Department of Electrical and Computer Engineering

University of Colorado at Boulder

ECEN5623 - Real Time Embedded Systems

Exercise 2

Submitted by

Parth Thakkar — Mrunal Yadav

Submitted on February 24, 2024

Contents

List of Figures 1

List of Tables 2

1 Question 1 4
A . . . 4
B . . . 6

2 Question 2 7
A . . . 7
B . . . 8

3 Question 3 9

4 Question 4 11
A . . . 11
Example 0 . . . 11
Example 1 . . . 12
Example 2 . . . 13
Example 3 . . . 14
Example 4 . . . 16
B . . . 17
Example 5 . . . 17
Example 6 . . . 19
Example 7 . . . 21
Example 8 . . . 24
Example 9 . . . 26
C . . . 28

5 Question 5 30
A . . . 30
B . . . 31

6 Challenges Faced 31

7 Conclusion 31

8 References 32

Appendices 33

A C Code for the Implementation 33

List of Figures

1 SSH connection with jatson nano . . . 4
2 VNC(user creation and showing) . . . 5
3 VNC . . . 6
4 VNC(user creation and showing) . . . 6
5 Frequency executive architecture . . . 7
6 Frequency executive architecture . . . 8
7 Cyclic Executive . . . 9
8 Example 0, RM test . . . 11

9 Example 1, RM analysis . . . 12
10 Example 2, RM analysis . . . 13
11 Example 3, RM analysis . . . 15
12 Example 4, RM analysis . . . 16
13 Example 5, RM analysis . . . 17
14 Example 5, EDF analysis . . . 18
15 Example 5, LLF analysis . . . 18
16 Example 6, RM analysis . . . 20
17 Example 6, Deadline Monotonic analysis . . . 20
18 Example 7, RM analysis . . . 22
19 Example 7, EDF analysis . . . 22
20 Example 7, LLF analysis . . . 23
21 Example 8, EDF analysis . . . 24
22 Example 8, LLF analysis . . . 25
23 Example 9, RM analysis . . . 26
24 Example 9, EDF analysis . . . 27
25 Example 9, LLF analysis . . . 27

List of Tables

1 Comparison Table . . . 10
2 Tasks Parameters . . . 29

PDF is clickable

Objective

1. Understanding the concept of the Cyclic Executive in comparison to Linux POSIX RT threading and RTOS. Implementing and analyzing custom feasibility test code for different scheduling policies (RM, EDF, LLF) using Cheddar.

2. Moreover, understanding the constraints, assumptions, and derivation steps in Rate Monotonic (RM) Least Upper Bound (LUB) as outlined in Chapter 3 of the textbook.

1 Question 1

Q: [5 points] make yourself an account on your Dev Kit.

(a) **Q: enable SSH via the GUI or with headless methods.**

Answer: To enable SSH on the Jetson Nano, either through the graphical user interface (GUI) or headlessly, follow these steps:

i. **Connecting the Device:** First, connect the Jetson Nano to your computer using a USB cable. This establishes a virtual Ethernet connection between your PC and the Nano.

ii. **Finding the IP Address:** Use the ipconfig command on Windows or ifconfig on Linux/Mac to list all network devices and their configurations. Look for an entry related to the Nano, which should show an IP address assigned by DHCP.

iii. **Identifying the Gateway IP:** To find the gateway IP address, use the arp -a command. This command displays the ARP table, which includes the IP addresses of all devices on the network. In this case, you found the gateway IP to be 192.168.55.1.

iv. **SSH into the Nano:** With the gateway IP identified, you can SSH into the Jetson Nano using its IP address and your username. The command format is ssh username@ip_address, so in my case, it would be ssh parthishere@192.168.55.1

v. Screenshot for SSH connection

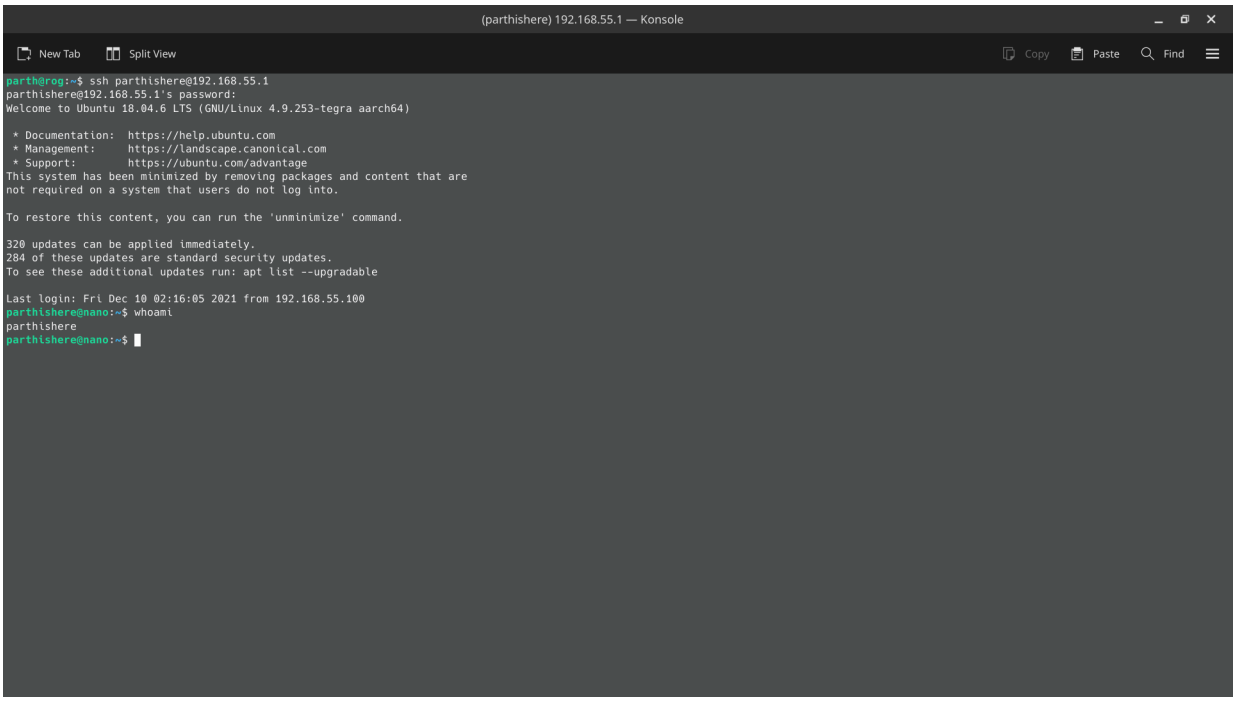


Figure 1: SSH connection with jatson nano

To set up VNC on the Jetson Nano for remote desktop access:

i. Editing Vino Settings: Open the Vino settings file for editing with

```
sudo nano /usr/share/glib-2.0/schemas/org.gnome.Vino.gschema.xml.
```

ii. Modifying the XML File: Insert the following XML key to enable remote desktop access through VNC:

```
<key name='enabled' type='b'>
  <summary>Enable remote access to the desktop</summary>
    <description>
      If true, allows remote access to the desktop via the RFB protocol.
```

```
      Users on remote machines may then connect to the desktop using a
      VNC viewer.
    </description>
  <default>true</default>
</key>
```

iii. Compiling Schemas: After adding the key, compile the schemas with sudo glib-compile-schemas /usr/share/glib-2.0/schemas.

iv. Adjusting Vino Settings: Disable encryption and the prompt for VNC connections by running:

```
gsettings set org.gnome.Vino require-encryption false
gsettings set org.gnome.Vino prompt-enabled false
```

v. Enabling Vino Server: Reload the system daemon and enable the Vino server for VNC access using:

```
systemctl daemon-reload
systemctl enable /usr/lib/vino/vino-server
```

vi. After setting up VNC on the Nano and identifying the IP address via arp -a, you can use a VNC viewer downloaded from the internet to connect to the Nano for remote desktop access.

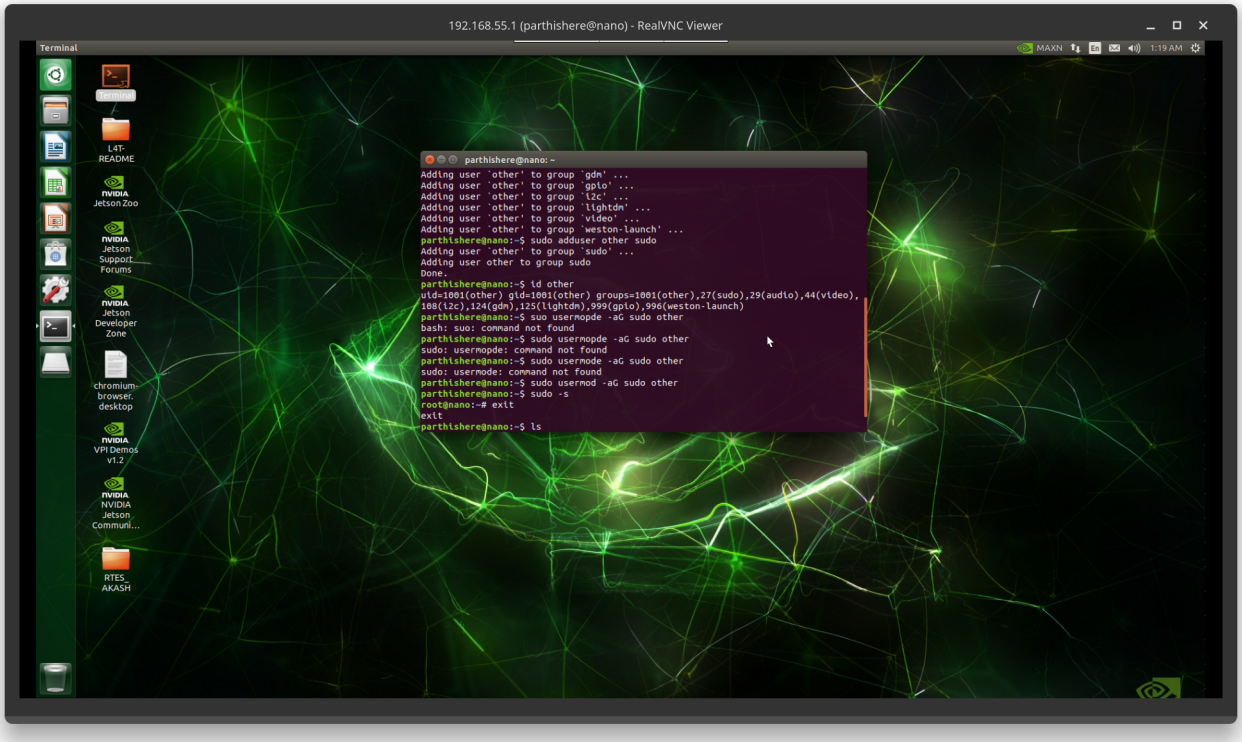


Figure 2: VNC(user creation and showing)

