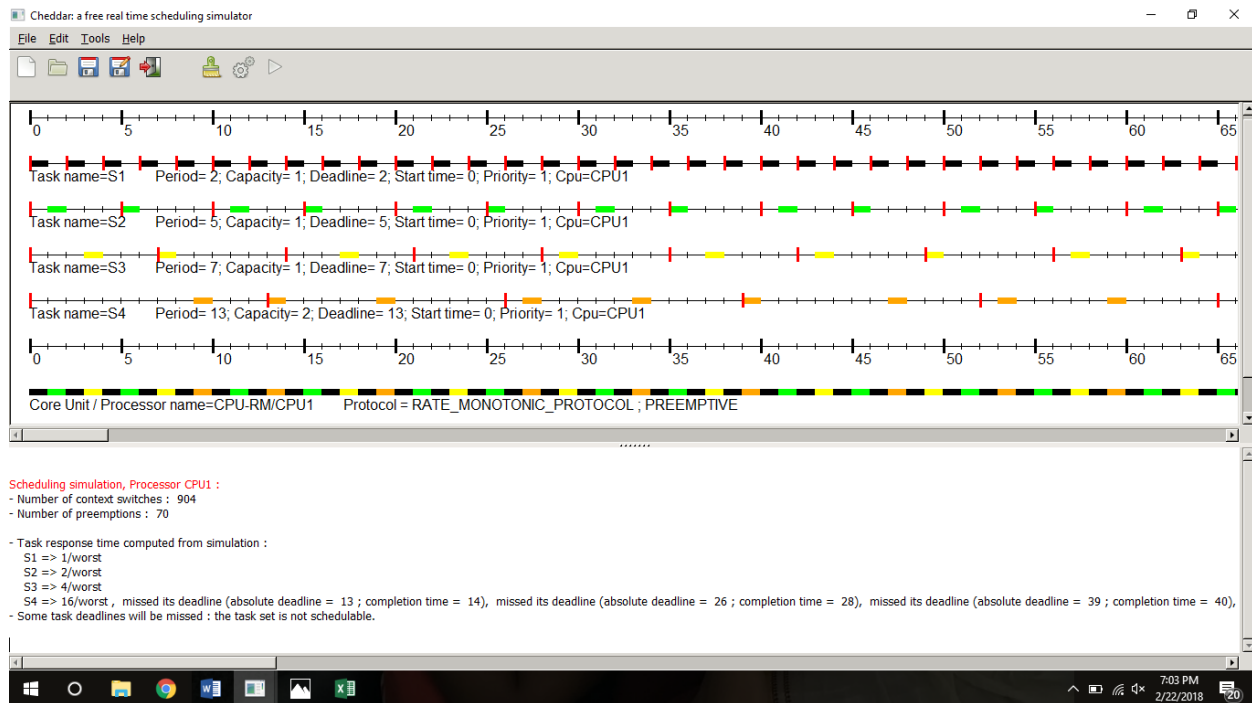
1) Feasibility test based on the processor utilization factor :

- The hyperperiod is 70 (see [18], page 5).
- 1 units of time are unused in the hyperperiod.
- Processor utilization factor with deadline is 0.98571 (see [1], page 6).
- Processor utilization factor with period is 0.98571 (see [1], page 6).
- In the preemptive case, with RM, we can not prove that the task set is schedulable because the processor utilization factor 0.98571 is more than 0.77976 (see [1], page 16, theorem 8).

2) Feasibility test based on worst case response time for periodic tasks :

- Worst Case task response time : (see [2], page 3, equation 4).
 - S3 => 8, missed its deadline (deadline = 7)
 - S2 => 2
 - S1 => 1
- Some task deadlines will be missed : the task set is not schedulable.

Example 2 – Cheddar



Example 2 – Feasibility

Scheduling feasibility, Processor CPU1 :

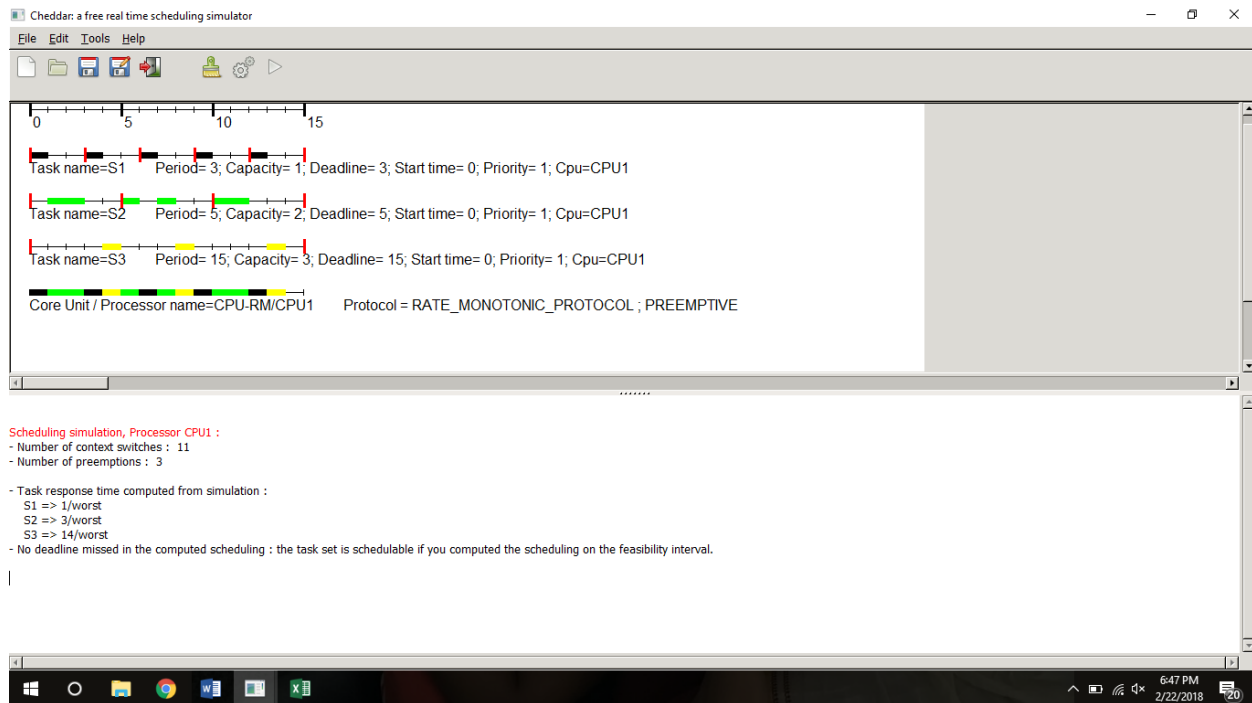
1) Feasibility test based on the processor utilization factor :

- The hyperperiod is 910 (see [18], page 5).
- 3 units of time are unused in the hyperperiod.
- Processor utilization factor with deadline is 0.99670 (see [1], page 6).
- Processor utilization factor with period is 0.99670 (see [1], page 6).
- In the preemptive case, with RM, we can not prove that the task set is schedulable because the processor utilization factor 0.99670 is more than 0.75683 (see [1], page 16, theorem 8).

2) Feasibility test based on worst case response time for periodic tasks :

- Worst Case task response time : (see [2], page 3, equation 4).
 - S4 => 16, missed its deadline (deadline = 13)
 - S3 => 4
 - S2 => 2
 - S1 => 1
- Some task deadlines will be missed : the task set is not schedulable.

Example 3 – Cheddar



Example 3 – Feasibility

Scheduling feasibility, Processor CPU1 :

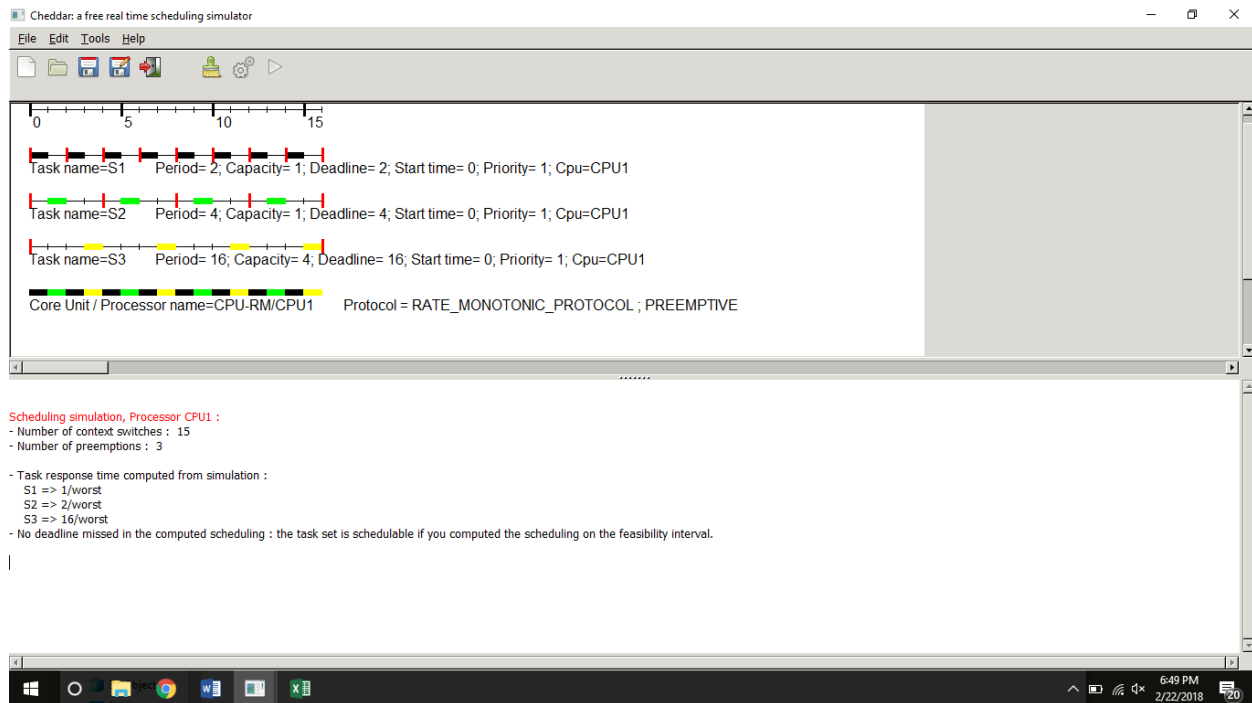
1) Feasibility test based on the processor utilization factor :

- The hyperperiod is 15 (see [18], page 5).
- 1 units of time are unused in the hyperperiod.
- Processor utilization factor with deadline is 0.93333 (see [1], page 6).
- Processor utilization factor with period is 0.93333 (see [1], page 6).
- In the preemptive case, with RM, we can not prove that the task set is schedulable because the processor utilization factor 0.93333 is more than 0.77976 (see [1], page 16, theorem 8).

2) Feasibility test based on worst case response time for periodic tasks :

- Worst Case task response time : (see [2], page 3, equation 4).
 - S3 => 14
 - S2 => 3
 - S1 => 1
- All task deadlines will be met : the task set is schedulable.

Example 4 – Cheddar



Example 4 – Feasibility

Scheduling feasibility, Processor CPU1 :

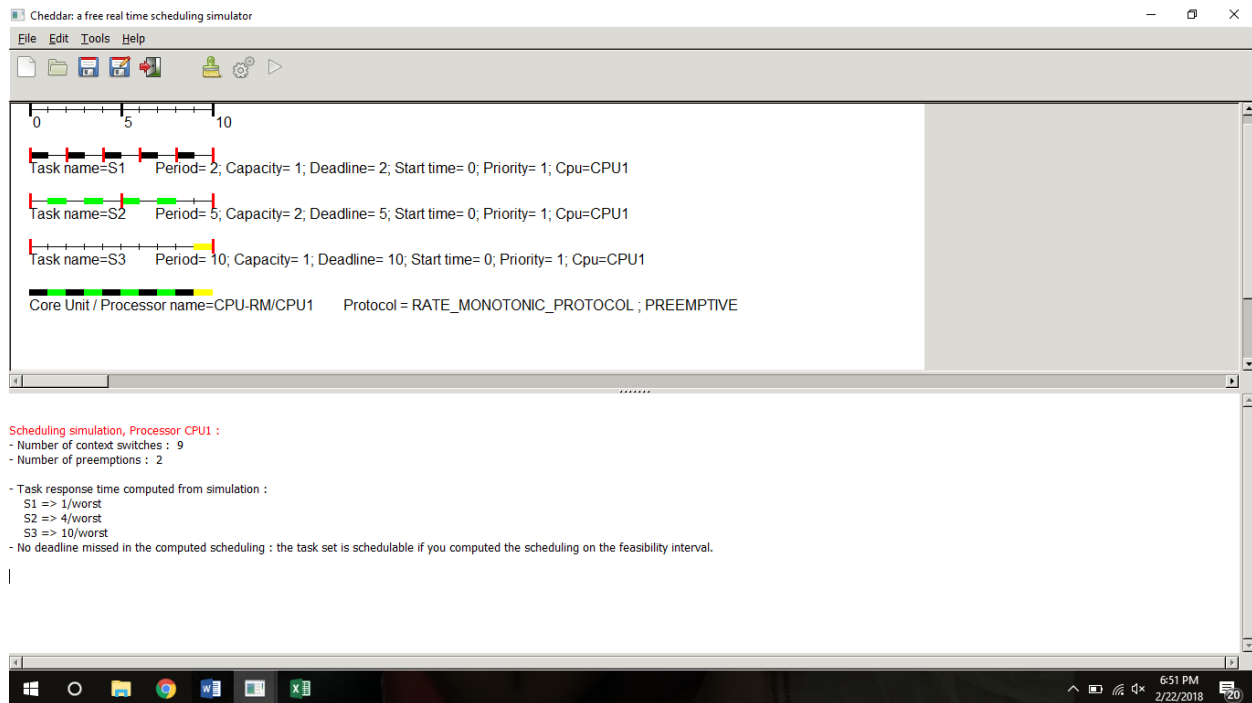
1) Feasibility test based on the processor utilization factor :

- The hyperperiod is 16 (see [18], page 5).
- 0 units of time are unused in the hyperperiod.
- Processor utilization factor with deadline is 1.00000 (see [1], page 6).
- Processor utilization factor with period is 1.00000 (see [1], page 6).
- In the preemptive case, with RM, the task set is schedulable because the processor utilization factor 1.00000 is equal or less than 1.00000 (see [19], page 13).

2) Feasibility test based on worst case response time for periodic tasks :

- Worst Case task response time : (see [2], page 3, equation 4).
 - S3 => 16
 - S2 => 2
 - S1 => 1
- All task deadlines will be met : the task set is schedulable.