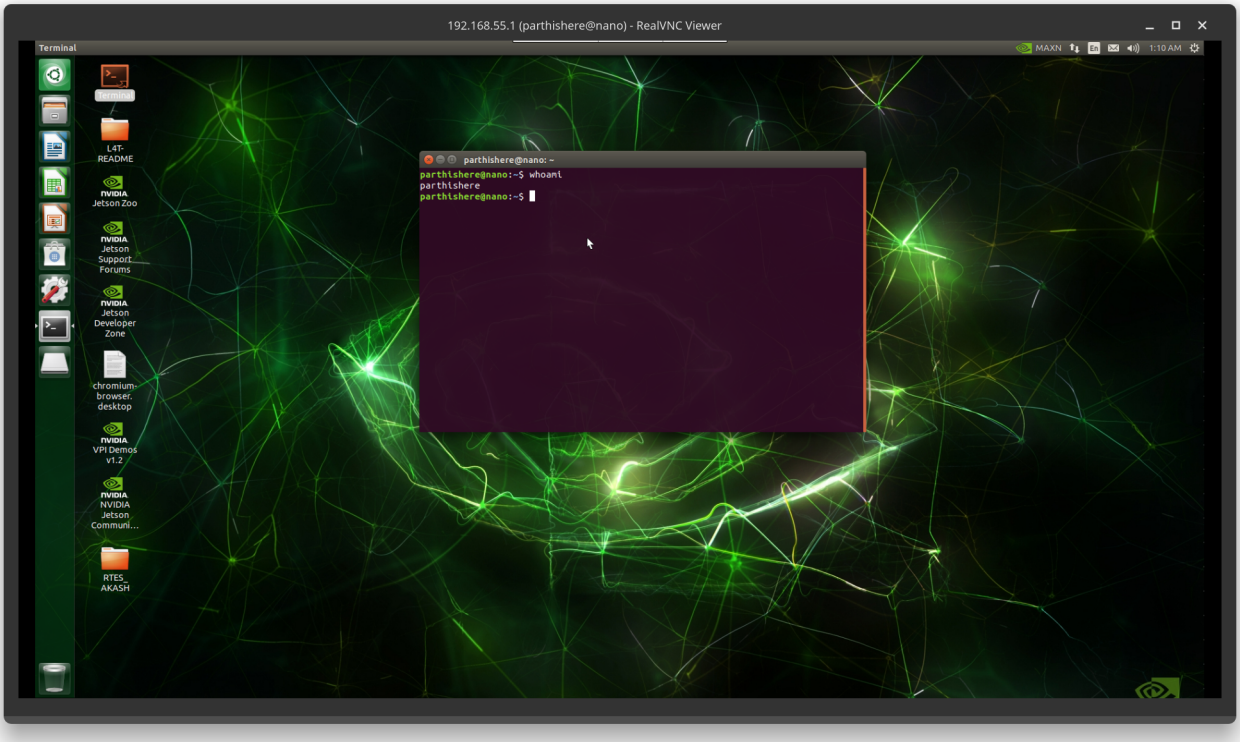


Figure 3: VNC

(b) **Q: Show evidence that you have created a custom login with screenshots or photos from your phone.**

Answer: We can see the title bar of my PC showing VNC viewer for the screen of jatson nano

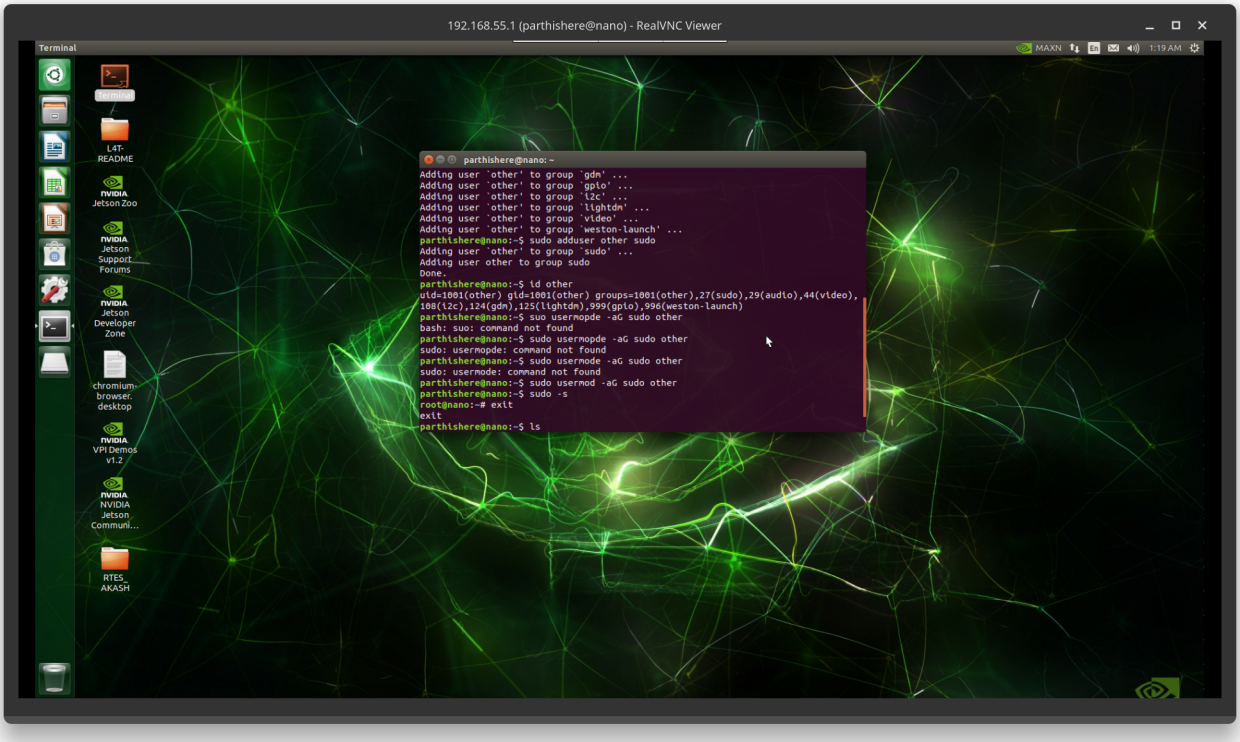


Figure 4: VNC(user creation and showing)

2 Question 2

Q: [10 points] Read the paper "Architecture of the Space Shuttle Primary Avionics Software System", by Gene Carlow.

(a) Q: Provide an explanation and critique of the frequency executive architecture.
Answer:

- The Frequency Executive architecture described in the "Architecture of the Space Shuttle Primary Avionics Software System" by Gene Carlow is central to the Guidance, Navigation, and Control GN&C design structure of the Space Shuttle's Primary Avionics Software System PASS.
- The on-board software is based on three applications.
- One among them is GN&C(Guidance, navigation and control) software.
- It is used to determine vehicle position, velocity and altitude.
- There are two modules in GN&C Design structure - Cyclic and Non-Cyclic Module.

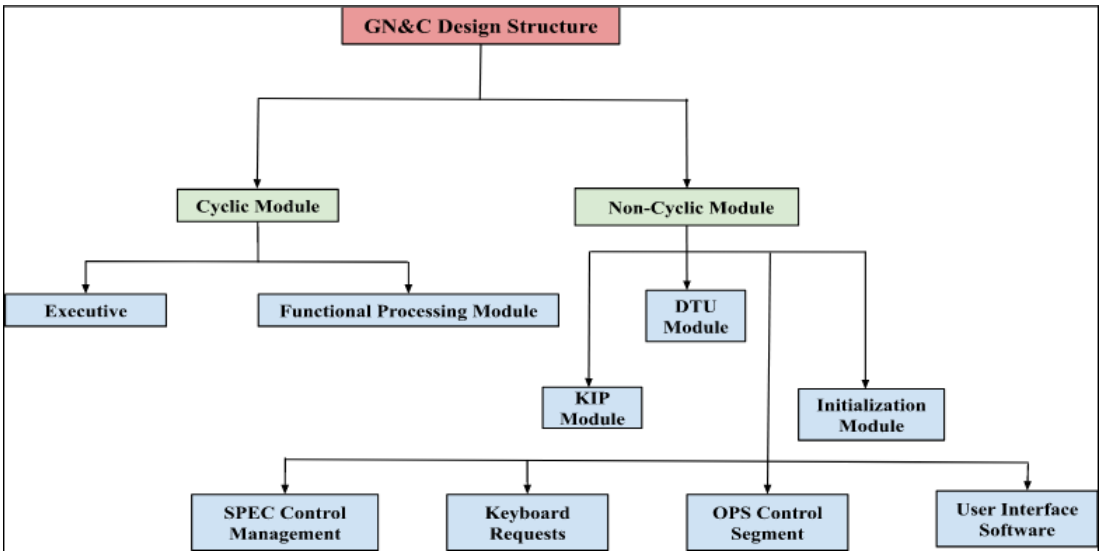


Figure 5: Frequency executive architecture

- OPS Control Segment : The control segment initiates a repeating set of tasks, and the FCOS(Fixed Cycle Offset Scheduling) scheduling method, facilitated by a Supervisor Call, plays a role in managing and executing these tasks.
- **Executive :**
  - It is a cyclic process that is used to control the starting and managing the timing and synchronizing the principal functions and I/Os.
  - It involves managing the phasing and sequencing between significant operational modes and the OPS through a Dispatcher Table Update (DTU) module.
  - This strategy enables flexible control and modification of the execution sequence of primary functions, accommodating dynamic adjustments as required.
- There are three executive structures those are integrated into the Guidance, Navigation, and Control *GN&C* design to ensure the timely completion of critical flight control processing within a 40-millisecond minor cycle.
- The high-frequency executive operates at a priority level that enables it to cycle at a rate of 25 Hz, initiating principal functions directly linked to vehicle flight control.
- Additionally, mid-frequency and low-frequency executives, scheduled at lower priorities, initiate principal function processes with the rates ranging from 6.25 Hz to 0.25 Hz.

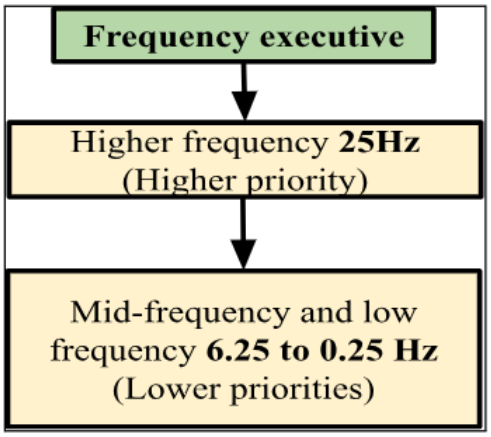


Figure 6: Frequency executive architecture

- **SPEC Control Management :**
  - The OPS substructure includes a second element known as the specialist function (SPEC).
  - Unlike major modes, SPEC is used for secondary functions and is activated within an OPS only in response to a keyboard entry by the crew.
  - After initialization, SPEC runs independently and concurrently with other processing within the OPS.
- **Initialization Module :** Loading and initialization are performed by the System Control.
- **Keyboard Requests :**
- – Communication links are formed between the keyboard/display units and the avionic subsystems they support throughout the execution of all Operational Sequences (OPSs).
  - A recurrent procedure is initiated at the systems level to ensure system integrity.
  - This process establishes the 40-millisecond timing cycle that is fundamental to the avionic system architecture and guarantees consistent service to these interfaces.
- **User Interface Software :** The user interface software manages communication between users and various systems or applications, overseeing intercomputer communication and ensuring redundant computer synchronization as part of the systems software.

(b) **Q: 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: Advantages

i. **Predictability and Determinism:** The cyclic priority scheduling of the Frequency Executive ensure that critical tasks are executed within predictable time frames, which is essential for the real-time requirements of spacecraft control. This predictability provides a high level of determinism in system behavior.

ii. **Modularity and Flexibility:** The use of a table-driven dispatching design allows for flexibility in modifying the sequencing of tasks as mission requirements change, without the need for significant code changes. This modularity supports the adaptability of the software to different phases of a space mission and facilitates updates and maintenance.

iii. **Efficient Resource Utilization:** By dividing tasks based on their frequency requirements and assigning them to different executives, the architecture enables efficient utilization of computing resources. This approach helps in managing the limited computational capacity of onboard systems by ensuring that high-priority tasks have the resources they need when they need them.

Disadvantages

i. **Complexity in Management and Scheduling:** While the Frequency Executive architecture offers flexibility and predictability, it also introduces complexity in the management and scheduling of tasks. Developers need to carefully plan and coordinate the execution frequencies and priorities of tasks to avoid conflicts and ensure that all tasks meet their timing requirements.

ii. **Limited Scalability:** As the number of tasks and their complexity grow, managing them within the constraints of the Frequency Executive architecture can become challenging. The fixed cyclic nature and the need for predefined scheduling tables may limit scalability and the ability to dynamically adjust to unforeseen operational demands.

iii. **Potential for Resource Under-Utilization:** While the architecture is to optimize resource utilization, the fixed scheduling can lead to periods where computational resources are underutilized, if tasks do not align perfectly with the executive cycles. This inefficiency could be a drawback in scenarios where maximizing computational throughput is critical.