Example 5 – Cheddar



Example 5 – Feasibility

Scheduling feasibility, Processor CPU1 :

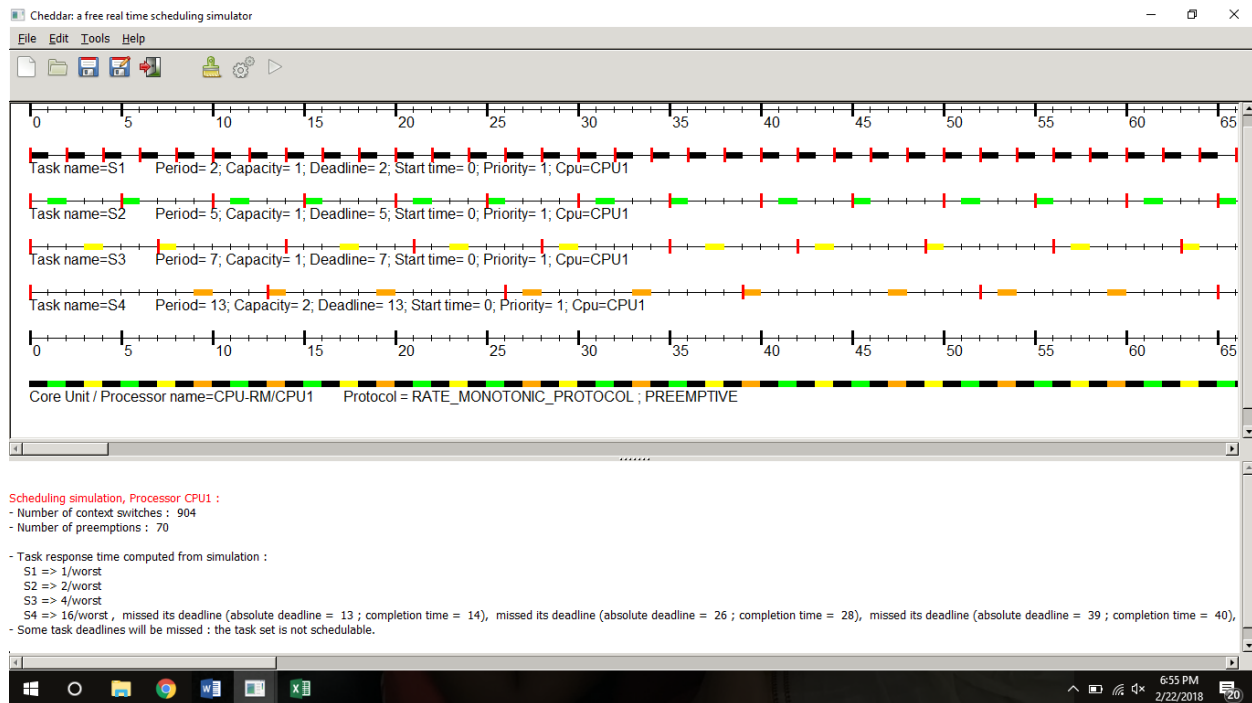
1) Feasibility test based on the processor utilization factor :

- The hyperperiod is 10 (see [18], page 5).
- 0 units of time are unused in the hyperperiod.
- Processor utilization factor with deadline is 1.00000 (see [1], page 6).
- Processor utilization factor with period is 1.00000 (see [1], page 6).
- In the preemptive case, with RM, we can not prove that the task set is schedulable because the processor utilization factor 1.00000 is more than 0.77976 (see [1], page 16, theorem 8).

2) Feasibility test based on worst case response time for periodic tasks :

- Worst Case task response time : (see [2], page 3, equation 4).
 - S3 => 10
 - S2 => 4
 - S1 => 1
- All task deadlines will be met : the task set is schedulable.

Example 6 – Cheddar



Example 6 – Feasibility

Scheduling feasibility, Processor CPU1 :

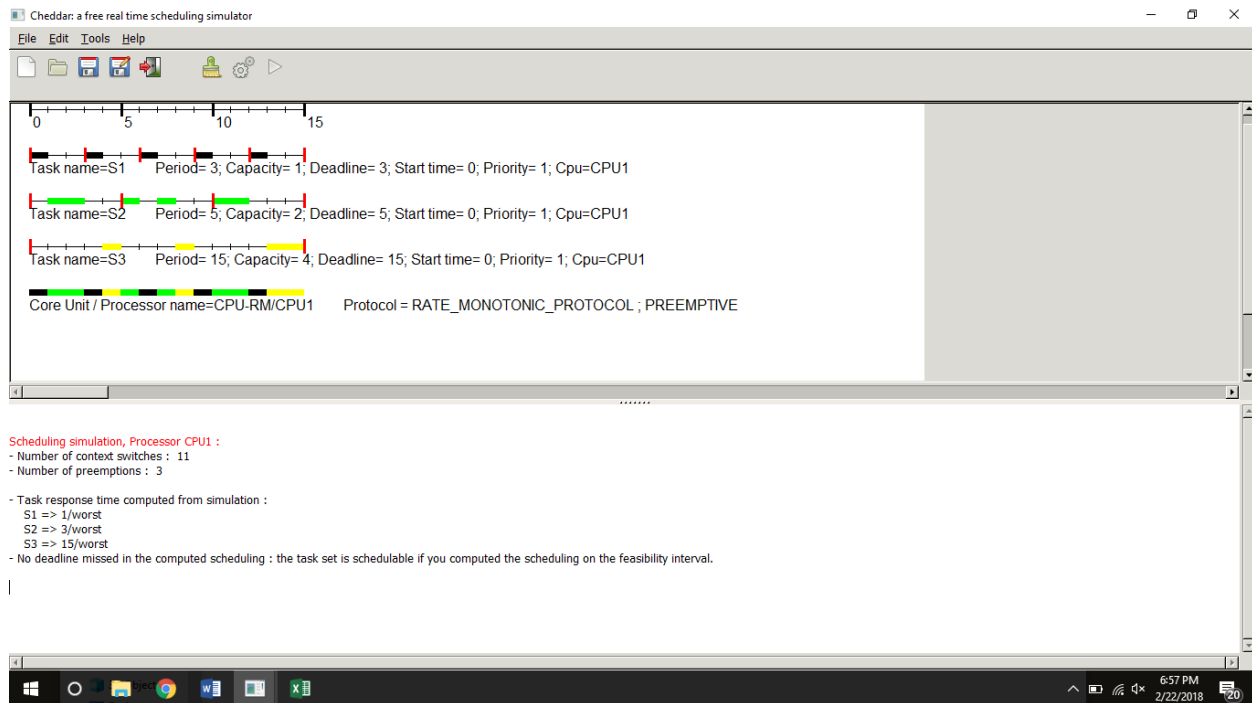
1) Feasibility test based on the processor utilization factor :

- The hyperperiod is 910 (see [18], page 5).
- 3 units of time are unused in the hyperperiod.
- Processor utilization factor with deadline is 0.99670 (see [1], page 6).
- Processor utilization factor with period is 0.99670 (see [1], page 6).
- In the preemptive case, with RM, we can not prove that the task set is schedulable because the processor utilization factor 0.99670 is more than 0.75683 (see [1], page 16, theorem 8).

2) Feasibility test based on worst case response time for periodic tasks :

- Worst Case task response time : (see [2], page 3, equation 4).
 - S4 => 16, missed its deadline (deadline = 13)
 - S3 => 4
 - S2 => 2
 - S1 => 1
- Some task deadlines will be missed : the task set is not schedulable.

Example 7 – Cheddar



Example 7 – Feasibility

Scheduling feasibility, Processor CPU1 :

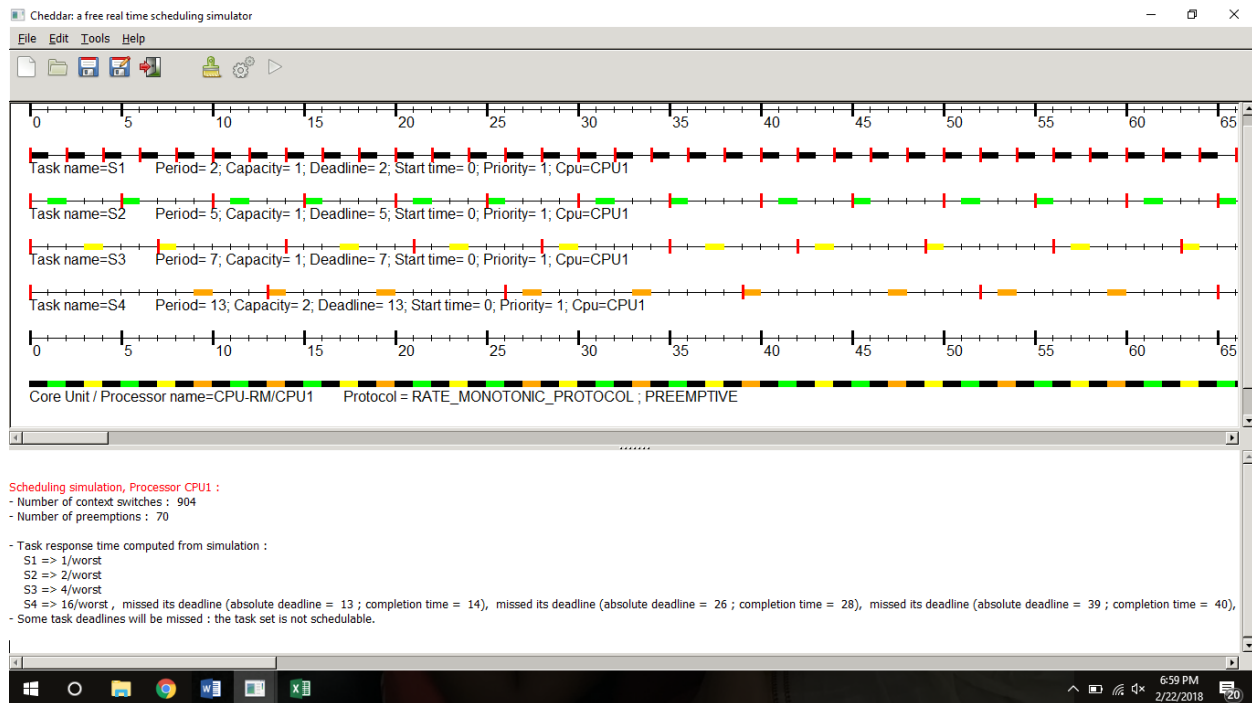
1) Feasibility test based on the processor utilization factor :

- The hyperperiod is 15 (see [18], page 5).
- 0 units of time are unused in the hyperperiod.
- Processor utilization factor with deadline is 1.00000 (see [1], page 6).
- Processor utilization factor with period is 1.00000 (see [1], page 6).
- In the preemptive case, with RM, we can not prove that the task set is schedulable because the processor utilization factor 1.00000 is more than 0.77976 (see [1], page 16, theorem 8).

2) Feasibility test based on worst case response time for periodic tasks :

- Worst Case task response time : (see [2], page 3, equation 4).
 - S3 => 15
 - S2 => 3
 - S1 => 1
- All task deadlines will be met : the task set is schedulable.

Example 8 – Cheddar



Example 8 – Feasibility

Scheduling feasibility, Processor CPU1 :

1) Feasibility test based on the processor utilization factor :

- The hyperperiod is 910 (see [18], page 5).
- 3 units of time are unused in the hyperperiod.
- Processor utilization factor with deadline is 0.99670 (see [1], page 6).
- Processor utilization factor with period is 0.99670 (see [1], page 6).
- In the preemptive case, with RM, we can not prove that the task set is schedulable because the processor utilization factor 0.99670 is more than 0.75683 (see [1], page 16, theorem 8).

2) Feasibility test based on worst case response time for periodic tasks :

- Worst Case task response time : (see [2], page 3, equation 4).
 - S4 => 16, missed its deadline (deadline = 13)
 - S3 => 4
 - S2 => 2
 - S1 => 1
- Some task deadlines will be missed : the task set is not schedulable.

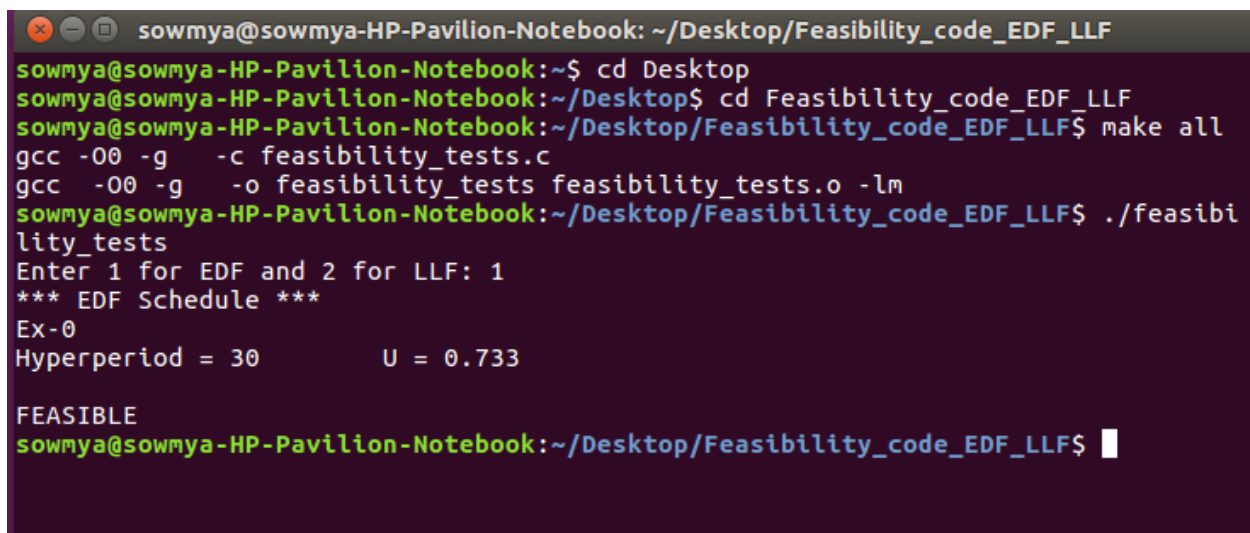
EDF and LLF

EDF and LLF schedule dynamically the tasks by changing priority keeping in mind the deadlines with necessary constraints. In EDF, the Earliest Deadline Task is implemented first while LLF executes the algorithm based on the laxity of the deadlines. EDF and LLF have higher overhead than Rate Monotonic.

EDF

The screenshot of execution for Examples 0 to 8 for **Earliest Deadline First Scheduling** is as shown below:

Example 0



```
sowmya@sowmya-HP-Pavilion-Notebook: ~/Desktop/Feasibility_code_EDF_LLFF
sowmya@sowmya-HP-Pavilion-Notebook:~$ cd Desktop
sowmya@sowmya-HP-Pavilion-Notebook:~/Desktop$ cd Feasibility_code_EDF_LLFF
sowmya@sowmya-HP-Pavilion-Notebook:~/Desktop/Feasibility_code_EDF_LLFF$ make all
gcc -O0 -g -c feasibility_tests.c
gcc -O0 -g -o feasibility_tests feasibility_tests.o -lm
sowmya@sowmya-HP-Pavilion-Notebook:~/Desktop/Feasibility_code_EDF_LLFF$ ./feasibility_tests
Enter 1 for EDF and 2 for LLF: 1
*** EDF Schedule ***
Ex-0
Hyperperiod = 30          U = 0.733

FEASIBLE
sowmya@sowmya-HP-Pavilion-Notebook:~/Desktop/Feasibility_code_EDF_LLFF$
```

Example 1



```
sowmya@sowmya-HP-Pavilion-Notebook:~/Desktop/Feasibility_code_EDF_LLFF$ ./feasibility_tests
Enter 1 for EDF and 2 for LLF: 1
*** EDF Schedule ***
Ex-1
Hyperperiod = 70          U = 0.986

FEASIBLE
sowmya@sowmya-HP-Pavilion-Notebook:~/Desktop/Feasibility_code_EDF_LLFF$
```

Example 2

```
sowmya@sowmya-HP-Pavilion-Notebook:~/Desktop/Feasibility_code_EDF_LLF$ ./feasibility_tests
Enter 1 for EDF and 2 for LLF: 1
*** EDF Schedule ***
Ex-2
Hyperperiod = 910          U = 0.997

FEASIBLE
sowmya@sowmya-HP-Pavilion-Notebook:~/Desktop/Feasibility_code_EDF_LLF$
```

Example 3

```
sowmya@sowmya-HP-Pavilion-Notebook:~/Desktop/Feasibility_code_EDF_LLF$ ./feasibility_tests
Enter 1 for EDF and 2 for LLF: 1
*** EDF Schedule ***
Ex-3
Hyperperiod = 15          U = 0.933

FEASIBLE
sowmya@sowmya-HP-Pavilion-Notebook:~/Desktop/Feasibility_code_EDF_LLF$
```

Example 4

```
sowmya@sowmya-HP-Pavilion-Notebook:~/Desktop/Feasibility_code_EDF_LLF$ ./feasibility_tests
Enter 1 for EDF and 2 for LLF: 1
*** EDF Schedule ***
Ex-4
Hyperperiod = 16          U = 1.000

FEASIBLE
sowmya@sowmya-HP-Pavilion-Notebook:~/Desktop/Feasibility_code_EDF_LLF$
```

Example 5

```
sowmya@sowmya-HP-Pavilion-Notebook:~/Desktop/Feasibility_code_EDF_LLF$ ./feasibility_tests
Enter 1 for EDF and 2 for LLF: 1
*** EDF Schedule ***
Ex-5
Hyperperiod = 10          U = 1.000

FEASIBLE
sowmya@sowmya-HP-Pavilion-Notebook:~/Desktop/Feasibility_code_EDF_LLF$
```

Example 6

```
sowmya@sowmya-HP-Pavilion-Notebook:~/Desktop/Feasibility_code_EDF_LLFS$ ./feasibility_tests
Enter 1 for EDF and 2 for LLF: 1
*** EDF Schedule ***
Ex-6
Hyperperiod = 910          U = 0.997

FEASIBLE
sowmya@sowmya-HP-Pavilion-Notebook:~/Desktop/Feasibility_code_EDF_LLFS$
```

Example 7

```
sowmya@sowmya-HP-Pavilion-Notebook:~/Desktop/Feasibility_code_EDF_LLFS$ ./feasibility_tests
Enter 1 for EDF and 2 for LLF: 1
*** EDF Schedule ***
Ex-7
Hyperperiod = 15          U = 1.000

FEASIBLE
sowmya@sowmya-HP-Pavilion-Notebook:~/Desktop/Feasibility_code_EDF_LLFS$
```

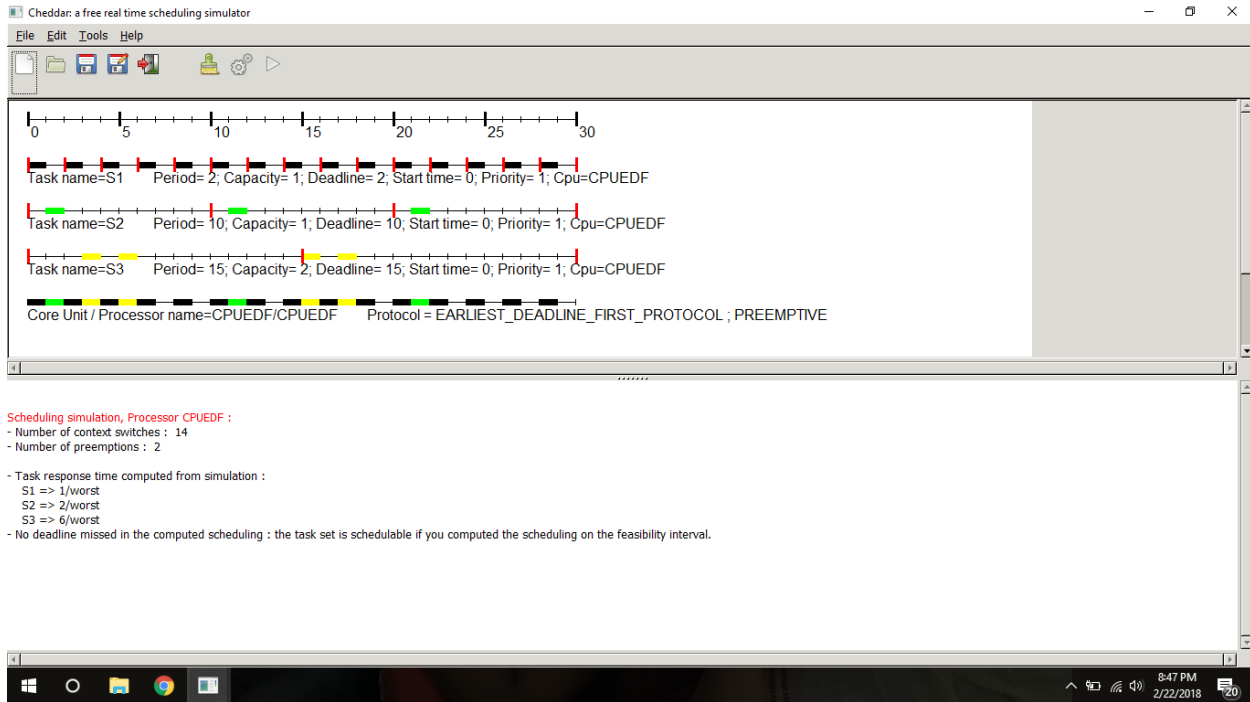
Example 8

```
sowmya@sowmya-HP-Pavilion-Notebook:~/Desktop/Feasibility_code_EDF_LLFS$ ./feasibility_tests
Enter 1 for EDF and 2 for LLF: 1
*** EDF Schedule ***
Ex-8
Hyperperiod = 910          U = 0.997

FEASIBLE
sowmya@sowmya-HP-Pavilion-Notebook:~/Desktop/Feasibility_code_EDF_LLFS$
```

The screenshots of Cheddar implementation for **Earliest Deadline First Scheduling** of Examples 0 to 8 are as shown below.

Example 0 – Cheddar



Example 0 – Feasibility

Scheduling feasibility, Processor CPUEDF :

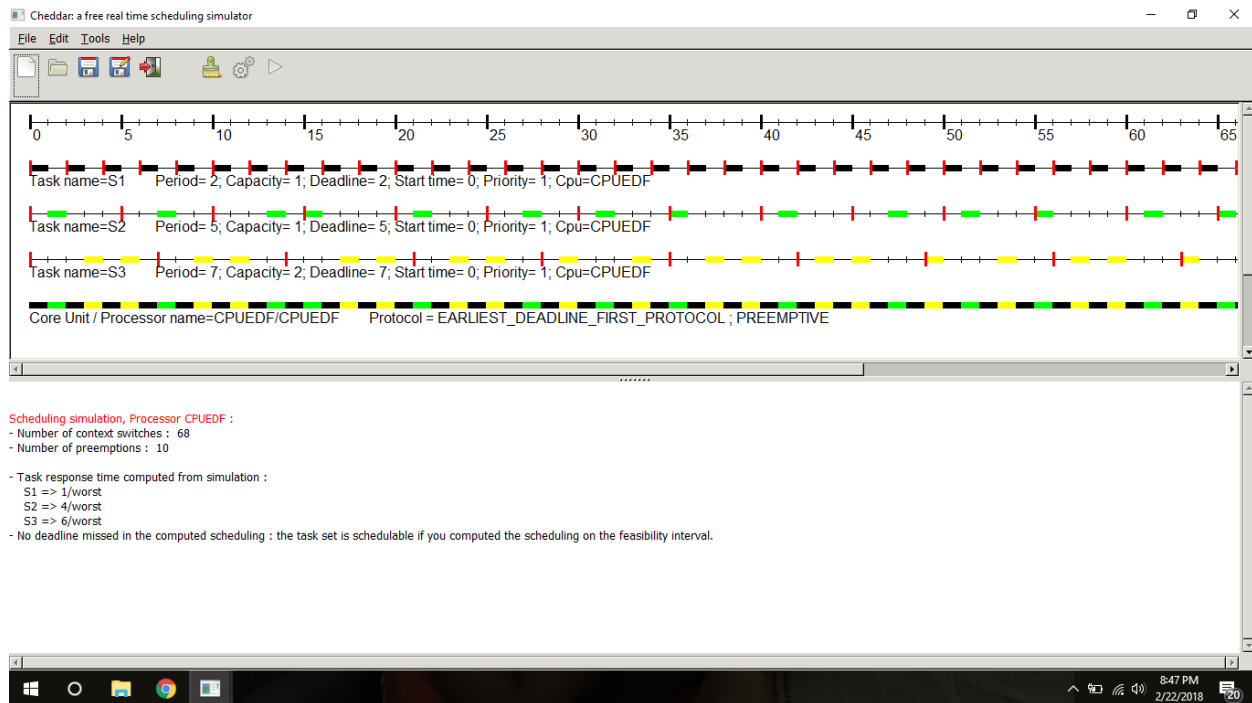
1) Feasibility test based on the processor utilization factor :

- The hyperperiod is 30 (see [18], page 5).
- 8 units of time are unused in the hyperperiod.
- Processor utilization factor with deadline is 0.73333 (see [1], page 6).
- Processor utilization factor with period is 0.73333 (see [1], page 6).
- In the preemptive case, with EDF, the task set is schedulable because the processor utilization factor 0.73333 is equal or less than 1.00000 (see [1], page 8, theorem 2).

2) Feasibility test based on worst case response time for periodic tasks :

- Worst Case task response time :
 - S1 => 1
 - S2 => 8
 - S3 => 13
- All task deadlines will be met : the task set is schedulable.

Example 1 – Cheddar



Example 1 – Feasibility

Scheduling feasibility, Processor CPUEDF :

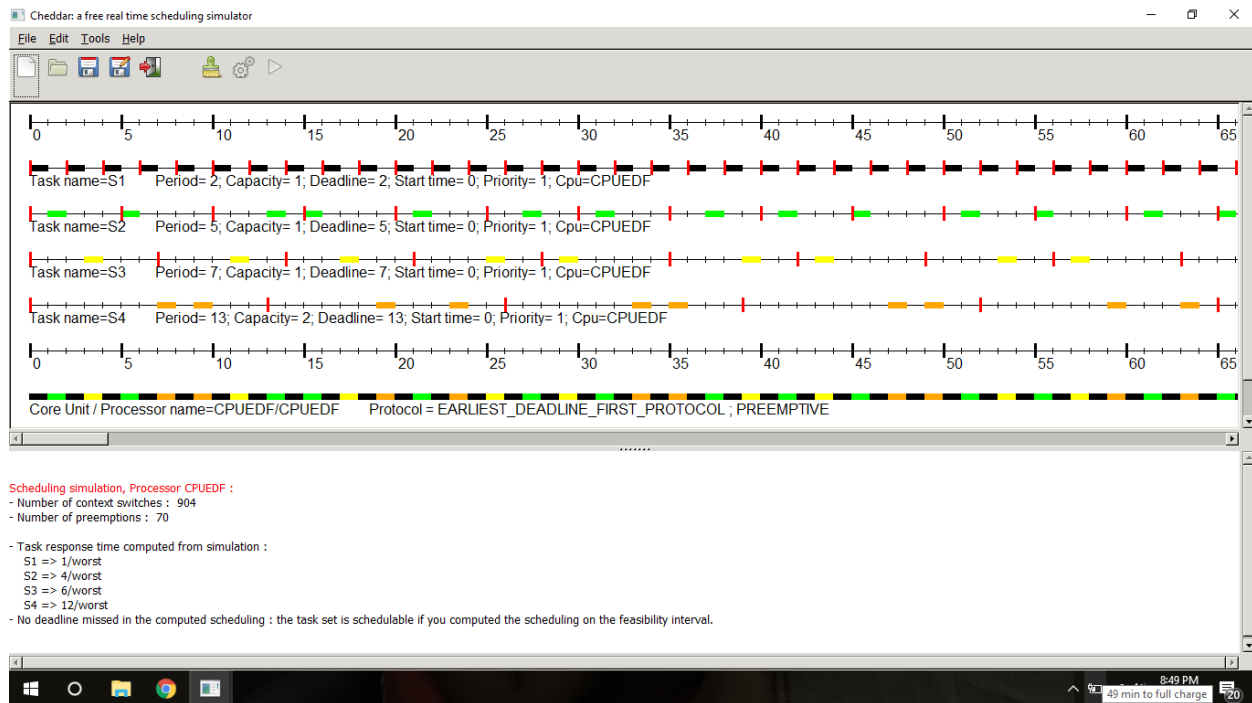
1) Feasibility test based on the processor utilization factor :

- The hyperperiod is 70 (see [18], page 5).
- 1 units of time are unused in the hyperperiod.
- Processor utilization factor with deadline is 0.98571 (see [1], page 6).
- Processor utilization factor with period is 0.98571 (see [1], page 6).
- In the preemptive case, with EDF, the task set is schedulable because the processor utilization factor 0.98571 is equal or less than 1.00000 (see [1], page 8, theorem 2).

2) Feasibility test based on worst case response time for periodic tasks :

- Worst Case task response time :
S1 => 1
S2 => 4
S3 => 6
- All task deadlines will be met : the task set is schedulable.