3 Question 3

Q: [5 points] Read the paper "Building Safety-Critical Real-Time Systems with Reuseable Cyclic Executives", available from 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:

The Cyclic Executive

Concept: The Cyclic Executive is a simple and deterministic scheduling method primarily used in hard real-time systems. It operates on a fixed schedule known as a major cycle, divided into minor cycles. Each minor cycle is further divided into frames, with each frame allocated to a specific task or a set of tasks. The tasks are executed in a predefined sequence, repeating in every major cycle. This approach eliminates the need for dynamic task scheduling, as the execution pattern is statically determined during the system design phase.

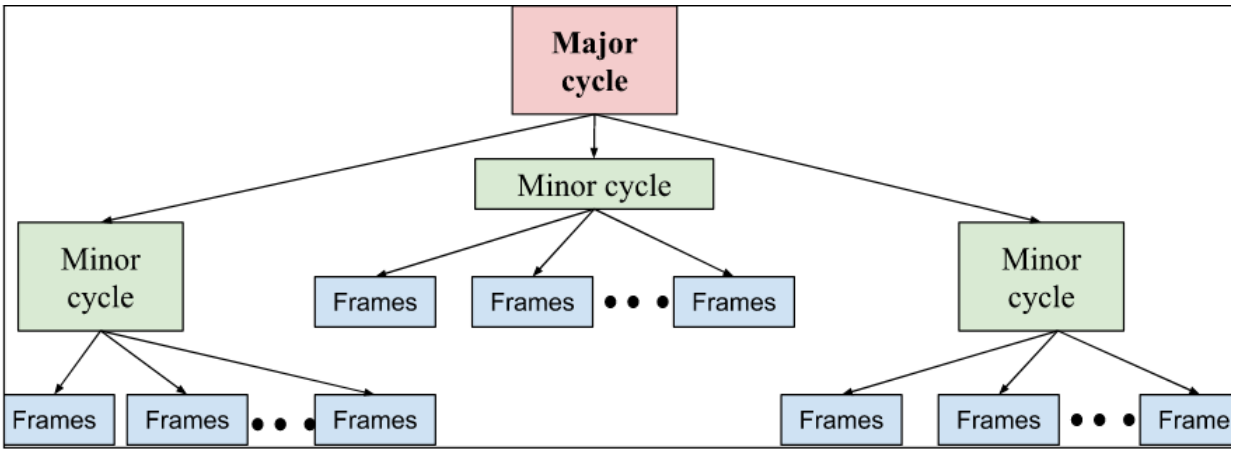


Figure 7: Cyclic Executive

Scheduling Structure:

i. **Major Cycle:** Represents the highest-level schedule, defining the sequence of processes that execute during a major cycle. This sequence cyclically repeats until the system's operating mode changes.

ii. **Minor Cycle:** Divides the major cycle into smaller units, and each tick of the minor cycle clock indicates the end of a minor cycle. Synchronization is enforced by the ticks of the minor cycle clock, ensuring that real-time requirements are met. Minor cycle clock ticks indicate the end of a minor cycle, and processes must complete their execution within this timeframe to avoid minor cycle overruns.

iii. **Frame:** Within each minor cycle, frames further divide the schedule. Only one process in the sequence corresponding to the minor schedule executes within a frame.

Interrupt Service Routine (ISR): In embedded systems, ISRs are commonly used to handle hardware interrupts and time-sensitive tasks. In Linux RT, ISRs can be implemented using kernel threads or dedicated real-time threads with high priority

Linux POSIX RT Threading

Concept: POSIX RT threading in Linux provides a set of standardized APIs for developing real-time applications within a general-purpose operating system environment. It includes features for creating real-time threads, setting thread priorities, and employing synchronization mechanisms like mutexes and condition variables.

Characteristics:

Preemptive Multitasking: Higher-priority threads can preempt lower-priority ones, allowing urgent real-time tasks to interrupt less critical tasks. **Concurrency:** Supports the concurrent execution of multiple threads, facilitating complex real-time applications. **Non-Deterministic Overheads:** Running atop a general-purpose OS, POSIX RT threads may encounter non-deterministic behaviors due to OS overheads and other system activities

Comparison

Table 1: Comparison Table

	Cyclic Executive	**RTOS**	**Linux POSIX with Real-Time Services**
Primary Usage	Deeply embedded systems (eg.shuttle flight)	Embedded systems, scalable and portable (eg. Vx Works)	General-purpose systems with real-time needs (eg. concurrent)
Scheduling approach	Custom, fixed sequence.	Preemptive and cooperative scheduling	Preemptive scheduling with real-time extensions
Task Management	Limited or none	Comprehensive task management	Supports multitasking and task synchronization
Overhead	Low	Medium	High
Maintenance	Costly due to hard coding	Simpler maintenance due to task abstractions	Requires constant updates, higher maintenance
Flexibility	Highly custom, efficient	Moderate flexibility	High flexibility applications
Licensing	Usually free or low cost.	Licensing costs may apply	Licensing costs may apply, open-source options available
Developement time	Longer development time	Quick	Balanced, quicker than cyclic executive, slower than RTOS

4 Question 4

Q: [50 points] CUSTOM FEASIBLITY TEST CODE. Download Feasibility example code (or get it from Canvas) and build it on a Jetson, Raspberry Pi, DE1-SoC or TIVA or Virtual Box and execute the code.

(a) **Q: Compare the feasibility tests provided by the example code to analysis using Cheddar for the first 5 examples (0-4).**
Answer:

i. **Example 0**

A. **Cheddar Output:**

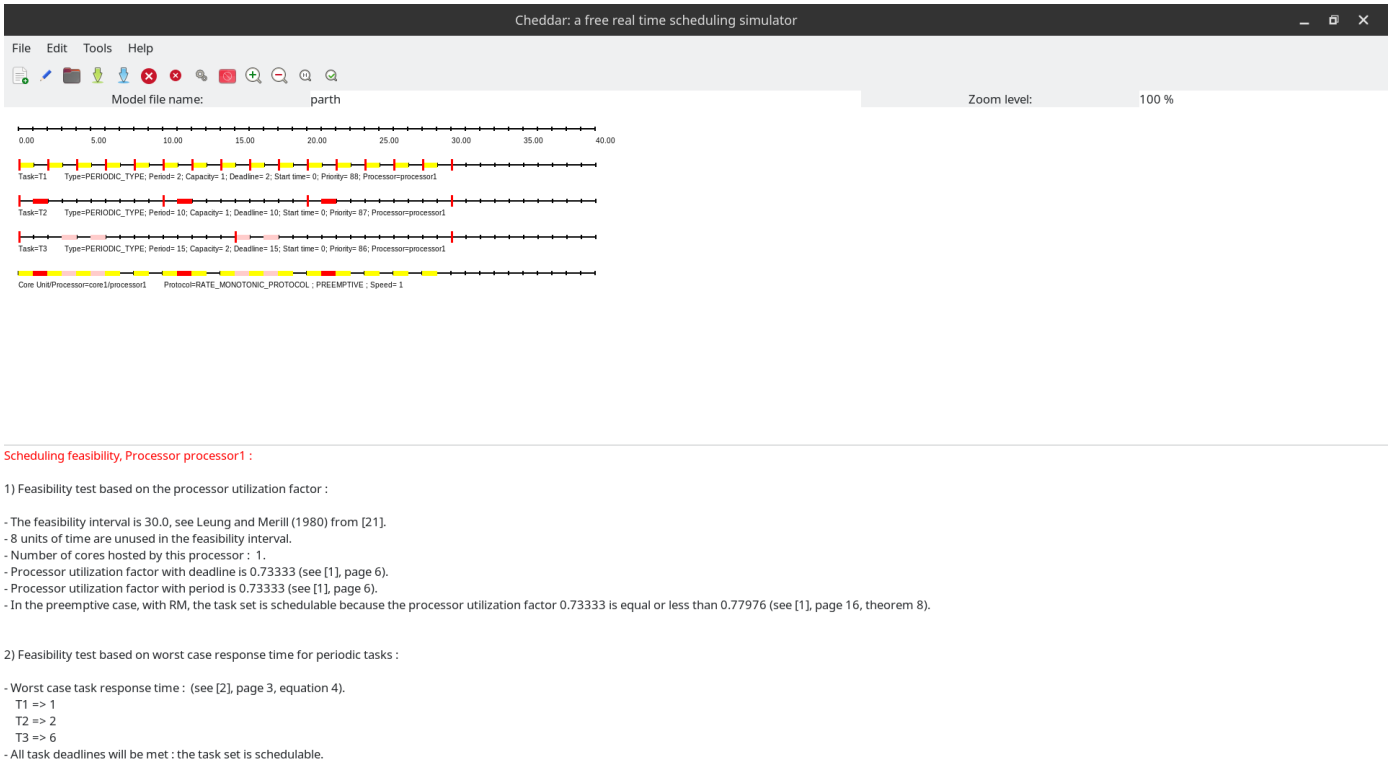


Figure 8: Example 0, RM test

B. **Code Output:**

```
***************
Example 0
C: 1 1 2
T: 2 10 15
D: 2 10 15

Task 0, WCET=1, Period=2, Utility Sum = 0.500000
Task 1, WCET=1, Period=10, Utility Sum = 0.600000
Task 2, WCET=2, Period=15, Utility Sum = 0.733333

Total Utility Sum = 0.733333
LUB = 0.779763
```

```
RM LUB: Feasible
Completion time feasibility: Feasible
Scheduling point feasibility: Feasible
```

C. **Conclusion:**

We found out that the Least Upper Bound (LUB) is 0.779763, both in our calculations and using the Cheddar tool, so our results match perfectly. The CPU usage we worked out is about 73.33%, and this number is the same when we look at how much of the CPU is used in the Cheddar tool, considering the time tasks take and their deadlines.

Because the LUB value is higher than the actual CPU usage RM_LUB would fail in the test, but all our checks for nessesarry and sufficient tests Completion time and Scheduling point feasibility shows that the tasks are feasible with RM schedule. This means we set up our tasks in a way that they can all get done within their set times without overloading the CPU. And we can confirm the results of no deadline miss in the cheddar.

ii. **Example 1**

A. **Cheddar Output:**

Figure 9: Example 1, RM analysis

B. **Code Output:**

```
***************
Example 1
C: 1 1 2
T: 2 5 7
D: 2 5 7

Task 0, WCET=1, Period=2, Utility Sum = 0.500000
Task 1, WCET=1, Period=5, Utility Sum = 0.700000
```

```
Task 2, WCET=2, Period=7, Utility Sum = 0.985714

Total Utility Sum = 0.985714
LUB = 0.779763
RM LUB: Infeasible
Completion time feasibility: Infeasible
Scheduling point feasibility: Infeasible
```

C. **Conclusion:**

We found that the Least Upper Bound (LUB) value is 0.779763, both in our code and when we checked with the Cheddar software, so our findings match. However, the CPU usage we calculated is quite high, at about 98.57%, and this same high usage shows up in Cheddar as well when looking at task periods and deadlines.

Because the LUB value is higher than the actual CPU usage RM_LUB would fail in the test, and our checks for nessesarry and sufficient tests Completion time and Scheduling point tests are failing, that indicate that the tasks we set up can't all be completed within their deadlines according to Rate Monotonic (RM) scheduling rules. This means our tasks are set up in a way that causes some to miss their deadlines. Our results and the Cheddar output confirm that these tasks aren't feasible for the completion test and scheduling point feasibility.
So these tasks are not Schedulable by RM policy.

iii. **Example 2**

A. **Cheddar Output:**

Figure 10: Example 2, RM analysis

B. **Code Output:**

```
***************
Example 2
C: 1 1 1 2
T: 2 5 7 13
D: 2 5 7 13

Task 0, WCET=1, Period=2, Utility Sum = 0.500000
Task 1, WCET=1, Period=5, Utility Sum = 0.700000
Task 2, WCET=1, Period=7, Utility Sum = 0.842857
Task 3, WCET=2, Period=13, Utility Sum = 0.996703

Total Utility Sum = 0.996703
LUB = 0.756828
RM LUB: Infeasible
Completion time feasibility: Infeasible
Scheduling point feasibility: Infeasible
```

C. **Conclusion:**
We found that the Least Upper Bound (LUB) value is 0.756828, both in our code and when we checked with the Cheddar software, so our findings match. However, the CPU usage we calculated is quite high, at about 99.6703%, and this same high usage shows up in Cheddar as well when looking at task periods and deadlines.