Example 2 – Cheddar



Example 2 – Feasibility

Scheduling feasibility, Processor CPUEDF :

1) Feasibility test based on the processor utilization factor :

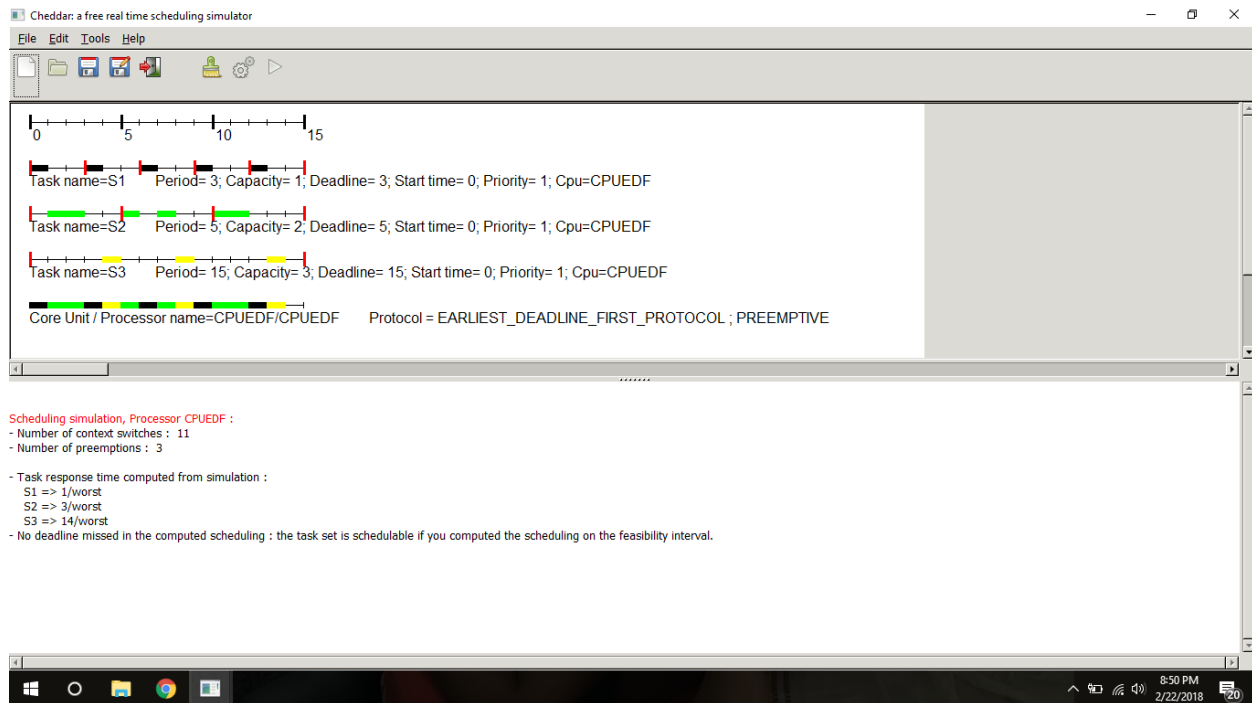
- The hyperperiod is 910 (see [18], page 5).
- 3 units of time are unused in the hyperperiod.
- Processor utilization factor with deadline is 0.99670 (see [1], page 6).
- Processor utilization factor with period is 0.99670 (see [1], page 6).
- In the preemptive case, with EDF, the task set is schedulable because the processor utilization factor 0.99670 is equal or less than 1.00000 (see [1], page 8, theorem 2).

2) Feasibility test based on worst case response time for periodic tasks :

- Worst Case task response time :

- S1 => 1
- S2 => 4
- S3 => 6
- S4 => 12
- All task deadlines will be met : the task set is schedulable.

Example 3 – Cheddar



Example 3 –Feasibility

Scheduling feasibility, Processor CPUEDF :

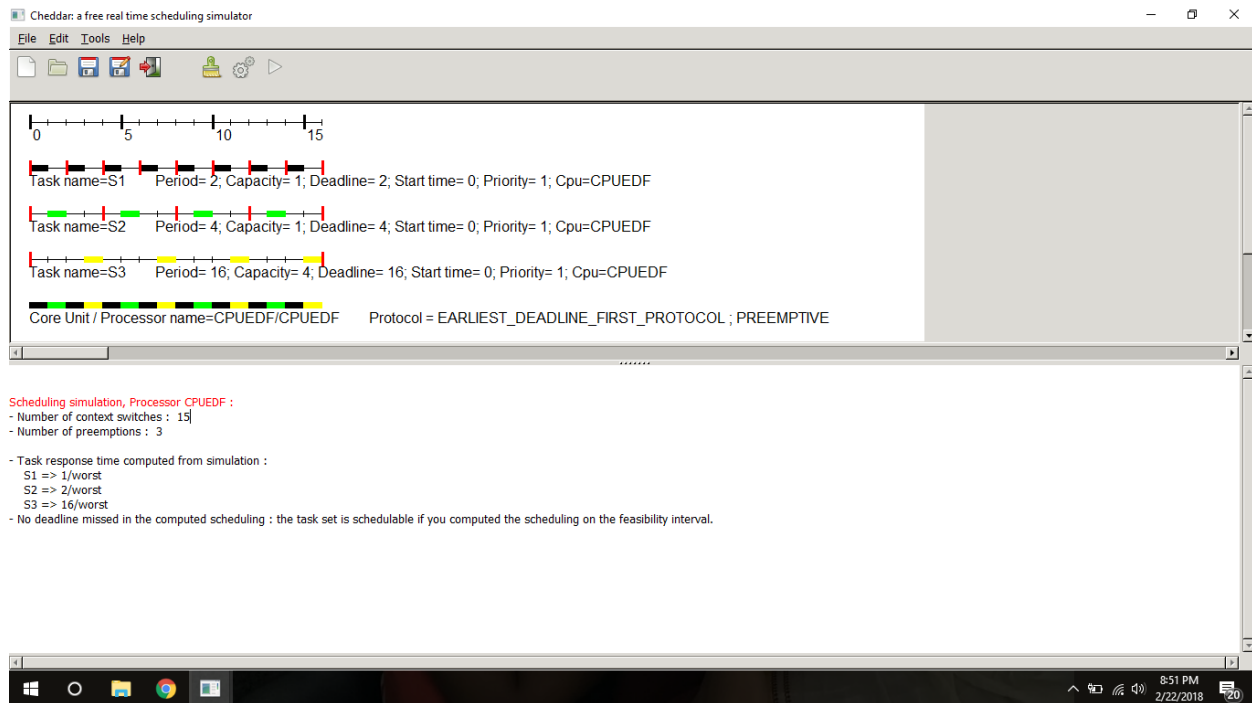
1) Feasibility test based on the processor utilization factor :

- The hyperperiod is 15 (see [18], page 5).
- 1 units of time are unused in the hyperperiod.
- Processor utilization factor with deadline is 0.93333 (see [1], page 6).
- Processor utilization factor with period is 0.93333 (see [1], page 6).
- In the preemptive case, with EDF, the task set is schedulable because the processor utilization factor 0.93333 is equal or less than 1.00000 (see [1], page 8, theorem 2).

2) Feasibility test based on worst case response time for periodic tasks :

- Worst Case task response time :
 - S1 => 2
 - S2 => 4
 - S3 => 14
- All task deadlines will be met : the task set is schedulable.

Example 4 – Cheddar



Example 4 – Feasibility

Scheduling feasibility, Processor CPUEDF :

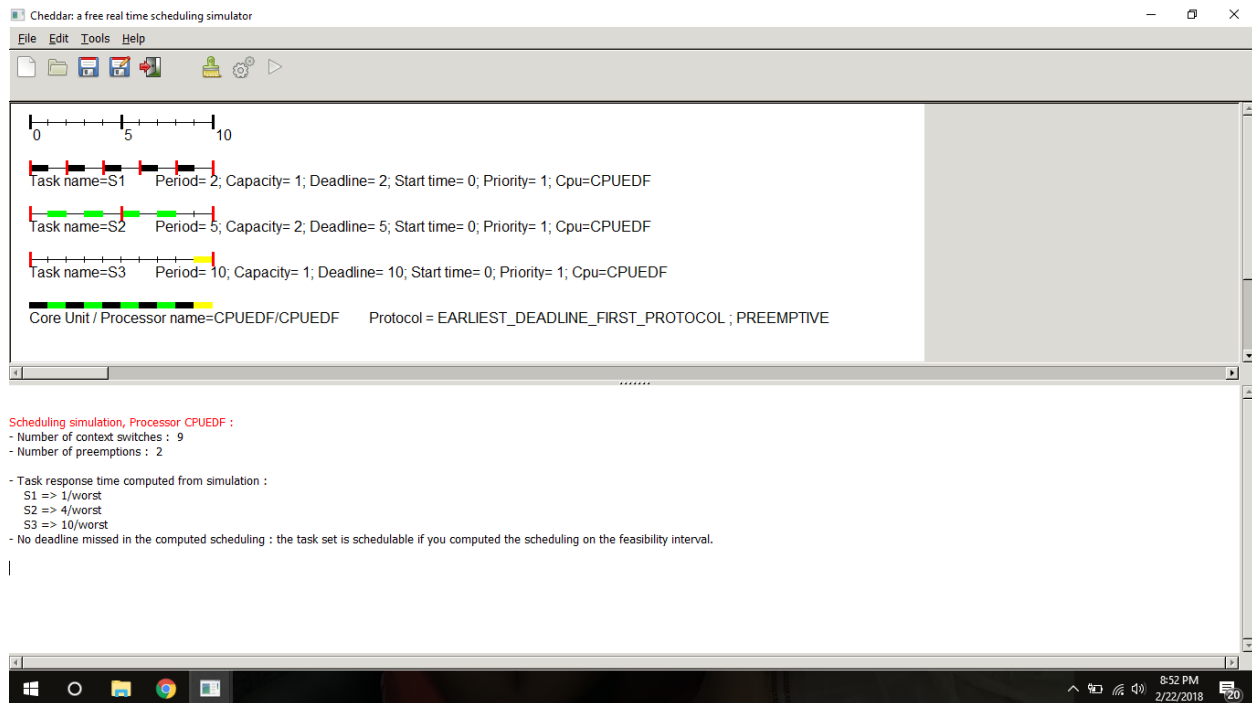
1) Feasibility test based on the processor utilization factor :

- The hyperperiod is 16 (see [18], page 5).
- 0 units of time are unused in the hyperperiod.
- Processor utilization factor with deadline is 1.00000 (see [1], page 6).
- Processor utilization factor with period is 1.00000 (see [1], page 6).
- In the preemptive case, with EDF, the task set is schedulable because the processor utilization factor 1.00000 is equal or less than 1.00000 (see [1], page 8, theorem 2).

2) Feasibility test based on worst case response time for periodic tasks :

- Worst Case task response time :
 - S1 => 2
 - S2 => 4
 - S3 => 16
- All task deadlines will be met : the task set is schedulable.

Example 5 – Cheddar



Example 5 – Feasibility

Scheduling feasibility, Processor CPUEDF :

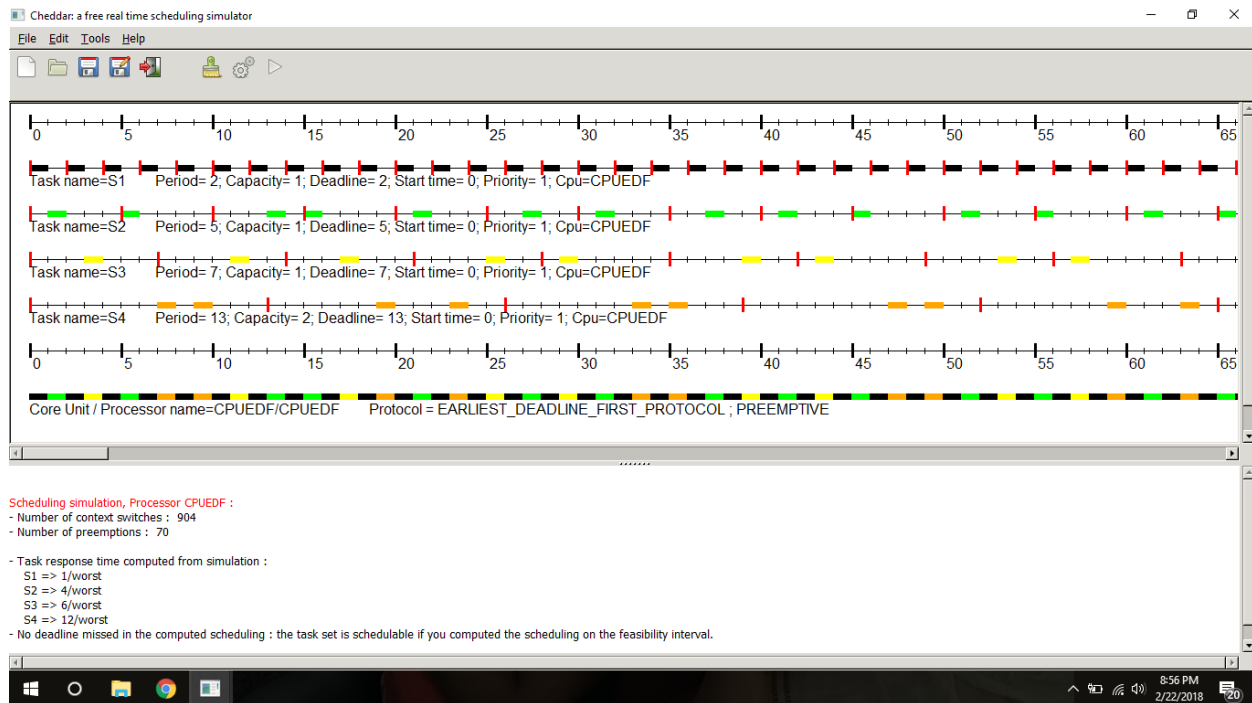
1) Feasibility test based on the processor utilization factor :

- The hyperperiod is 10 (see [18], page 5).
- 0 units of time are unused in the hyperperiod.
- Processor utilization factor with deadline is 1.00000 (see [1], page 6).
- Processor utilization factor with period is 1.00000 (see [1], page 6).
- In the preemptive case, with EDF, the task set is schedulable because the processor utilization factor 1.00000 is equal or less than 1.00000 (see [1], page 8, theorem 2).

2) Feasibility test based on worst case response time for periodic tasks :

- Worst Case task response time :
 - S1 => 2
 - S2 => 5
 - S3 => 10
- All task deadlines will be met : the task set is schedulable.

Example 6 – Cheddar



Example 6 – Feasibility

Scheduling feasibility, Processor CPUEDF :

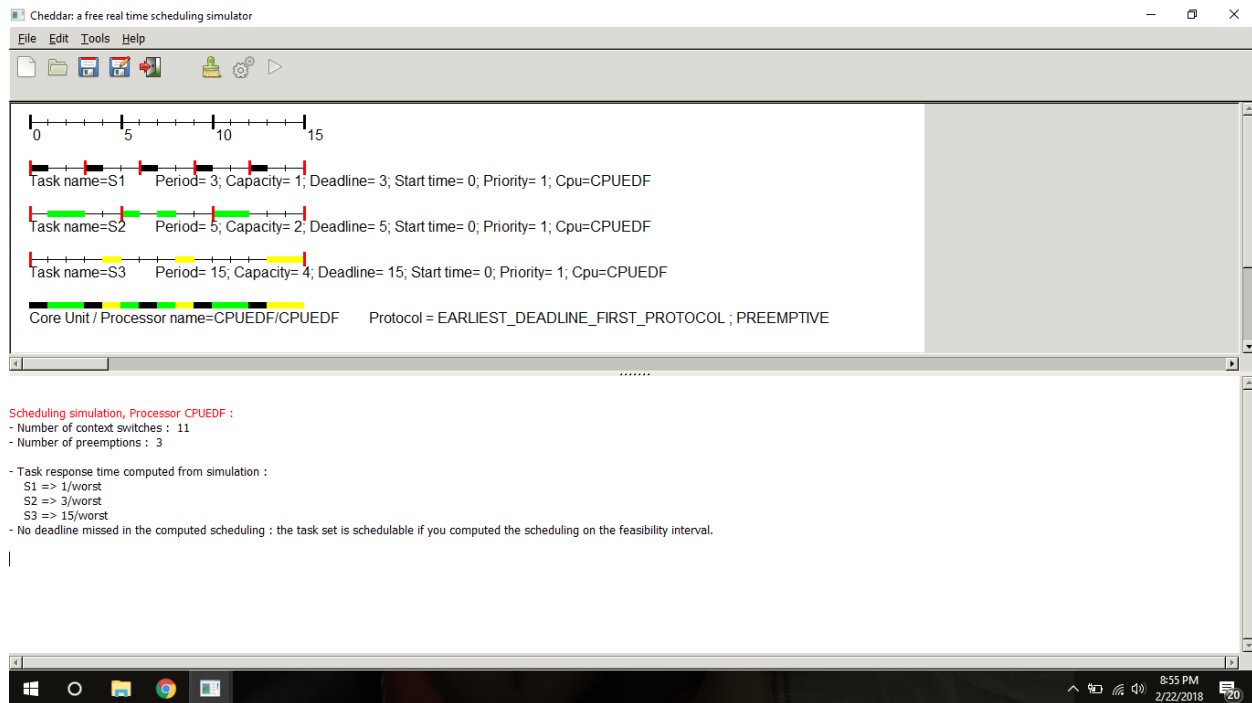
1) Feasibility test based on the processor utilization factor :

- The hyperperiod is 910 (see [18], page 5).
- 3 units of time are unused in the hyperperiod.
- Processor utilization factor with deadline is 0.99670 (see [1], page 6).
- Processor utilization factor with period is 0.99670 (see [1], page 6).
- In the preemptive case, with EDF, the task set is schedulable because the processor utilization factor 0.99670 is equal or less than 1.00000 (see [1], page 8, theorem 2).

2) Feasibility test based on worst case response time for periodic tasks :

- Worst Case task response time :
 - S1 => 1
 - S2 => 4
 - S3 => 6
 - S4 => 12
- All task deadlines will be met : the task set is schedulable.

Example 7 – Cheddar



Example 7 – Feasibility

Scheduling feasibility, Processor CPUEDF :

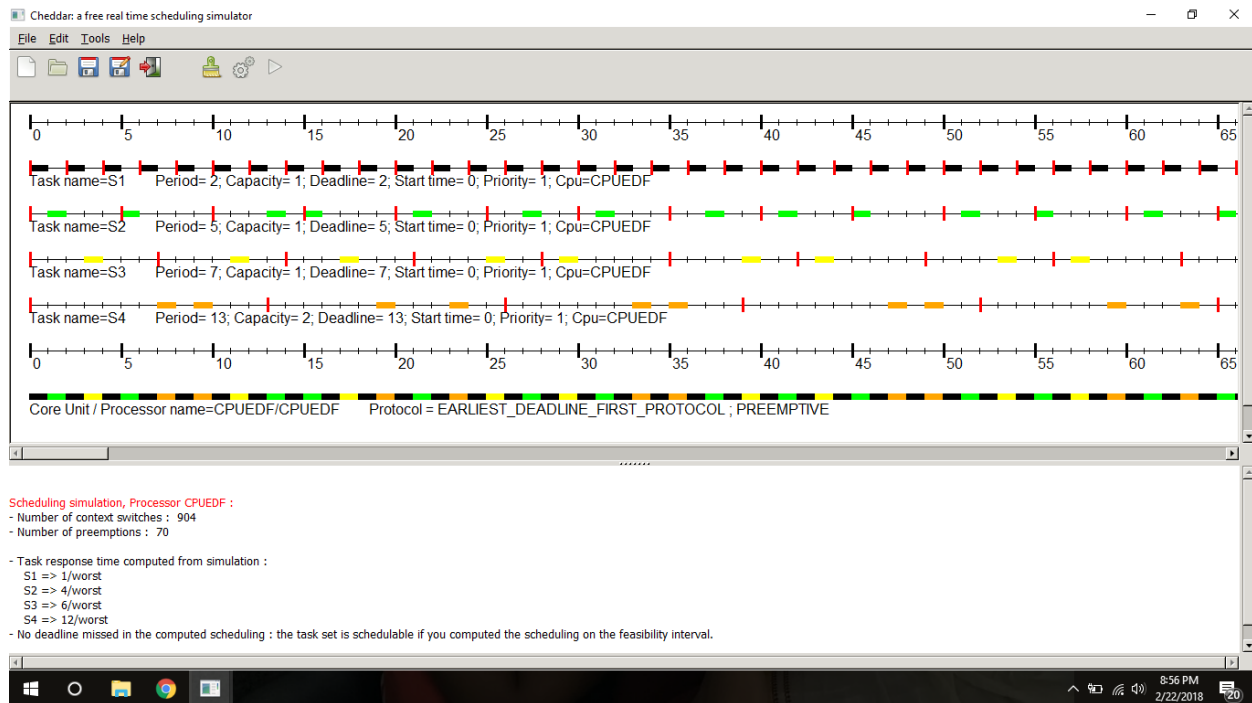
1) Feasibility test based on the processor utilization factor :

- The hyperperiod is 15 (see [18], page 5).
- 0 units of time are unused in the hyperperiod.
- Processor utilization factor with deadline is 1.00000 (see [1], page 6).
- Processor utilization factor with period is 1.00000 (see [1], page 6).
- In the preemptive case, with EDF, the task set is schedulable because the processor utilization factor 1.00000 is equal or less than 1.00000 (see [1], page 8, theorem 2).

2) Feasibility test based on worst case response time for periodic tasks :

- Worst Case task response time :
 - S1 => 3
 - S2 => 5
 - S3 => 15
- All task deadlines will be met : the task set is schedulable.

Example 8 – Cheddar



Example 8 – Feasibility

Scheduling feasibility, Processor CPUEDF :

1) Feasibility test based on the processor utilization factor :

- The hyperperiod is 910 (see [18], page 5).
- 3 units of time are unused in the hyperperiod.
- Processor utilization factor with deadline is 0.99670 (see [1], page 6).
- Processor utilization factor with period is 0.99670 (see [1], page 6).
- In the preemptive case, with EDF, the task set is schedulable because the processor utilization factor 0.99670 is equal or less than 1.00000 (see [1], page 8, theorem 2).

2) Feasibility test based on worst case response time for periodic tasks :

- Worst Case task response time :
 - S1 => 1
 - S2 => 4
 - S3 => 6
 - S4 => 12
- All task deadlines will be met : the task set is schedulable.

LLF

The screenshot of execution for Examples 0 to 8 for **Least Laxity First Scheduling** is as shown below:

Example 0

```
sowmya@sowmya-HP-Pavilion-Notebook:~/Desktop/Feasibility_code_EDF_LLFS$ ./feasibility_tests
Enter 1 for EDF and 2 for LLF: 2
*** LLF Schedule ***
Ex-0
Hyperperiod = 30          U = 0.733
FEASIBLE
sowmya@sowmya-HP-Pavilion-Notebook:~/Desktop/Feasibility_code_EDF_LLFS$
```

Example 1

```
sowmya@sowmya-HP-Pavilion-Notebook:~/Desktop/Feasibility_code_EDF_LLFS$ ./feasibility_tests
Enter 1 for EDF and 2 for LLF: 2
*** LLF Schedule ***
Ex-1
Hyperperiod = 70          U = 0.986
FEASIBLE
sowmya@sowmya-HP-Pavilion-Notebook:~/Desktop/Feasibility_code_EDF_LLFS$
```

Example 2

```
sowmya@sowmya-HP-Pavilion-Notebook:~/Desktop/Feasibility_code_EDF_LLFS$ ./feasibility_tests
Enter 1 for EDF and 2 for LLF: 2
*** LLF Schedule ***
Ex-2
Hyperperiod = 910          U = 0.997
FEASIBLE
sowmya@sowmya-HP-Pavilion-Notebook:~/Desktop/Feasibility_code_EDF_LLFS$
```

Example 3

```
sowmya@sowmya-HP-Pavilion-Notebook:~/Desktop/Feasibility_code_EDF_LLFS$ ./feasibility_tests
Enter 1 for EDF and 2 for LLF: 2
*** LLF Schedule ***
Ex-3
Hyperperiod = 15          U = 0.933
FEASIBLE
sowmya@sowmya-HP-Pavilion-Notebook:~/Desktop/Feasibility_code_EDF_LLFS$
```

Example 4

```
sowmya@sowmya-HP-Pavilion-Notebook:~/Desktop/Feasibility_code_EDF_LLF$ ./feasibility_tests
Enter 1 for EDF and 2 for LLF: 2
*** LLF Schedule ***
Ex-4
Hyperperiod = 16          U = 1.000

FEASIBLE
sowmya@sowmya-HP-Pavilion-Notebook:~/Desktop/Feasibility_code_EDF_LLF$
```

Example 5

```
sowmya@sowmya-HP-Pavilion-Notebook:~/Desktop/Feasibility_code_EDF_LLF$ ./feasibility_tests
Enter 1 for EDF and 2 for LLF: 2
*** LLF Schedule ***
Ex-5
Hyperperiod = 10          U = 1.000

FEASIBLE
sowmya@sowmya-HP-Pavilion-Notebook:~/Desktop/Feasibility_code_EDF_LLF$
```

Example 6

```
sowmya@sowmya-HP-Pavilion-Notebook:~/Desktop/Feasibility_code_EDF_LLF$ ./feasibility_tests
Enter 1 for EDF and 2 for LLF: 2
*** LLF Schedule ***
Ex-6
Hyperperiod = 910         U = 0.997

FEASIBLE
sowmya@sowmya-HP-Pavilion-Notebook:~/Desktop/Feasibility_code_EDF_LLF$
```

Example 7

```
sowmya@sowmya-HP-Pavilion-Notebook:~/Desktop/Feasibility_code_EDF_LLF$ ./feasibility_tests
Enter 1 for EDF and 2 for LLF: 2
*** LLF Schedule ***
Ex-7
Hyperperiod = 15          U = 1.000

FEASIBLE
sowmya@sowmya-HP-Pavilion-Notebook:~/Desktop/Feasibility_code_EDF_LLF$
```

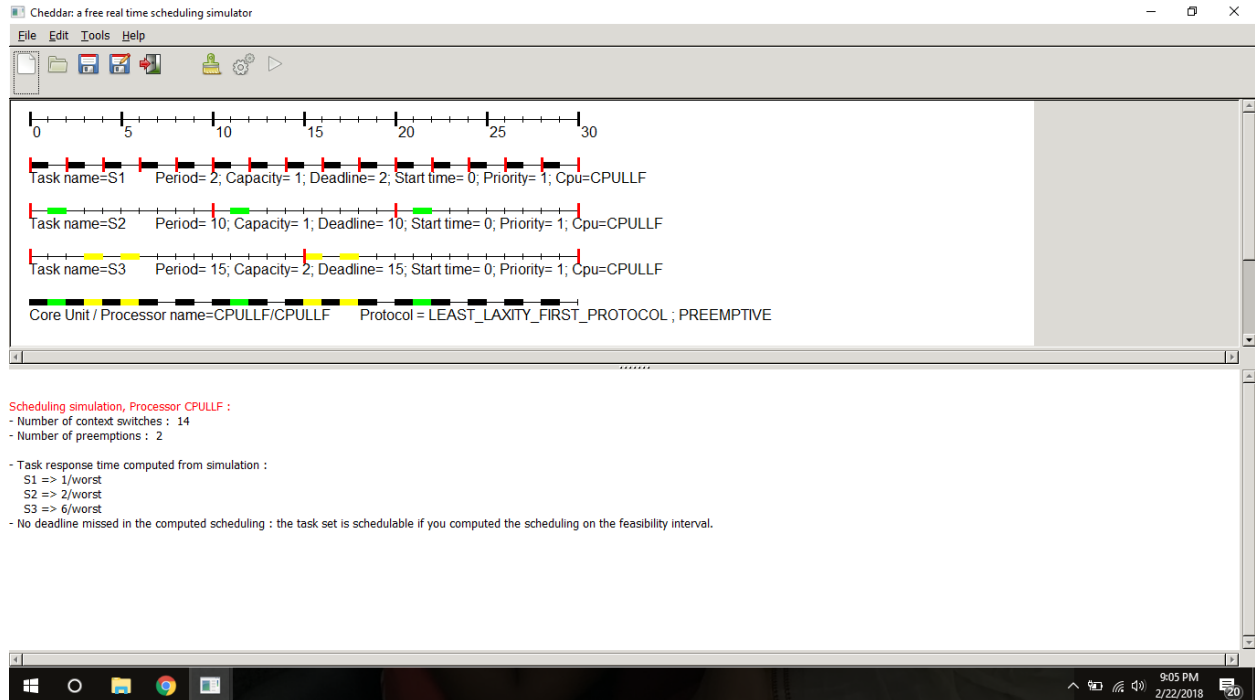
Example 8

```
sowmya@sowmya-HP-Pavilion-Notebook:~/Desktop/Feasibility_code_EDF_LLF$ ./feasibility_tests
Enter 1 for EDF and 2 for LLF: 2
*** LLF Schedule ***
Ex-8
Hyperperiod = 910      U = 0.997

FEASIBLE
sowmya@sowmya-HP-Pavilion-Notebook:~/Desktop/Feasibility_code_EDF_LLF$
```

The screenshots of Cheddar implementation for **Least Laxity First Scheduling** of Examples 0 to 8 are as shown below.

Example 0 – Cheddar



Example 0 – Feasibility

Scheduling feasibility, Processor CPULLF :

1) Feasibility test based on the processor utilization factor :

- The hyperperiod is 30 (see [18], page 5).
- 8 units of time are unused in the hyperperiod.
- Processor utilization factor with deadline is 0.73333 (see [1], page 6).
- Processor utilization factor with period is 0.73333 (see [1], page 6).
- In the preemptive case, with LLF, the task set is schedulable because the processor utilization factor 0.73333 is equal or less than 1.00000 (see [7]).