Because the LUB value is higher than the actual CPU usage RM_LUB would fail in the test, and our checks for nessesarry and sufficient tests Completion time and Scheduling point tests are failing, that indicate that the tasks we set up can't all be completed within their deadlines according to Rate Monotonic (RM) scheduling rules. This means our tasks are set up in a way that causes some to miss their deadlines. Our results and the Cheddar output confirm that these tasks aren't feasible for the completion test and scheduling point feasibility.

So these tasks are not Schedulable by RM policy.

iv. **Example 3**

A. **Cheddar Output:**

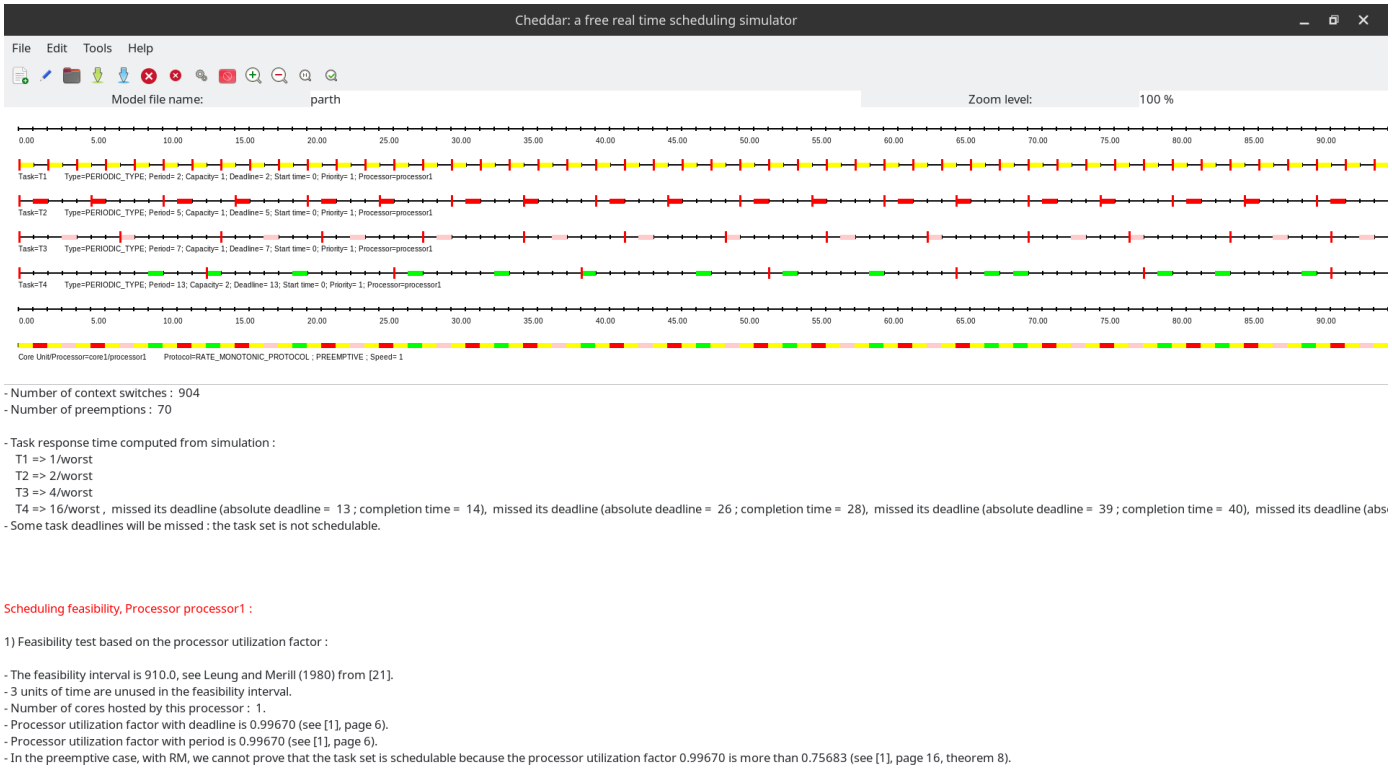


Figure 11: Example 3, RM analysis

B. **Code Output:**

```
***************
Example 3
C: 1 2 3
T: 3 5 15
D: 3 5 15

Task 0, WCET=1, Period=3, Utility Sum = 0.333333
Task 1, WCET=2, Period=5, Utility Sum = 0.733333
Task 2, WCET=3, Period=15, Utility Sum = 0.933333

Total Utility Sum = 0.933333
LUB = 0.779763
RM LUB: Infeasible
Completion time feasibility: Feasible
Scheduling point feasibility: Feasible
```

C. **Conclusion:**

We discovered that the Least Upper Bound (LUB) number is 0.779763, a figure that matches both in our own calculations and in the Cheddar software tests. However, we noticed that the CPU usage is really high, around 93.33%, and this level of usage is also reflected in the Cheddar analysis when we look at the timing of tasks and their deadlines.

Because the LUB value is higher than the actual CPU usage RM_LUB would fail in the test, but all our checks for nessesarry and sufficient tests Completion time and Scheduling point feasibility shows that the tasks are feasible with RM schedule. This means we set up our tasks in a way that they can all get done within their set times without overloading the CPU. And we can confirm the results of no deadline miss in the cheddar.

So these tasks are Schedulable by RM policy.

v. **Example 4**

A. **Cheddar Output:**

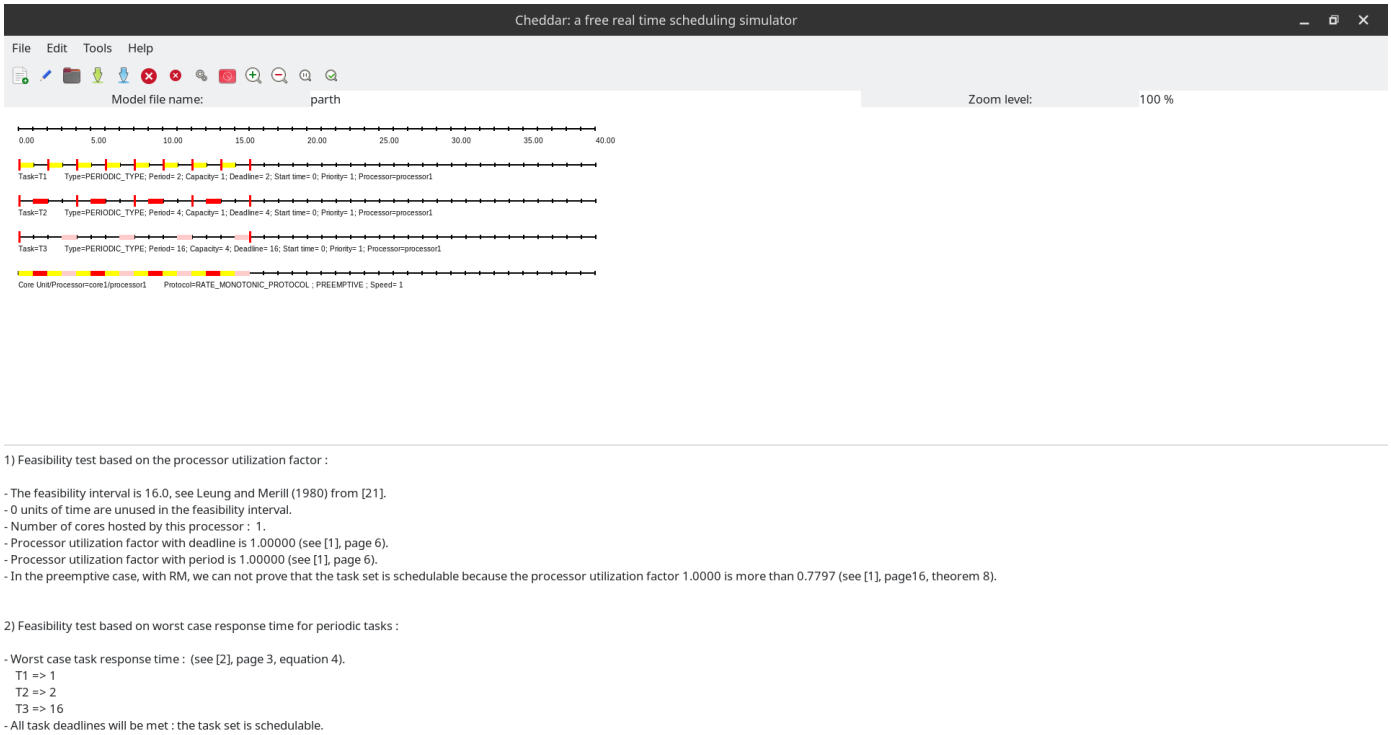


Figure 12: Example 4, RM analysis

B. **Code Output:**

```
***************
Example 4
C: 1 1 4
T: 2 4 16
D: 2 4 16

Task 0, WCET=1, Period=2, Utility Sum = 0.500000
Task 1, WCET=1, Period=4, Utility Sum = 0.750000
Task 2, WCET=4, Period=16, Utility Sum = 1.000000

Total Utility Sum = 1.000000
LUB = 0.779763
RM LUB: Infeasible
Completion time feasibility: Feasible
Scheduling point feasibility: Feasible
```

C. **Conclusion:**

The Least Upper Bound (LUB) number is 0.779763, a figure that matches both in our own calculations and in the Cheddar software tests. However, we noticed that the CPU usage is really high, around 100%, and this level of usage is also reflected in the Cheddar analysis when we look at the timing of tasks and their deadlines.

Because the LUB value is higher than the actual CPU usage RM_LUB would fail in the test, but all our checks for nessesarry and sufficient tests Completion time and Scheduling point feasibility shows that the tasks are feasible with RM schedule. This means we set up our tasks in a way that they can all get done within their set times without overloading the CPU. And we can confirm the results of no deadline miss in the cheddar.

So these tasks are Schedulable by RM policy.

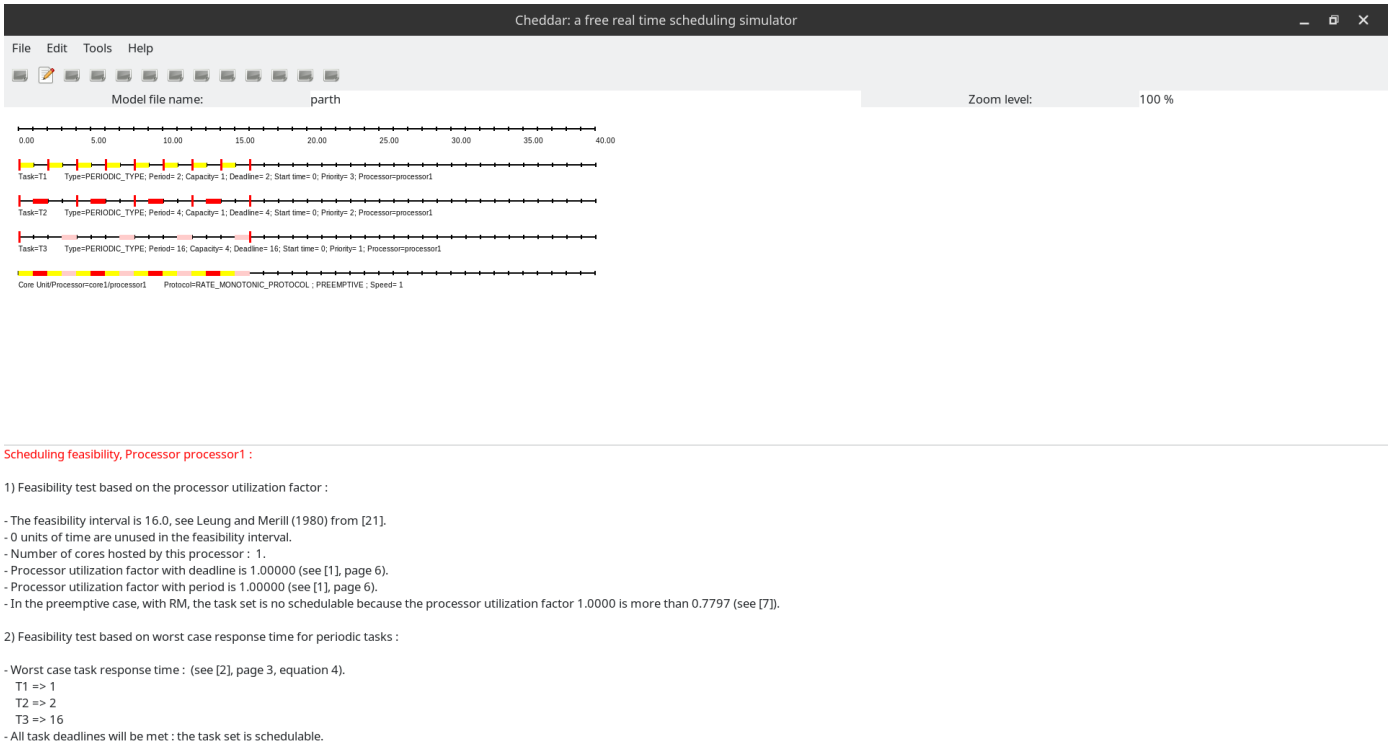
(b) **Q: Now, implement the remaining examples [5 more, 5-9] that we reviewed in class (found here, and on Canvas) by modifying the example code to include the other examples. Complete analysis for all three policies using Cheddar (RM, EDF, LLF) and by adding EDF and LLF feasibility tests to the code, except for example 6, which should use RM and DM. 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".**