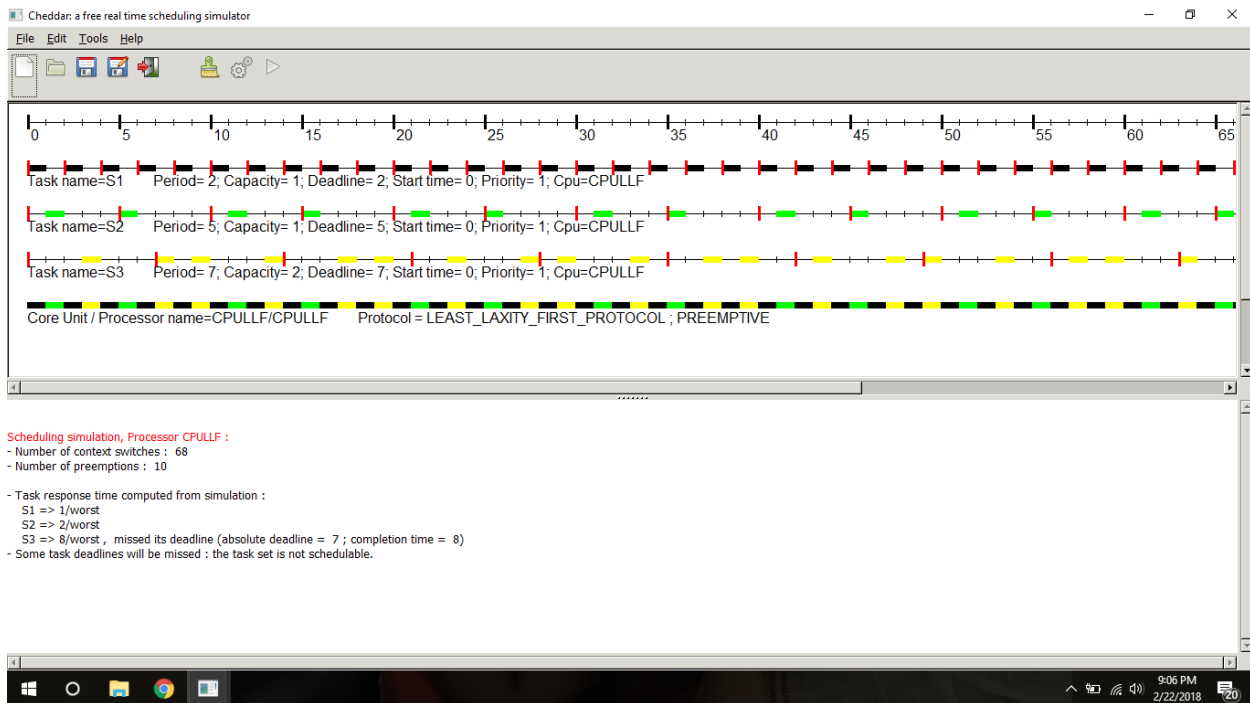
2) Feasibility test based on worst case response time for periodic tasks :

- Worst Case task response time :

- S1 => 1
- S2 => 8
- S3 => 13

- All task deadlines will be met : the task set is schedulable.

Example 1 – Cheddar



Example 1 – Feasibility

Scheduling feasibility, Processor CPULLF :

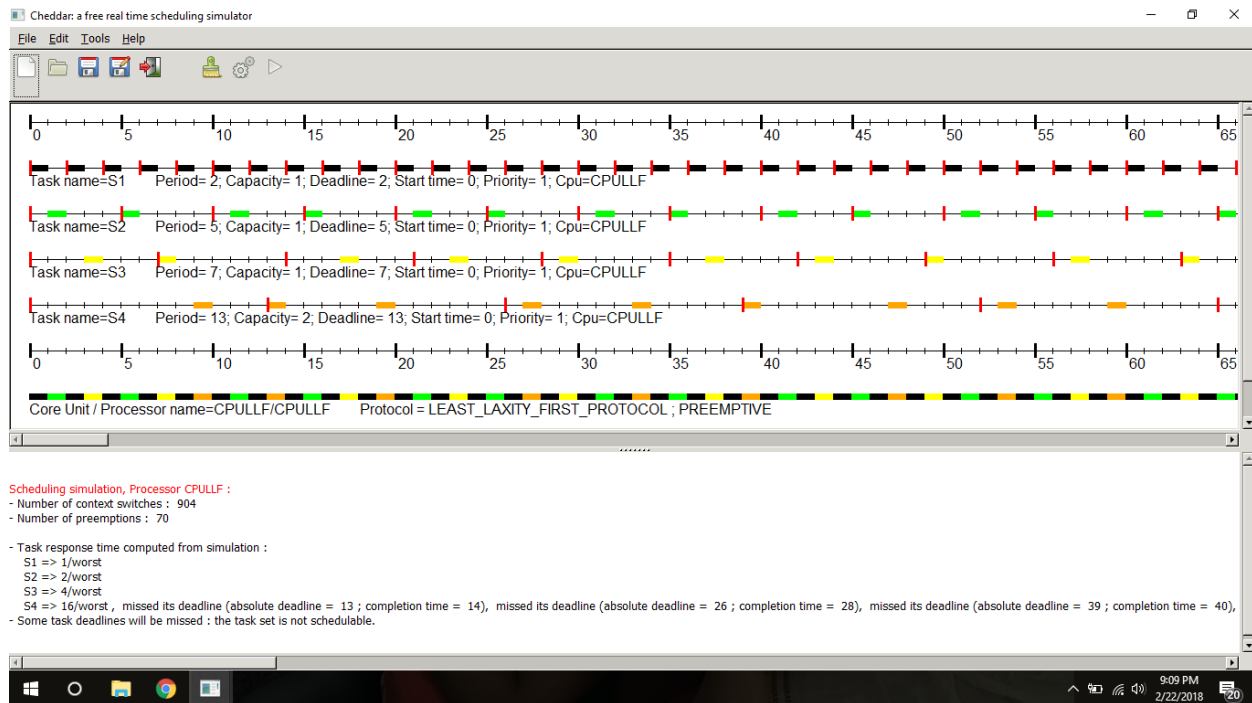
1) Feasibility test based on the processor utilization factor :

- The hyperperiod is 70 (see [18], page 5).
- 1 units of time are unused in the hyperperiod.
- Processor utilization factor with deadline is 0.98571 (see [1], page 6).
- Processor utilization factor with period is 0.98571 (see [1], page 6).
- In the preemptive case, with LLF, the task set is schedulable because the processor utilization factor 0.98571 is equal or less than 1.00000 (see [7]).

2) Feasibility test based on worst case response time for periodic tasks :

- Worst Case task response time :
 - S1 => 1
 - S2 => 4
 - S3 => 6
- All task deadlines will be met : the task set is schedulable.

Example 2 – Cheddar



Example 2 – Feasibility

Scheduling feasibility, Processor CPULLF :

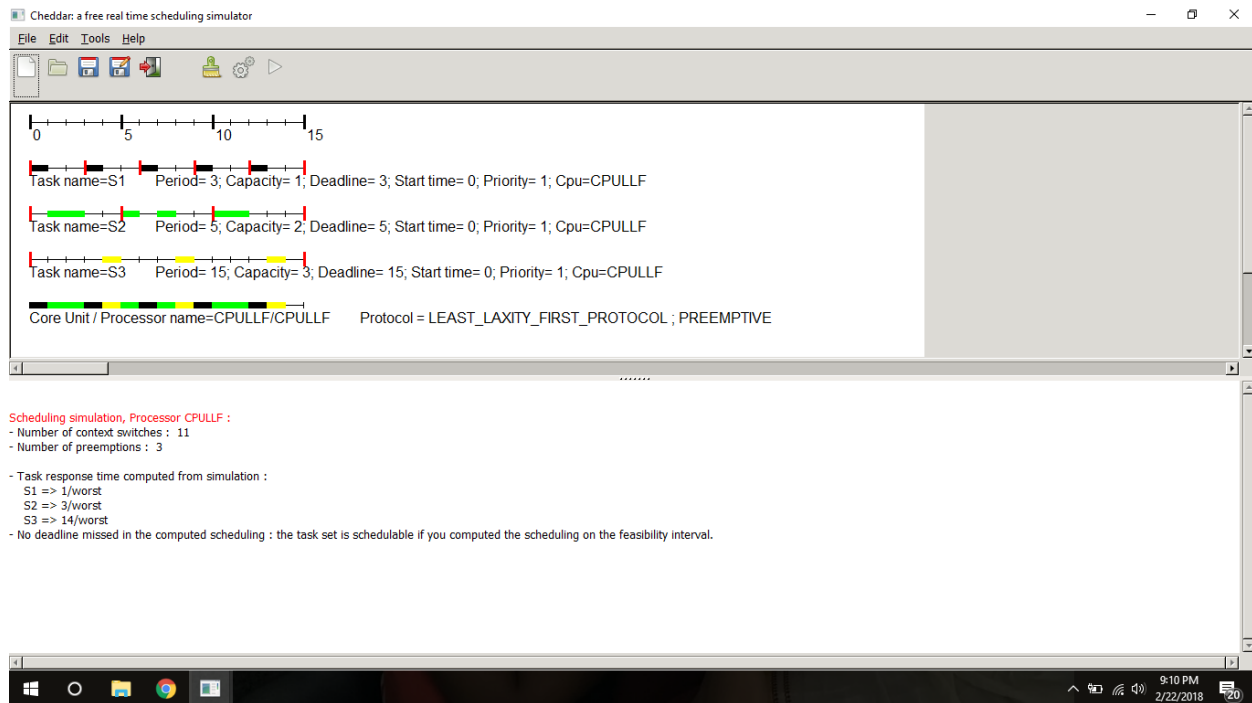
1) Feasibility test based on the processor utilization factor :

- The hyperperiod is 910 (see [18], page 5).
- 3 units of time are unused in the hyperperiod.
- Processor utilization factor with deadline is 0.99670 (see [1], page 6).
- Processor utilization factor with period is 0.99670 (see [1], page 6).
- In the preemptive case, with LLF, the task set is schedulable because the processor utilization factor 0.99670 is equal or less than 1.00000 (see [7]).

2) Feasibility test based on worst case response time for periodic tasks :

- Worst Case task response time :
 - S1 => 1
 - S2 => 4
 - S3 => 6
 - S4 => 12
- All task deadlines will be met : the task set is schedulable.

Example 3 – Cheddar



Example 3 – Feasibility

Scheduling feasibility, Processor CPULLF :

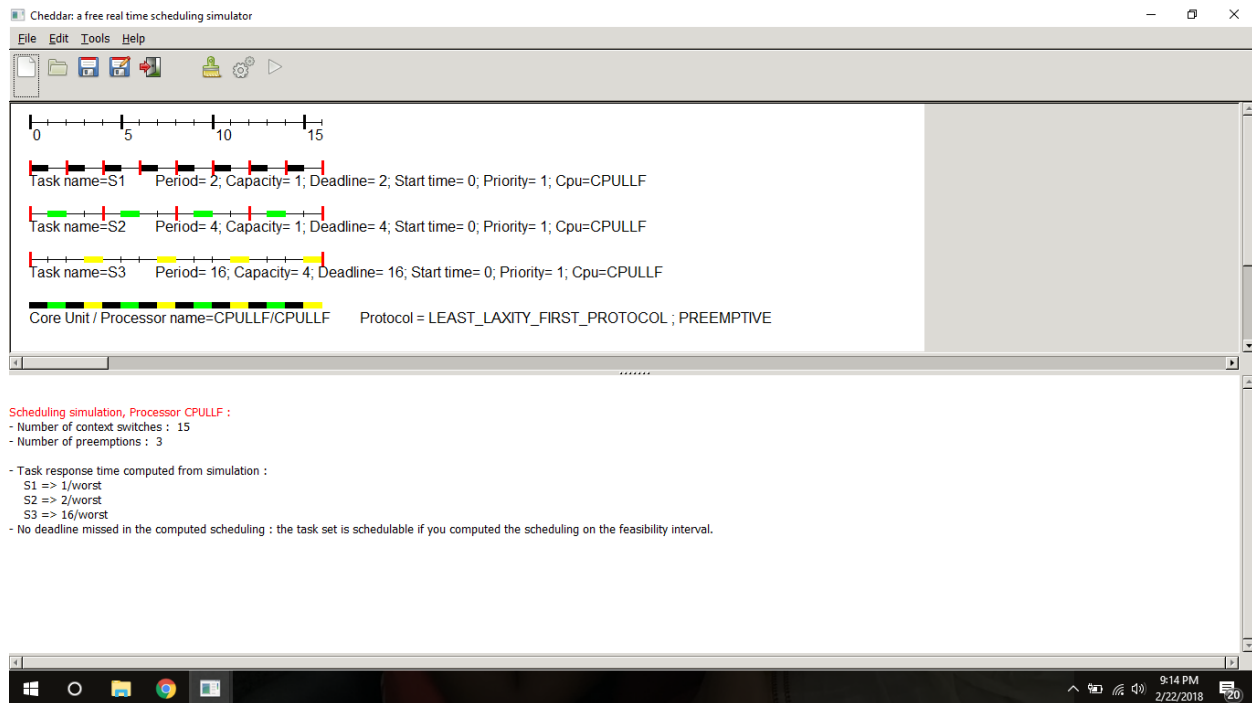
1) Feasibility test based on the processor utilization factor :

- The hyperperiod is 15 (see [18], page 5).
- 1 units of time are unused in the hyperperiod.
- Processor utilization factor with deadline is 0.93333 (see [1], page 6).
- Processor utilization factor with period is 0.93333 (see [1], page 6).
- In the preemptive case, with LLF, the task set is schedulable because the processor utilization factor 0.93333 is equal or less than 1.00000 (see [7]).

2) Feasibility test based on worst case response time for periodic tasks :

- Worst Case task response time :
 - S1 => 2
 - S2 => 4
 - S3 => 14
- All task deadlines will be met : the task set is schedulable.

Example 4 – Cheddar



Example 4 – Feasibility

Scheduling feasibility, Processor CPULLF :

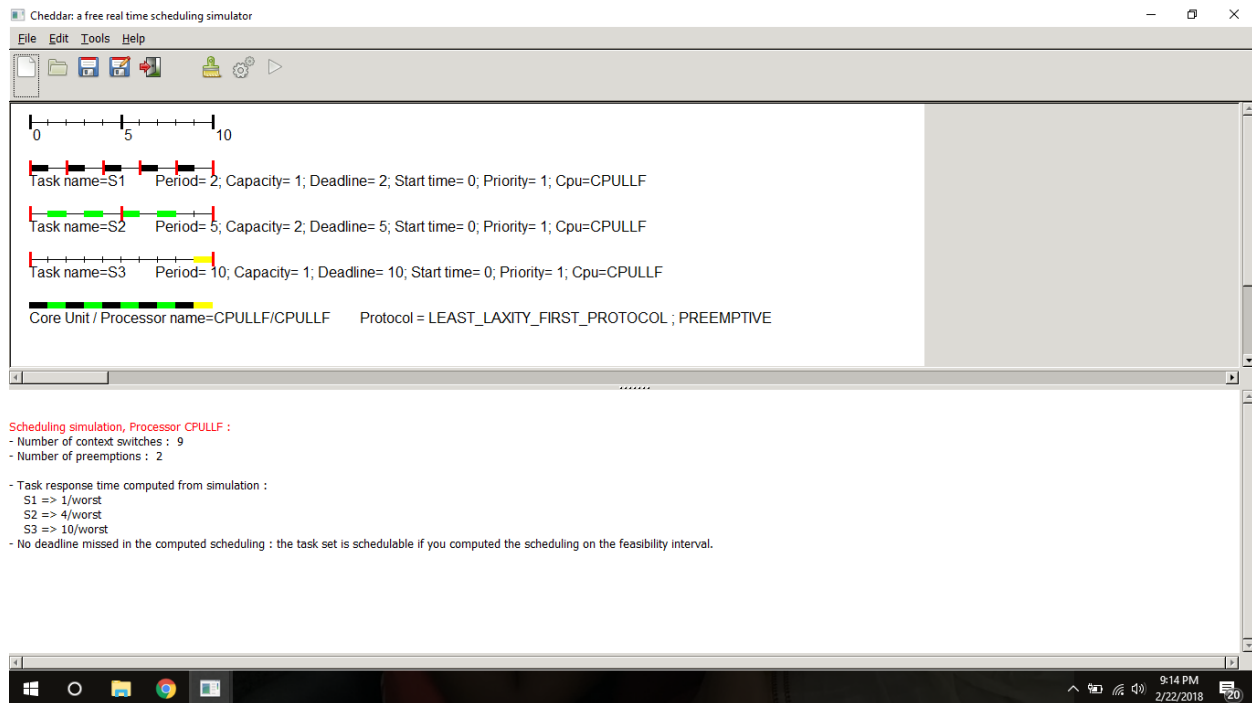
1) Feasibility test based on the processor utilization factor :

- The hyperperiod is 16 (see [18], page 5).
- 0 units of time are unused in the hyperperiod.
- Processor utilization factor with deadline is 1.00000 (see [1], page 6).
- Processor utilization factor with period is 1.00000 (see [1], page 6).
- In the preemptive case, with LLF, the task set is schedulable because the processor utilization factor 1.00000 is equal or less than 1.00000 (see [7]).

2) Feasibility test based on worst case response time for periodic tasks :

- Worst Case task response time :
 - S1 => 2
 - S2 => 4
 - S3 => 16
- All task deadlines will be met : the task set is schedulable.

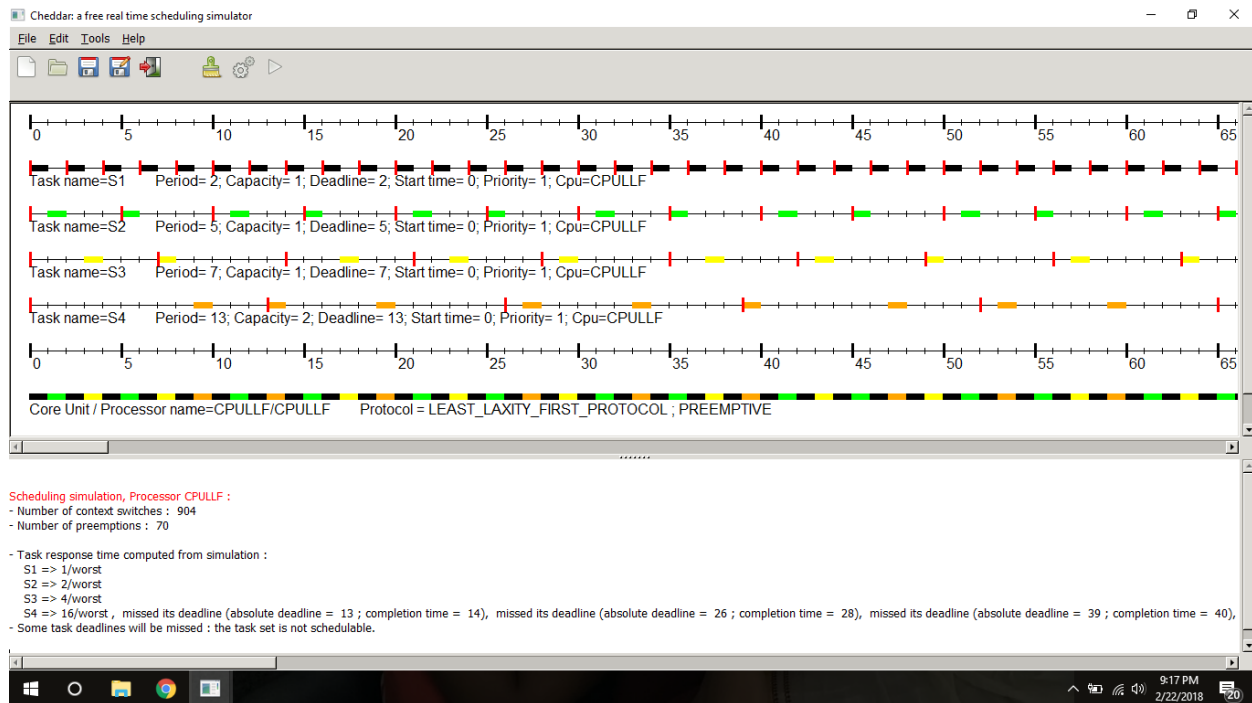
Example 5 – Cheddar



Example 5 – Feasibility

- Scheduling feasibility, Processor CPULLF :
- 1) Feasibility test based on the processor utilization factor :
 - The hyperperiod is 10 (see [18], page 5).
 - 0 units of time are unused in the hyperperiod.
 - Processor utilization factor with deadline is 1.00000 (see [1], page 6).
 - Processor utilization factor with period is 1.00000 (see [1], page 6).
 - In the preemptive case, with LLF, the task set is schedulable because the processor utilization factor 1.00000 is equal or less than 1.00000 (see [7]).
 - 2) Feasibility test based on worst case response time for periodic tasks :
 - Worst Case task response time :
 - S1 => 2
 - S2 => 5
 - S3 => 10
 - All task deadlines will be met : the task set is schedulable.

Example 6 – Cheddar



Example 6 – Feasibility

Scheduling feasibility, Processor CPULLF :

1) Feasibility test based on the processor utilization factor :

- The hyperperiod is 910 (see [18], page 5).
- 3 units of time are unused in the hyperperiod.
- Processor utilization factor with deadline is 0.99670 (see [1], page 6).
- Processor utilization factor with period is 0.99670 (see [1], page 6).
- In the preemptive case, with LLF, the task set is schedulable because the processor utilization factor 0.99670 is equal or less than 1.00000 (see [7]).

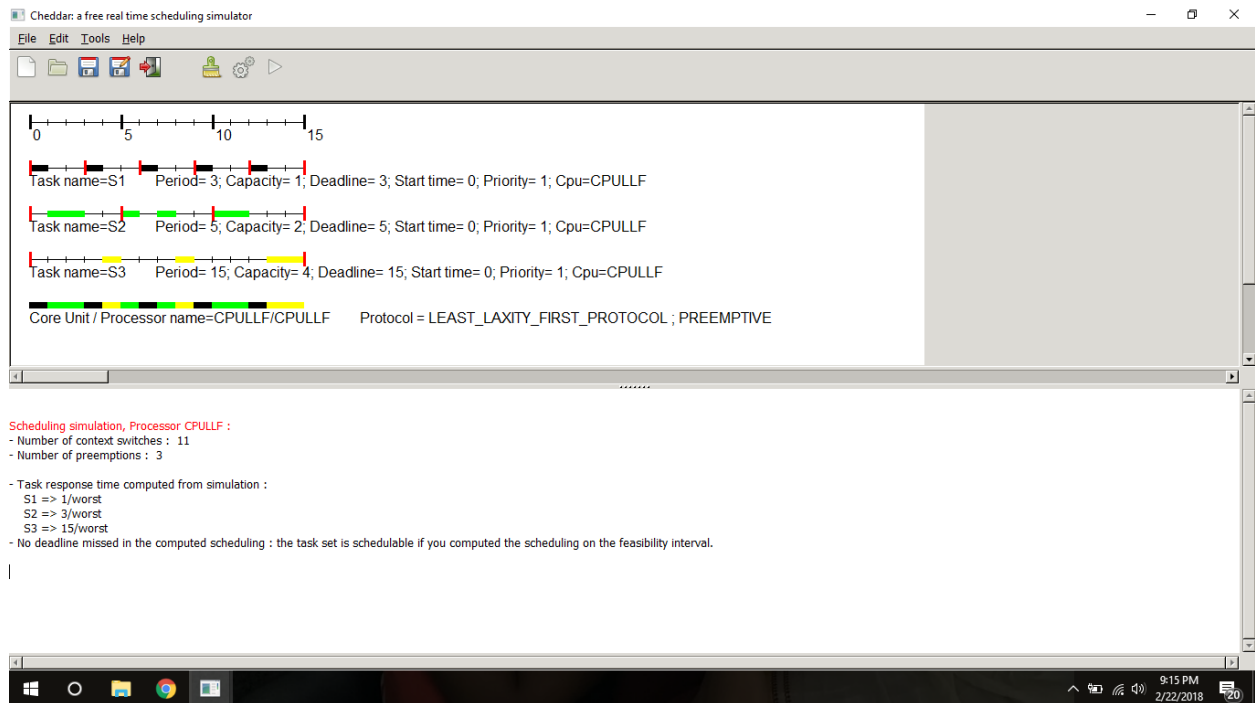
2) Feasibility test based on worst case response time for periodic tasks :

- Worst Case task response time :

- S1 => 1
- S2 => 4
- S3 => 6
- S4 => 12

- All task deadlines will be met : the task set is schedulable.

Example 7 – Cheddar



Example 7 – Feasibility

Scheduling feasibility, Processor CPULLF :

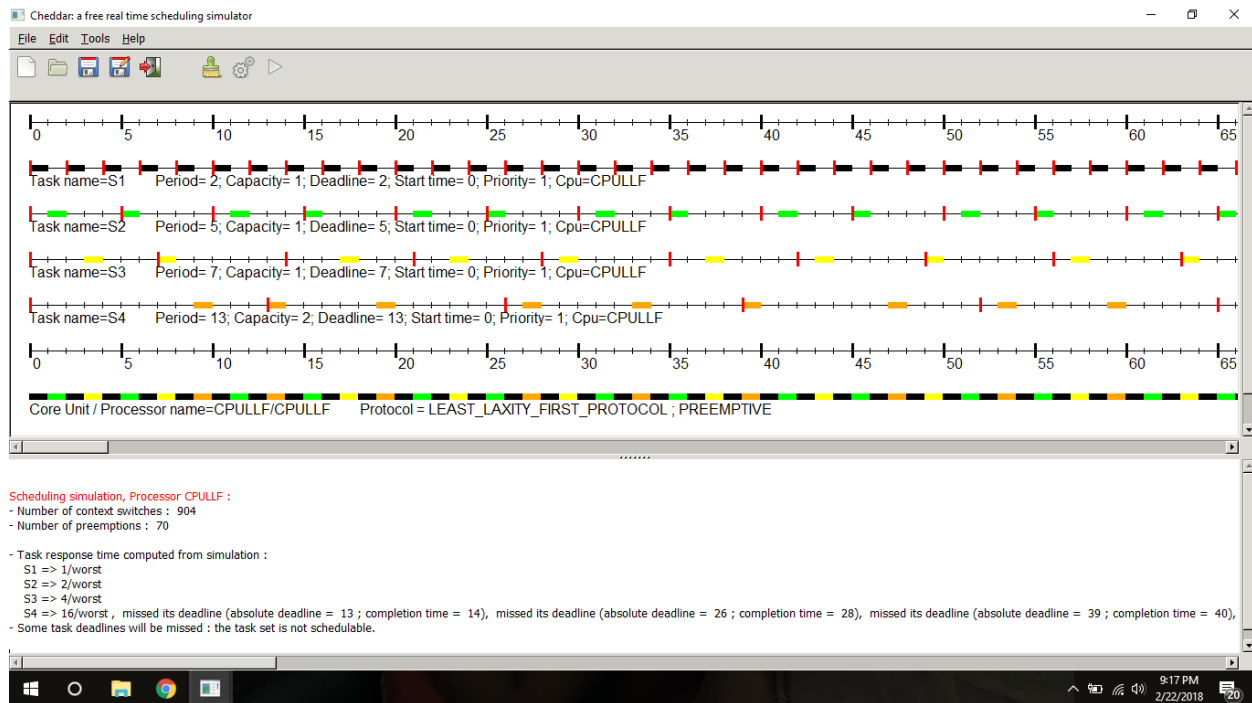
1) Feasibility test based on the processor utilization factor :

- The hyperperiod is 15 (see [18], page 5).
- 0 units of time are unused in the hyperperiod.
- Processor utilization factor with deadline is 1.00000 (see [1], page 6).
- Processor utilization factor with period is 1.00000 (see [1], page 6).
- In the preemptive case, with LLF, the task set is schedulable because the processor utilization factor 1.00000 is equal or less than 1.00000 (see [7]).

2) Feasibility test based on worst case response time for periodic tasks :

- Worst Case task response time :
 - S1 => 3
 - S2 => 5
 - S3 => 15
- All task deadlines will be met : the task set is schedulable.

Example 8 – Cheddar



Example 8 – Feasibility

Scheduling feasibility, Processor CPULLF :

1) Feasibility test based on the processor utilization factor :

- The hyperperiod is 910 (see [18], page 5).
- 3 units of time are unused in the hyperperiod.
- Processor utilization factor with deadline is 0.99670 (see [1], page 6).
- Processor utilization factor with period is 0.99670 (see [1], page 6).
- In the preemptive case, with LLF, the task set is schedulable because the processor utilization factor 0.99670 is equal or less than 1.00000 (see [7]).

2) Feasibility test based on worst case response time for periodic tasks :

- Worst Case task response time :
 - S1 => 1
 - S2 => 4
 - S3 => 6
 - S4 => 12
- All task deadlines will be met : the task set is schedulable.