Answer:

i. **Example 5**

A. **Cheddar Output:**

RM schedule



Scheduling feasibility, Processor processor1 :

1) Feasibility test based on the processor utilization factor :

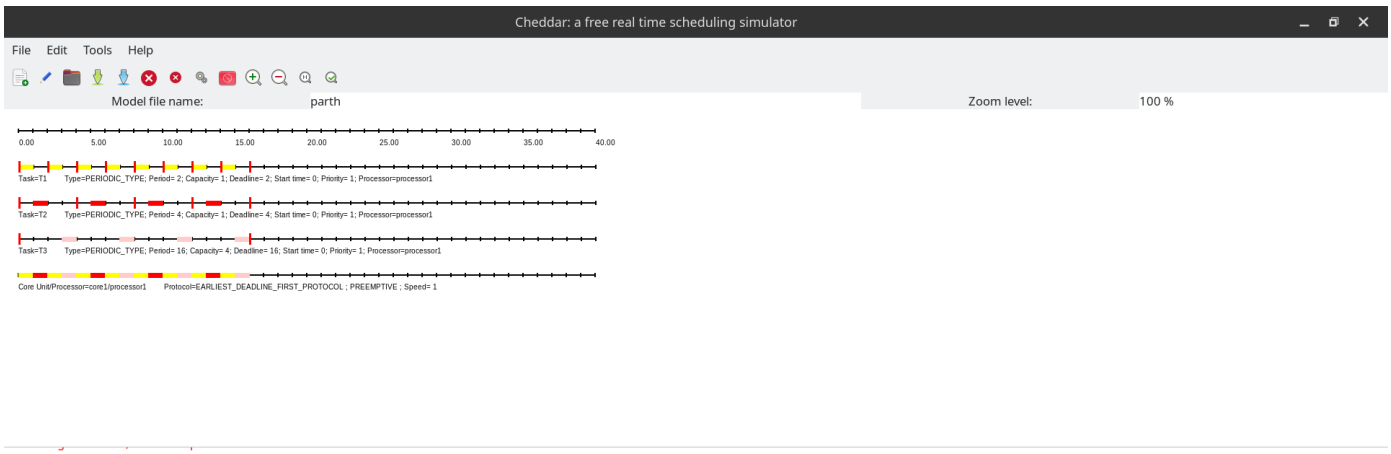
- The feasibility interval is 16.0, see Leung and Merill (1980) from [21].
- 0 units of time are unused in the feasibility interval.
- Number of cores hosted by this processor : 1.
- Processor utilization factor with deadline is 1.00000 (see [1], page 6).
- Processor utilization factor with period is 1.00000 (see [1], page 6).
- In the preemptive case, with RM, the task set is no schedulable because the processor utilization factor 1.0000 is more than 0.7797 (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).
  T1 => 1
  T2 => 2
  T3 => 16
- All task deadlines will be met : the task set is schedulable.

Figure 13: Example 5, RM analysis

EDF Schedule



- Number of context switches : 15
- Number of preemptions : 3

- Task response time computed from simulation :
  T1 => 1/worst
  T2 => 2/worst
  T3 => 16/worst
- No deadline missed in the computed scheduling : the task set is schedulable if you computed the scheduling on the feasibility interval.

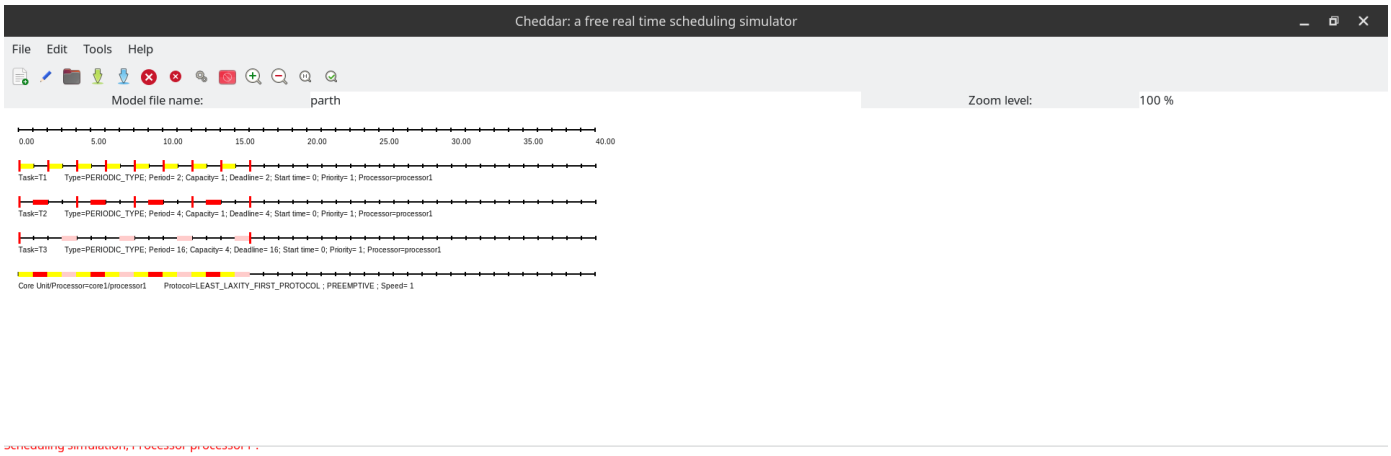
Scheduling feasibility, Processor processor1 :

1) Feasibility test based on the processor utilization factor :

- The feasibility interval is 16.0, see Goossens and Devillers (1997) from [21].
- 0 units of time are unused in the feasibility interval.
- Number of cores hosted by this processor : 1.
- 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).

Figure 14: Example 5, EDF analysis

LLF Schedule



- Number of context switches : 15
- Number of preemptions : 3

- Task response time computed from simulation :
  T1 => 1/worst
  T2 => 2/worst
  T3 => 16/worst
- No deadline missed in the computed scheduling : the task set is schedulable if you computed the scheduling on the feasibility interval.

Scheduling feasibility, Processor processor1 :

1) Feasibility test based on the processor utilization factor :

- The feasibility interval is 16.0, see Leung and Merrill (1980) from [21].
- 0 units of time are unused in the feasibility interval.
- Number of cores hosted by this processor : 1.
- 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]).

Figure 15: Example 5, LLF analysis

B. **Code Output:**

```
***************
Example 5
C: 1 1 4
T: 2 4 16
D: 2 4 16

Task 0, WCET=1, Period=2, Utility Sum = 0.500000
Task 1, WCET=1, Period=4, Utility Sum = 0.750000
Task 2, WCET=4, Period=16, Utility Sum = 1.000000

Total Utility Sum = 1.000000
LUB = 0.779763
RM LUB: Infeasible
Completion time feasibility: Feasible
Scheduling point feasibility: Feasible

(Period)
Total utility in EDF: 1.000000 Which is less than 1.0
EDF: Feasible
Total utility in LLF: 1.000000 Which is less than 1.0
LLF: Feasible
```

C. **Conclusion:**
The Least Upper Bound (LUB) number is 0.779763, a figure that matches both in our own calculations and in the Cheddar software tests. However, we noticed that the CPU usage is really high, around 100%, and this level of usage is also reflected in the Cheddar analysis when we look at the timing of tasks and their deadlines.

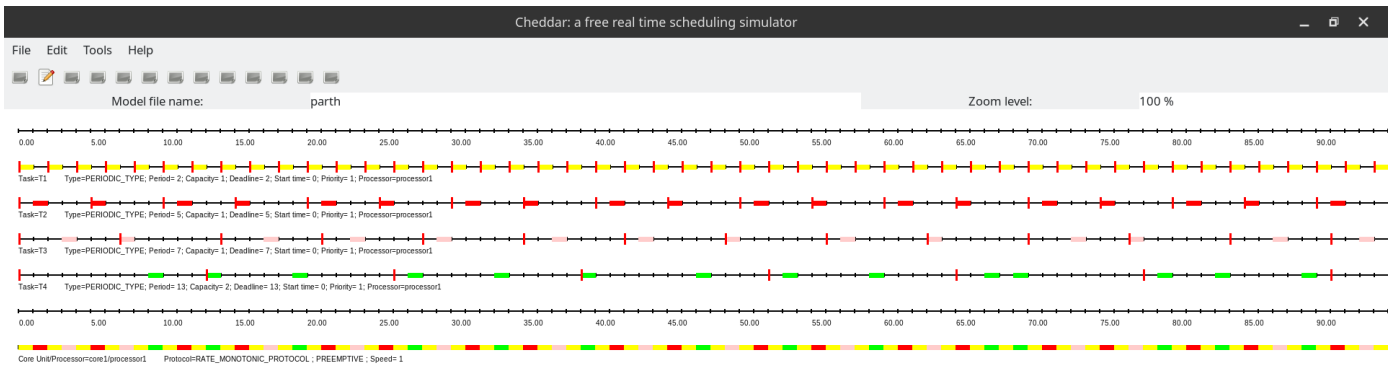
Because the LUB value is higher than the actual CPU usage RM_LUB would fail in the test, but all our checks for nessesarry and sufficient tests Completion time and Scheduling point feasibility shows that the tasks are feasible with RM schedule. This means we set up our tasks in a way that they can all get done within their set times without overloading the CPU. And we can confirm the results of no deadline miss in the cheddar.

The output of code for the EDF and LLF shows that these set of tasks are schedulable with EDF and LLF as the utilization is less than or equal to 100% and we can see in the cheddar output that EDF and LLF are indeed feasible. Cheddar shows that in EDF or LLF, no deadlines are missed and therefor these set of tasks are feasible for EDF and LLF.

So, these task are not schedulable by RM but can be schedulable by EDF or LLF Scheduling.

ii. **Example 6**

A. **Cheddar Output:**
RM schedule



Scheduling simulation, Processor processor1 :

- Number of context switches : 904
- Number of preemptions : 70

- Task response time computed from simulation :
  T1 => 1/worst
  T2 => 2/worst
  T3 => 4/worst
  T4 => 16/worst , missed its deadline (absolute deadline = 13 ; completion time = 14), missed its deadline (absolute deadline = 26 ; completion time = 28), missed its deadline (absolute deadline = 39 ; completion time = 40), missed its deadline (abs
- Some task deadlines will be missed : the task set is not schedulable.

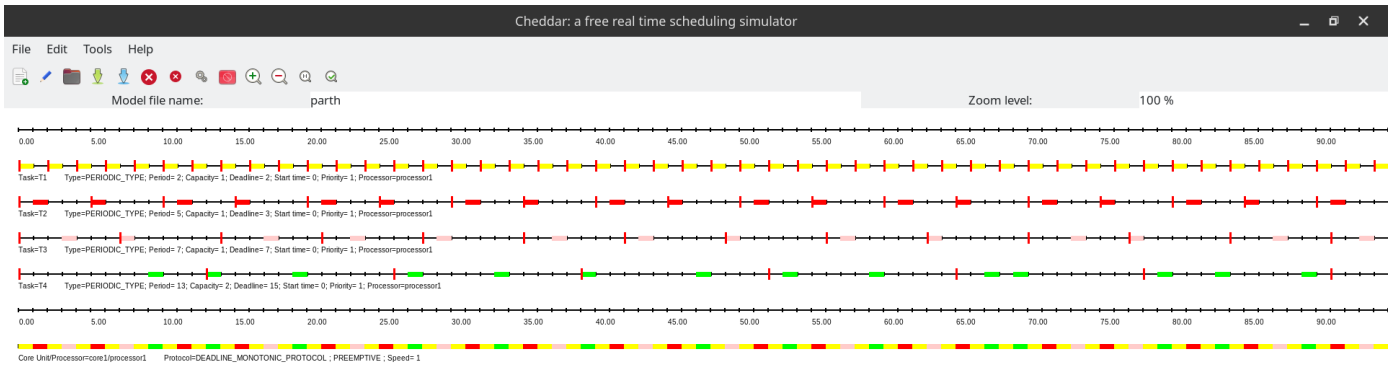
Scheduling feasibility, Processor processor1 :

1) Feasibility test based on the processor utilization factor :

- The feasibility interval is 910.0, see Leung and Merill (1980) from [21].
- 3 units of time are unused in the feasibility interval.
- Number of cores hosted by this processor : 1.
- 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 cannot 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).

Figure 16: Example 6, RM analysis

EDF Schedule



Scheduling simulation, Processor processor1 :

- Number of context switches : 904
- Number of preemptions : 70

- Task response time computed from simulation :
  T1 => 1/worst
  T2 => 2/worst
  T3 => 4/worst
  T4 => 16/worst , missed its deadline (absolute deadline = 67 ; completion time = 68)
- Some task deadlines will be missed : the task set is not schedulable.

Scheduling feasibility, Processor processor1 :

1) Feasibility test based on the processor utilization factor :

- The feasibility interval is 910.0, see Leung and Merrill (1980) from [21].
- 3 units of time are unused in the feasibility interval.
- Number of cores hosted by this processor : 1.
- Processor utilization factor with deadline is 1.10952 (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 1.10952 is more than 0.75683 (see [7]).

Figure 17: Example 6, Deadline Monotonic analysis

B. Code Output:

```
***************
Example 6
C: 1 1 1 2
T: 2 5 7 13
D: 2 3 7 15

Task 0, WCET=1, Period=2, Utility Sum = 0.500000
Task 1, WCET=1, Period=5, Utility Sum = 0.700000
Task 2, WCET=1, Period=7, Utility Sum = 0.842857
Task 3, WCET=2, Period=13, Utility Sum = 0.996703

Total Utility Sum = 0.996703
LUB = 0.756828
RM LUB: Infeasible
Completion time feasibility: Infeasible
Scheduling point feasibility: Infeasible
Deadline monotonic: Infeasible

(Period)
Total utility in EDF: 0.996703 Which is less than 1.0
EDF on Period: Feasible
Total utility in LLF: 0.996703 Which is less than 1.0
LLF on Period: Feasible

(Deadline)
Total utility in EDF: 1.109524 Which is less than 1.0
EDF on Deadline: Infeasible
Total utility in LLF: 1.109524 Which is less than 1.0
LLF on Deadline: Infeasible
```

C. **Conclusion:**
The Least Upper Bound (LUB) number is 0.756828, a figure that matches both in our own calculations and in the Cheddar software tests. However, we noticed that the CPU usage is really high, around 99.67%, and this level of usage is also reflected in the Cheddar analysis when we look at the timing of tasks and their deadlines.

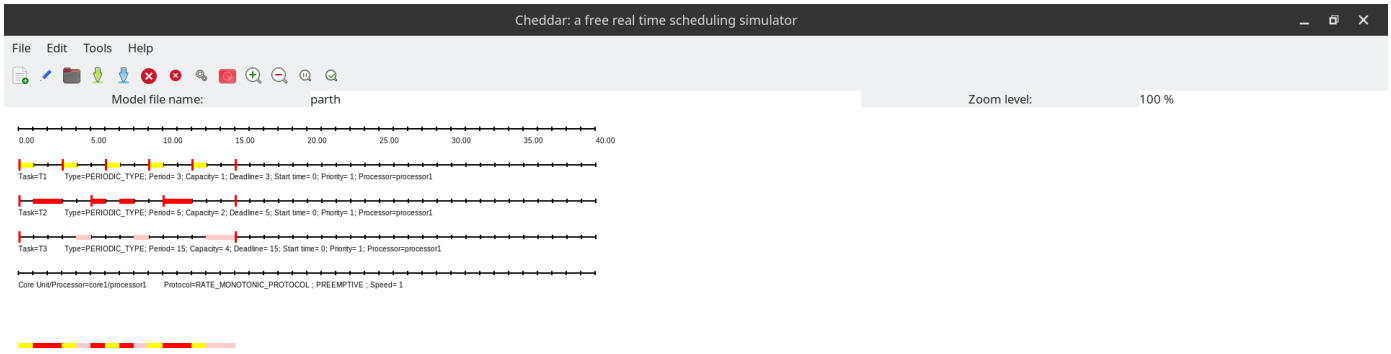
Because the LUB value is higher than the actual CPU usage RM_LUB would fail in the test, and our checks for nessesarry and sufficient tests Completion time and Scheduling point tests are failing, that indicate that the tasks we set up can't all be completed within their deadlines according to Rate Monotonic (RM) scheduling rules. This means our tasks are set up in a way that causes some to miss their deadlines. Our results and the Cheddar output confirm that these tasks aren't feasible for the completion test and scheduling point feasibility.

Also Deadline monotonic feasibility test is failing in our case that can be confirmed by Cheddar's output for Deadline monotonic scheduling, we can see that in deadline monotonic scheduling cheddar shows one deadline is being missed.

So, these task are not schedulable by Either RM or Deadline Monotonic Scheduling.

iii. **Example 7**

A. **Cheddar Output:**
RM schedule



Scheduling simulation, Processor processor1 :

- Number of context switches : 11
- Number of preemptions : 3

- Task response time computed from simulation :
  T1 => 1/worst
  T2 => 3/worst
  T3 => 15/worst
- No deadline missed in the computed scheduling : the task set is schedulable if you computed the scheduling on the feasibility interval.

Scheduling feasibility, Processor processor1 :

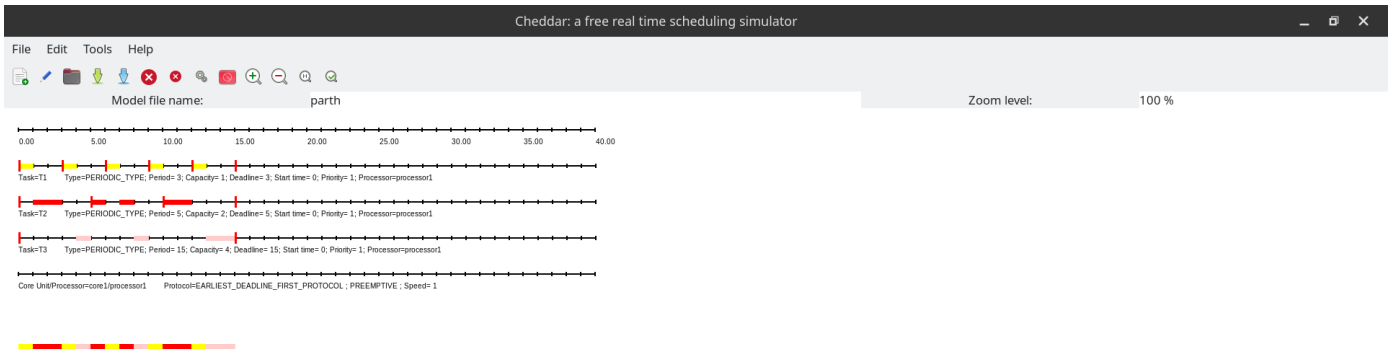
1) Feasibility test based on the processor utilization factor :

- The feasibility interval is 15.0, see Leung and Merill (1980) from [21].
- 0 units of time are unused in the feasibility interval.
- Number of cores hosted by this processor : 1.
- 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 cannot 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 :

Figure 18: Example 7, RM analysis

EDF Schedule



Scheduling simulation, Processor processor1 :

- Number of context switches : 11
- Number of preemptions : 3

- Task response time computed from simulation :
  T1 => 1/worst
  T2 => 3/worst
  T3 => 15/worst
- No deadline missed in the computed scheduling : the task set is schedulable if you computed the scheduling on the feasibility interval.

Scheduling feasibility, Processor processor1 :

1) Feasibility test based on the processor utilization factor :

- The feasibility interval is 15.0, see Goossens and Devillers (1997) from [21].
- 0 units of time are unused in the feasibility interval.
- Number of cores hosted by this processor : 1.
- 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 :

Figure 19: Example 7, EDF analysis

LLF Schedule

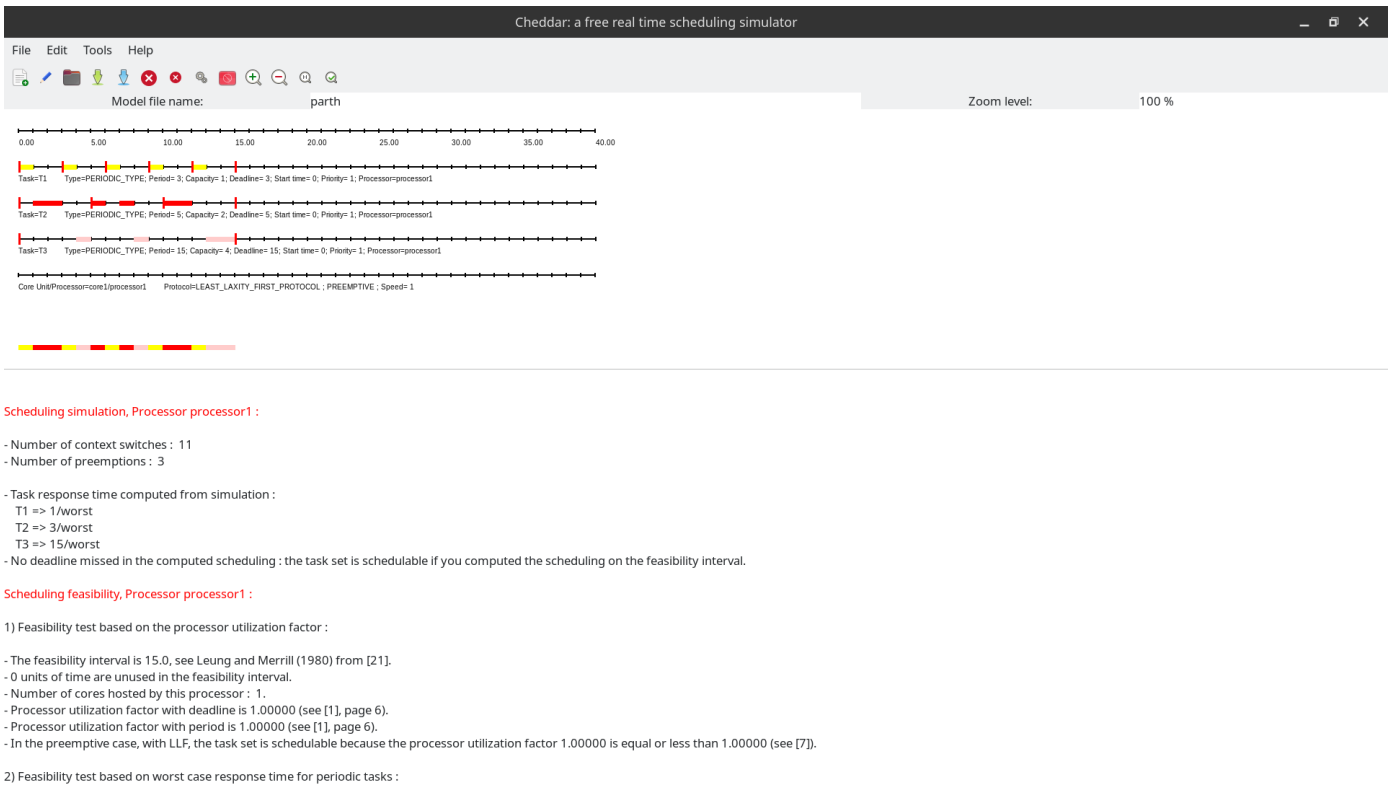


Figure 20: Example 7, LLF analysis

B. **Code Output:**

```
***************
Example 7
C: 1 2 4
T: 3 5 15
D: 3 5 15

Task 0, WCET=1, Period=3, Utility Sum = 0.333333
Task 1, WCET=2, Period=5, Utility Sum = 0.733333
Task 2, WCET=4, Period=15, Utility Sum = 1.000000

Total Utility Sum = 1.000000
LUB = 0.779763
RM LUB: Infeasible
Completion time feasibility: Feasible
Scheduling point feasibility: Feasible

(Period)
Total utility in EDF: 1.000000 Which is less than 1.0
EDF: Feasible
Total utility in LLF: 1.000000 Which is less than 1.0
LLF: Feasible
```