DM

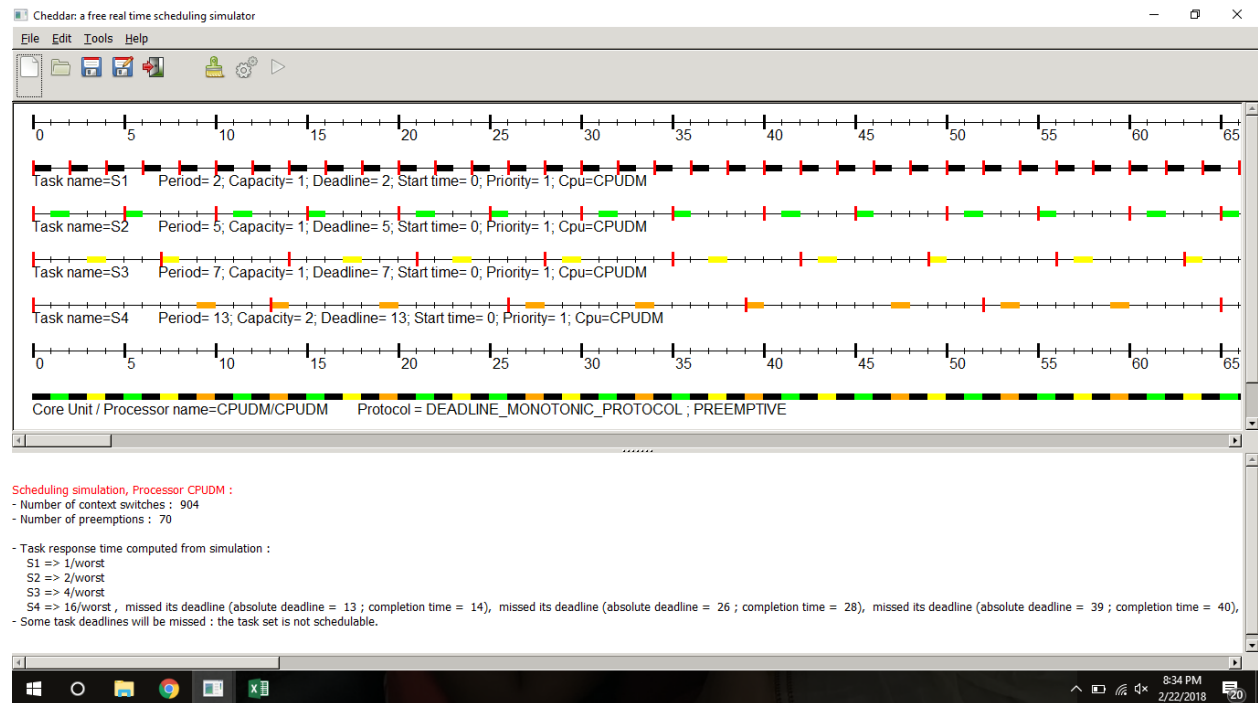
Deadline-monotonic priority assignment is a priority assignment policy used with fixed priority pre-emptive scheduling. With **deadline-monotonic** priority assignment, tasks are assigned priorities according to their **deadlines**; the task with the shortest **deadline** being assigned the highest priority.

The screenshot of execution for Examples 0 to 8 for **Deadline Monotonic Scheduling** is as shown below:

```
***** DM Scheduler *****
Utilization 1.109524
INFEASIBLE
priority[0]:0
priority[1]:1
priority[2]:2
****Cycle 0*****Service1***
****Cycle 1*****Service 2***
****Cycle 2*****Service1***
****Cycle 3*****Service 3***
****Cycle 4*****Service1***
****Cycle 5*****Service 2***
****Cycle 6*****Service1***
****Cycle 7*****Service 3***
****Cycle 8*****Service1***
****Cycle 9*****Service 4***
****Cycle 10*****Service1***
****Cycle 11*****Service 2***
****Cycle 12*****Service1***
****Cycle 13*****Service 4***
****Cycle 14*****Service1***
****Cycle 15*****Service 2***
****Cycle 16*****Service1***
****Cycle 17*****Service 3***
****Cycle 18*****Service1***
****Cycle 19*****Service 4***
****Cycle 20*****Service1***
****Cycle 21*****Service 2***
****Cycle 22*****Service1***
****Cycle 23*****Service 3***
****Cycle 24*****Service1***
****Cycle 25*****Service 2***
****Cycle 26*****Service1***
****Cycle 27*****Service 4***
****Cycle 28*****Service1***
****Cycle 29*****Service 3***
****Cycle 30*****Service1***
****Cycle 31*****Service 2***
****Cycle 32*****Service1***
****Cycle 33*****Service 4***
****Cycle 34*****Service1***
****Cycle 35*****Service 2***
****Cycle 36*****Service1***
****Cycle 37*****Service 3***
****Cycle 38*****Service1***
****Cycle 39*****Service 4***
****Cycle 40*****Service1***
****Cycle 41*****Service 2***
****Cycle 42*****Service1***
```

The screenshots of Cheddar implementation for **Deadline Monotonic Scheduling** of Example 6 is as shown below.

Example 6 – Cheddar



Example 6 – Feasibility

Scheduling feasibility, Processor CPUDM :

1) Feasibility test based on the processor utilization factor :

- The hyperperiod is 910 (see [18], page 5).
- 3 units of time are unused in the hyperperiod.
- Processor utilization factor with deadline is 0.99670 (see [1], page 6).
- Processor utilization factor with period is 0.99670 (see [1], page 6).
- In the preemptive case, with DM, the task set is not schedulable because the processor utilization factor 0.99670 is more than 0.75683 (see [7]).

2) Feasibility test based on worst case response time for periodic tasks :

- Worst Case task response time : (see [2], page 3, equation 4).
 - S4 => 16, missed its deadline (deadline = 13)
 - S3 => 4
 - S2 => 2
 - S1 => 1
- Some task deadlines will be missed : the task set is not schedulable.

Examples 1,2,6 and 8 are NOT FEASIBLE according to the RATE MONOTIC scheduling. The same examples are FEASIBLE with EDF and LLF.

Explanation for failure of RM for examples 1, 2, 6 and 8 (examples 6 and 8 are same as example 2) while feasibility using EDF, LLF:

- In example 1, service 3 missed one deadline. In example 2, service 4 missed several deadlines. These services are the lowest priority services in their respective sets. Rate Monotonic policy fundamentally assigns more priority to services which have a higher request rate. In the process, services having a very low request rate might not be processed and may miss deadlines.
- The amount of safe margin that has to be maintained to guarantee meeting of deadlines is high. This is because RM policy's bound on CPU utilization is pessimistic and services that have higher utilization factors might fail.
- In case of example 1, the CPU utilization is around 0.98 which is way too high compared to a sure feasibility bound of 0.78. In case of example 2, the utilization factor is 0.996 whereas the RM least upper bound is 0.756. Any utilization below these least upper bounds ensures feasibility.
- CPU utilization over 1 guarantees failure. Anything in the middle must be simulated. This is because although the utilization factor is less than 1, it only ensures that till the LCM, all requested tasks will be serviced, but it does not guarantee that intermediate deadlines will be met.
- Sometimes when utilization factor is close to 1, due to context switches and pre-emption, there might be significant time lost in overhead and hence the required utilization goes above 1, hence making tasks infeasible.
- On the other hand, EDF and LLF are dynamic priority services wherein higher priorities are assigned to tasks having shorter deadlines. This means that even for tasks that run infrequently, they are bound to run when their deadlines come closer. Also, bound for sure feasibility is 1 which is why the tasks are feasible with EDF and LLF but not with RM.

Outputs from our code match the feasibility test and simulations from Cheddar. This is because Cheddar feasibility tests make use of RM-LUB scheduling point and completion time tests. Since our feasibility test code for RM also implemented the scheduling point and completion time tests, the outputs were bound to match. For EDF and LLF, we compare our results from cheddar with timing diagrams provided as excel sheets for each example. We find that timing diagram drawn manually are same as those obtained from cheddar and hence the feasibilities were verified. Hence, we can state that our modified Feasibility code agrees with the Cheddar analysis for all cases.

All code, screenshots are attached as a .zip file.

Also, We have tried to emulate LLF, DM schedulers.

Screenshots attached as zip file.

For emulation we have attached 2 screenshots for each example. Screenshots are named as per the example number and part no Ex: LLF_sch0_1

For 8th example screenshot when changed the execution time it misses a deadline and is named as LLF_sch_8_1_missed_deadlines.

5.

Provide 3 constraints that are made on the RM LUB derivation and 3 assumptions as documented in the Liu and Layland paper and in Chapter 3 of the text. Finally, list 3 key derivation steps in the RM LUB derivation that you either do not understand or that you would consider “tricky” math. Attempt to describe the rationale for those steps as best you can do based upon reading in Chapter 3 of the text.

Answer:

Constraints that are made on the least upper bound derivation as documented in the Liu and Layland paper are:

1. The deadline for a given task is equal to its period of completion, which might not be the case always.
2. The threads have fixed priority based on the time period.
3. All tasks are assumed to be preemptive and run till completion.
4. The only resource shared by the tasks is the CPU.
5. Periodicity and Run-time: a period which is equal to the shortest request interval and a run-time equal to the longest run-time. This is a constraint because the assumption may not be always true.

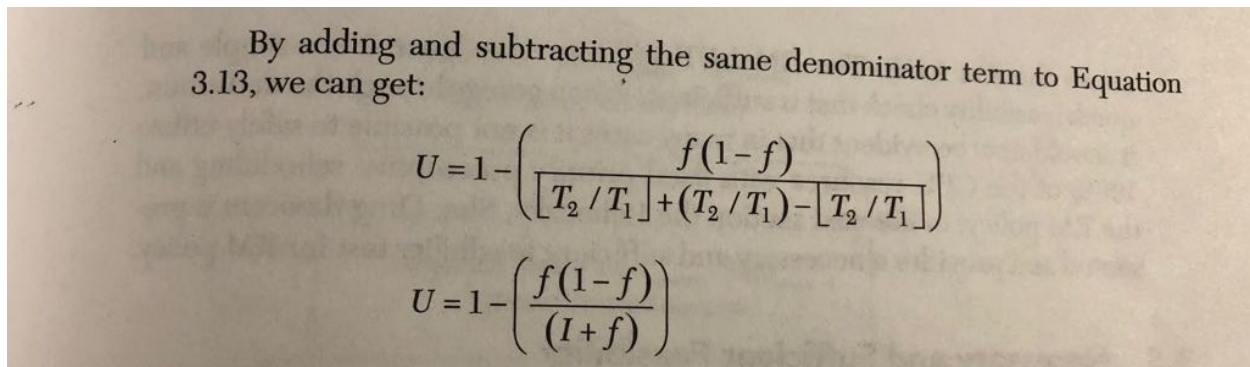
Assumptions of the Liu & Layland Paper are:

1. Requests for all tasks for which hard deadlines exist are periodic with constant interval between requests.
2. The tasks are independent in that requests for a certain task do not depend on the initiation or the completion of requests for other tasks.
3. Deadlines consist of run-ability constraints only. Each task must be completed before the next request for it occurs.
4. Run-time for each task is constant for that task and does not vary with time. Run-time here refers to the time which is taken by a processor to execute the task without interruption.

Key Derivation steps in RM LUB Derivation found to be 'Tricky' and description of rationale:

1.

The step after obtaining Equation 3.13 in Chapter 3 of the textbook looks complex, because of the floor functions. The value for achieving Utilization is tricky because same numerator and denominator gets added and subtracted. Replacing the floor functions with another variable would bring big complications to derive the equations that follow.



By adding and subtracting the same denominator term to Equation 3.13, we can get:

$$U = 1 - \left(\frac{f(1-f)}{\left[\frac{T_2}{T_1} \right] + (T_2/T_1) - \left[\frac{T_2}{T_1} \right]} \right)$$
$$U = 1 - \left(\frac{f(1-f)}{(1+f)} \right)$$

2.

Case 1:

Liu and Layland Paper:

The intermediate steps in the derivation of the equation for utilization factor are ignored/not detailed., and thus proves to be tricky/not easily comprehensible.

Chapter 3:

The steps are detailed and explained in a clear and concise manner. The floor expression (T_2/T_1) is used, which generalizes the overlap of T_1 and T_2 . Further steps simplify the equation and break it down to separate terms whose effect is analyzed. We hence finally achieve the plot depicting how Utilization factor decreases monotonically as C_1 increases.

3.

Case 2:

Liu and Layland Paper:

Similarly, in case 2, the derivation quickly progresses to the utilization factor equation from the equation for largest value of C_2 without providing adequate explanation for each step.

Chapter 3:

Chapter 3 explains this very well and effectively explains how the utilization is derived and why the floor (T_2/T_1) expression is used. The text further goes on to analyze the terms of the utilization

equation using values for T2 and T1. The final plot shows how the utilization factor increases as C1 increases.

4.

Liu and Layland Paper:

In the derivation of Theorem 4, previous results are used in the derivation of expression for utilization factor for m tasks. It is not clear how $C1 = T2 - T1$ is same as $C1'$ and how $C2' = C2 + \Delta$ but others don't follow this.

Chapter 3:

The approach to derive an expression for m tasks is different in the text book.

Now taking the derivative of U w.r.t. f , and solving for the extreme, we get:

$$\partial U / \partial f = \frac{(1+f)(1-2f) - (f-f^2)(1)}{(1+f)^2} = 0$$

Solving for f , we get:

$$f = (2^{1/2} - 1)$$

And, plugging f back into U, we get:

$$U = 2(2^{1/2} - 1)$$

The RM LUB of $m(2^{1/m} - 1)$ is $2(2^{1/2} - 1)$ for $m=2$, which is true for the two-service case—QED.

This is a bit confusing because it skips a few steps. A better explanation would be that U is monotonic and increasing with 'i'. The smallest possible values of 'i' would be 1 and hence the above equation could reduce based on this assumption to the above mentioned equations. Also, a simple comparison of expression for two tasks and showing that $m=2$ and hence coming up with a generic expression is lucid and easy to understand but not very convincing.

REFERENCES

1. ["Architecture of the Space Shuttle Primary Avionics Software System" by Gene Carlow](#)
2. "Building Safety-Critical Real-Time Systems with Reuseable Cyclic Executives", [http://dx.doi.org/10.1016/S0967-0661\(97\)00088-9](http://dx.doi.org/10.1016/S0967-0661(97)00088-9).

3. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment – Liu and Layland
4. Real-Time Embedded Components and Systems – Sam Siewart (Chapter 3)
5. <https://embeddedbhavesh.wordpress.com/category/real-time-systems/>
6. https://en.wikipedia.org/wiki/Earliest_deadline_first_scheduling
7. https://en.wikipedia.org/wiki/Least_slack_time_scheduling
8. <http://beru.univ-brest.fr/~singhoff/cheddar/>
9. https://en.wikipedia.org/wiki/Deadline-monotonic_scheduling
10. The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior - John Lehoczky, Lui Sha and Ye Ding
11. http://www.cs.cmu.edu/~ssaewong/research/audsley_DMS.pdf