C. **Conclusion:**

The Least Upper Bound (LUB) number is 0.779763, a figure that matches both in our own calculations and in the Cheddar software tests. However, we noticed that the CPU usage is really high, around 100.00%, and this level of usage is also reflected in the Cheddar analysis when we look at the timing of tasks and their deadlines.

Because the LUB value is higher than the actual CPU usage RM_LUB would fail in the test, but all our checks for nessesarry and sufficient tests Completion time and Scheduling point feasibility shows that the tasks are feasible with RM schedule. This means we set up our tasks in a way that they can all get done within their set times without overloading the CPU. And we can confirm the results of no deadline miss in the cheddar.

The output of code for the EDF and LLF shows that these set of tasks are schedulable with EDF and LLF as the utilization is less than or equal to 100% and we can see in the cheddar output that EDF and LLF are indeed feasible. Cheddar shows that in EDF or LLF, no deadlines are missed and therefor these set of tasks are feasible for EDF and LLF.

So, these task are schedulable by RM ,EDF or LLF Scheduling.

iv. **Example 8**

A. **Cheddar Output:**
EDF Schedule

- Task response time computed from simulation :
  T1 => 1/worst
  T2 => 2/worst
  T3 => 6/worst
  T4 => 15/worst
- No deadline missed in the computed scheduling : the task set is schedulable if you computed the scheduling on the feasibility interval.

Scheduling feasibility, Processor processor1 :

1) Feasibility test based on the processor utilization factor :

- The feasibility interval is 910.0, see Goossens and Devillers (1999) from [21].
- 3 units of time are unused in the feasibility interval.
- Number of cores hosted by this processor : 1.
- Processor utilization factor with deadline is 1.10952 (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 not schedulable because the processor utilization factor 1.10952 is more than 1.00000 (see [1], page 8, theorem 2).

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

Figure 21: Example 8, EDF analysis

LLF Schedule

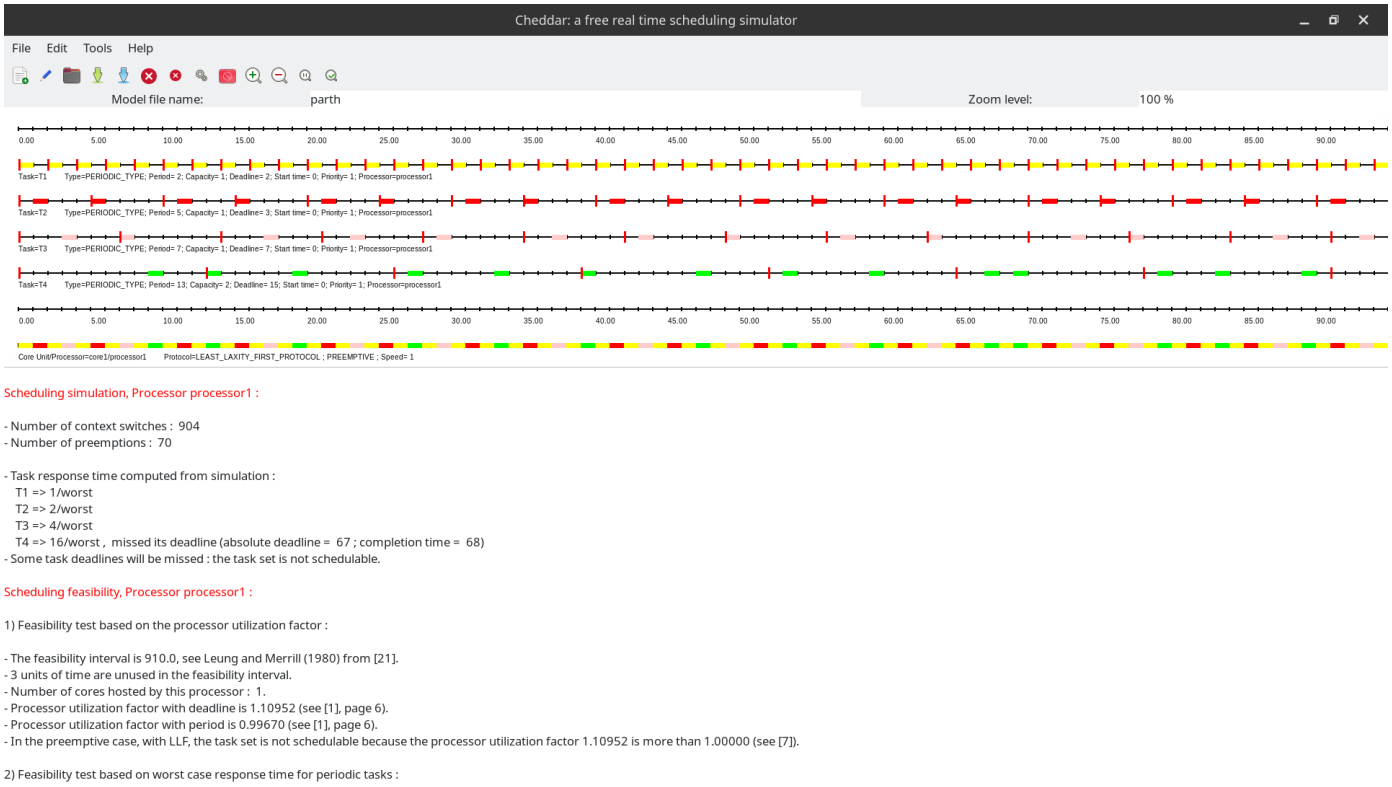


Figure 22: Example 8, LLF analysis

B. **Code Output:**

```
***************
Example 8
C: 1 1 1 2
T: 2 5 7 13
D: 2 5 7 13

Task 0, WCET=1, Period=2, Utility Sum = 0.500000
Task 1, WCET=1, Period=5, Utility Sum = 0.700000
Task 2, WCET=1, Period=7, Utility Sum = 0.842857
Task 3, WCET=2, Period=13, Utility Sum = 0.996703

Total Utility Sum = 0.996703
LUB = 0.756828
RM LUB: Infeasible
Completion time feasibility: Infeasible
Scheduling point feasibility: Infeasible

(Period)
Total utility in EDF: 0.996703 Which is less than 1.0
EDF: Feasible
Total utility in LLF: 0.996703 Which is less than 1.0
LLF: Feasible
```

C. **Conclusion:**

The Least Upper Bound (LUB) number is 0.756828, a figure that matches both in our own calculations and in the Cheddar software tests. However, we noticed that the CPU usage is really high, around 99.67%, and this level of usage is also reflected in the

Cheddar analysis when we look at the timing of tasks and their deadlines.

Because the LUB value is higher than the actual CPU usage RM_LUB would fail in the test, and our checks for nessesarry and sufficient tests Completion time and Scheduling point tests are failing, that indicate that the tasks we set up can't all be completed within their deadlines according to Rate Monotonic (RM) scheduling rules. This means our tasks are set up in a way that causes some to miss their deadlines. Our results and the Cheddar output confirm that these tasks aren't feasible for the completion test and scheduling point feasibility (RM_Schedule).

The output of code for the EDF and LLF shows that these set of tasks are schedulable with EDF and LLF as the utilization is less than or equal to 100% and we can see in the cheddar output that EDF and LLF are indeed feasible. Cheddar shows that in EDF or LLF, no deadlines are missed and therefor these set of tasks are feasible for EDF and LLF.

So, these task are not schedulable by RM but can be schedulable by EDF or LLF Scheduling.

v. **Example 9**

A. **Cheddar Output:**
RM schedule

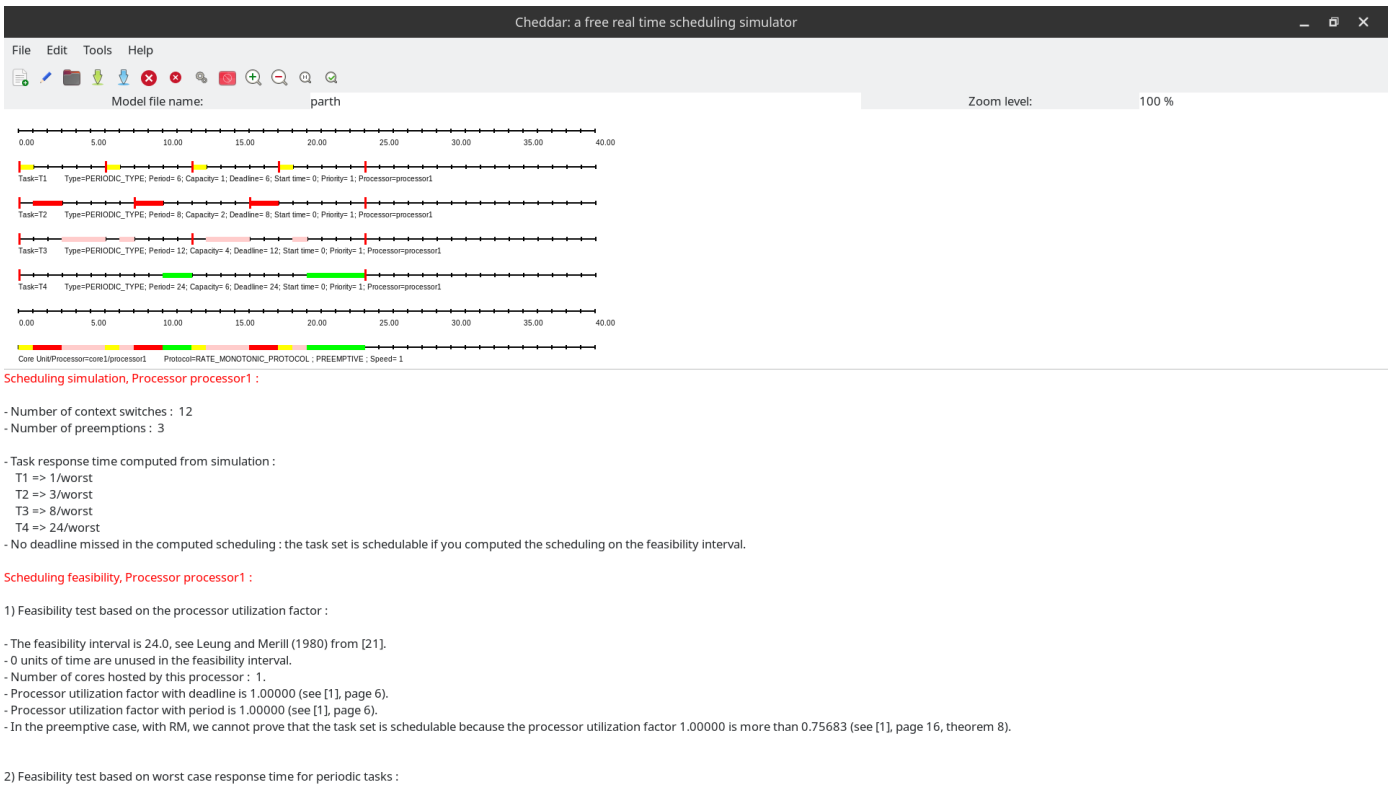


Figure 23: Example 9, RM analysis

EDF Schedule

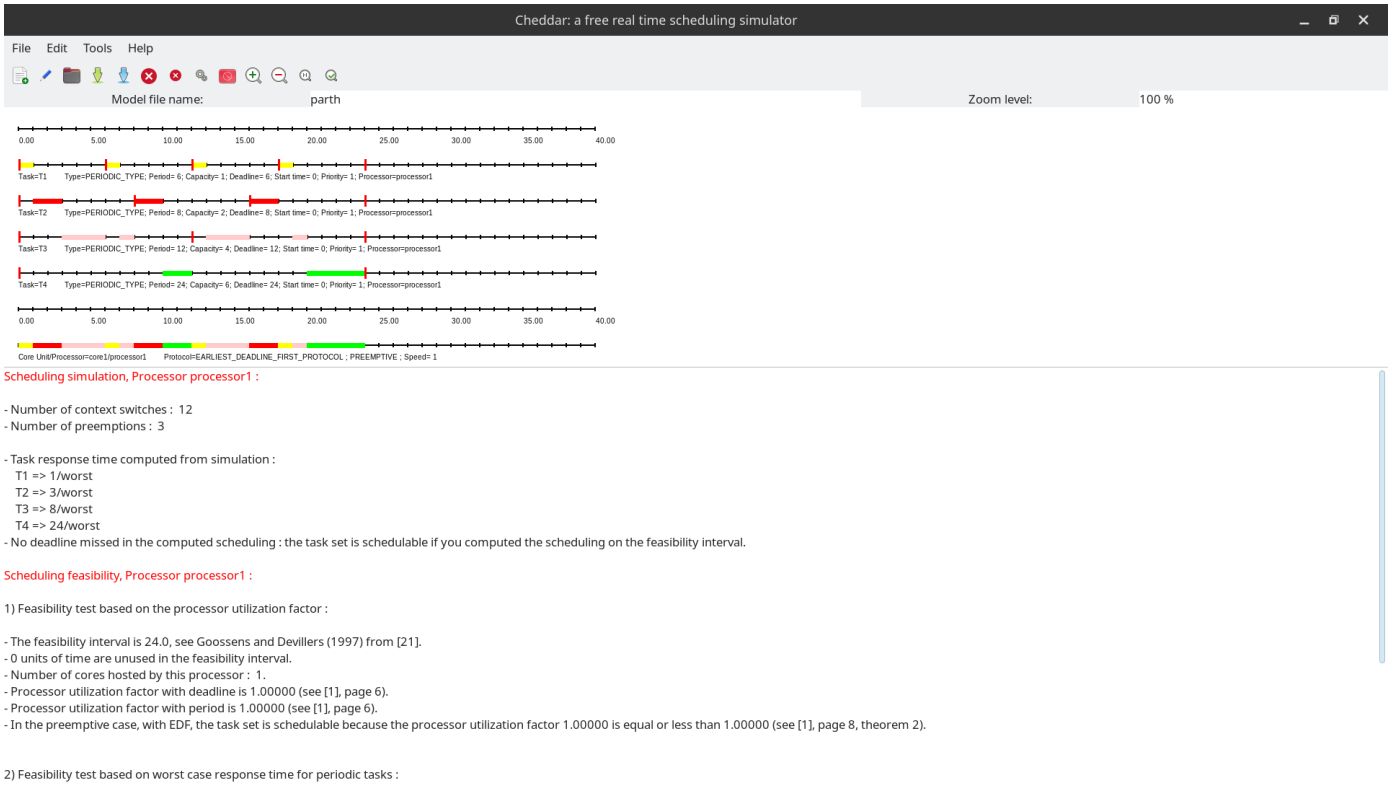


Figure 24: Example 9, EDF analysis

LLF Schedule

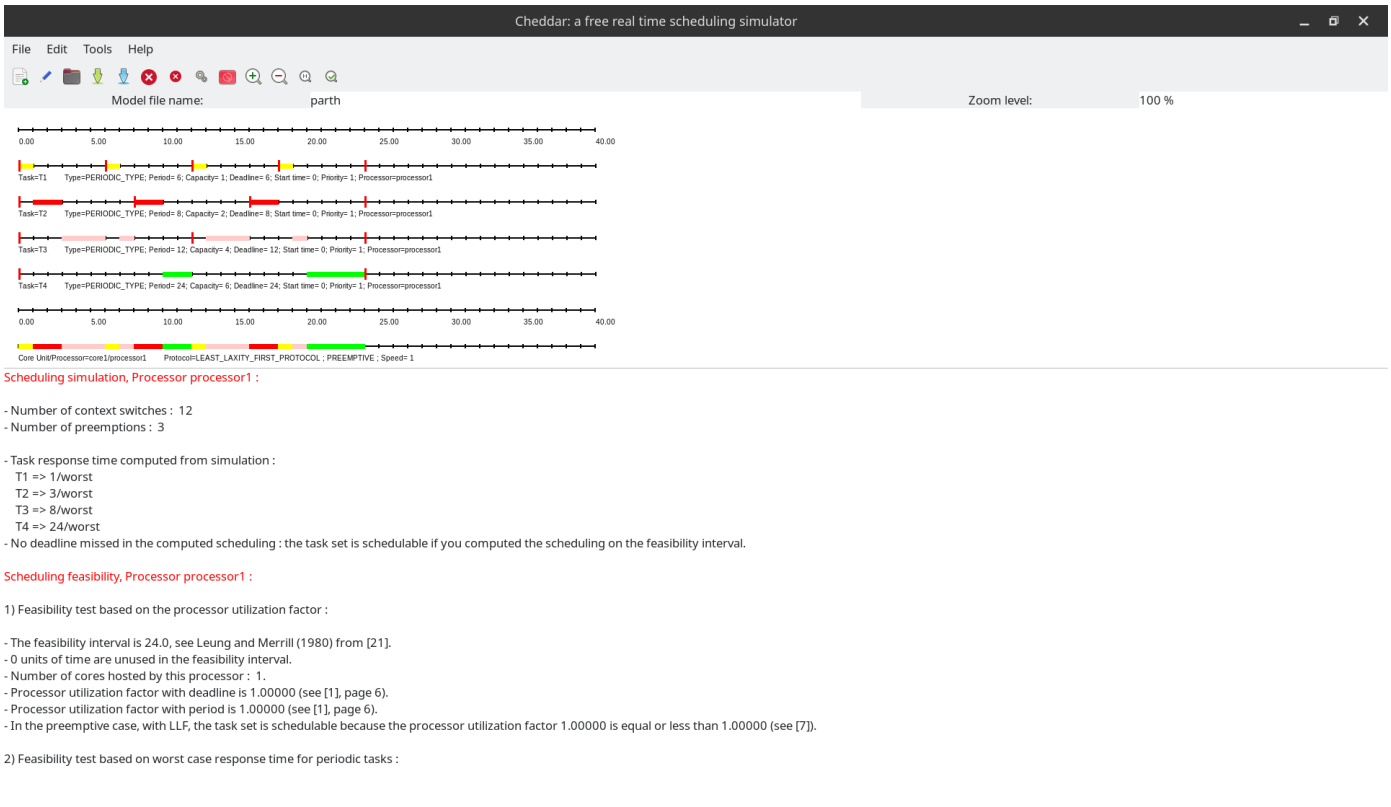


Figure 25: Example 9, LLF analysis

B. Code Output:

```
***************
Example 9
C: 1 2 4 6
T: 6 8 12 24
D: 6 8 12 24

Task 0, WCET=1, Period=6, Utility Sum = 0.166667
Task 1, WCET=2, Period=8, Utility Sum = 0.416667
Task 2, WCET=4, Period=12, Utility Sum = 0.750000
Task 3, WCET=6, Period=24, Utility Sum = 1.000000

Total Utility Sum = 1.000000
LUB = 0.756828
RM LUB: Infeasible
Completion time feasibility: Feasible
Scheduling point feasibility: Feasible

(Period)
Total utility in EDF: 1.000000 Which is less than 1.0
EDF: Feasible
Total utility in LLF: 1.000000 Which is less than 1.0
LLF: Feasible
```

C. **Conclusion:**
The Least Upper Bound (LUB) number is 0.756828, a figure that matches both in our own calculations and in the Cheddar software tests. However, we noticed that the CPU usage is really high, around 99.67%, and this level of usage is also reflected in the Cheddar analysis when we look at the timing of tasks and their deadlines.

Because the LUB value is higher than the actual CPU usage RM_LUB would fail in the test, but all our checks for nessesarry and sufficient tests Completion time and Scheduling point feasibility shows that the tasks are feasible with RM schedule. This means we set up our tasks in a way that they can all get done within their set times without overloading the CPU. And we can confirm the results of no deadline miss in the cheddar.

The output of code for the EDF and LLF shows that these set of tasks are schedulable with EDF and LLF as the utilization is less than or equal to 100% and we can see in the cheddar output that EDF and LLF are indeed feasible. Cheddar shows that in EDF or LLF, no deadlines are missed and therefor these set of tasks are feasible for EDF and LLF.

So, these task are schedulable by RM, EDF or LLF Policy.

(c) **Q: Does your modified Feasibility code agree with Cheddar analysis in all 5 additional cases? Why or why not? And why edf and llf do not miss deadlines in the example while RM misses the deadline**

Answer: Output of the cheddar and the code is equally right and it satisfies the feasible tests.

Now, let's Take example Example 8 in which RM is Infeasible and EDF and LLF are feasible,

Table 2: Tasks Parameters

Task	Capacity	Period	Deadline
S_1	$C_1 = 1$	$T_1 = 2$	$D_1 = 2$
S_2	$C_2 = 1$	$T_2 = 5$	$D_2 = 5$
S_3	$C_3 = 1$	$T_3 = 7$	$D_3 = 7$
S_4	$C_4 = 2$	$T_4 = 13$	$D_4 = 13$

Given the task characteristics, let's analyze why RM (Rate Monotonic) scheduling might miss deadlines, even though the total utilization is below 1, indicating that EDF (Earliest Deadline First) and LLF (Least Laxity First) would be feasible.

Rate Monotonic (RM) Scheduling In RM scheduling, tasks are assigned priorities based on their period lengths, with shorter periods getting higher priority. Accordingly, the priorities here would be:

Highest Priority: Task 0 Lower Priorities: Task 1 > Task 2 > Task 3

Analysis for Deadline Miss
To understand where a deadline miss occurs in RM scheduling, we'll examine the scheduling timeline closely.
Initial Phase (First 13 Units of Time): The crucial point to examine is the first 13 units of time, which is the period of the lowest priority task (Task 3). During this time, Task 0 will execute 6 times, Task 1 will execute 2 times, and Task 2 will execute once, before Task 3 gets a chance to execute its 2 units of work. Task 3 Deadline Miss: Task 3, having the lowest priority, can potentially miss its deadline due to the following scenario:
Task 0, 1, and 2 collectively require
$6 \cdot 1 + 2 \cdot 1 + 1 \cdot 1 = 9$,
9 units of time within the first 13-time units, excluding overheads and assuming they preempt Task 3 whenever they become ready.

Task 3 requires 2 consecutive units of time to execute. However, due to its lower priority, its execution can be delayed by the execution of higher-priority tasks. If any of the higher-priority tasks become ready during Task 3's execution window, Task 3 will be preempted, potentially leading to a situation where it cannot complete its 2 units of execution before its deadline at time 13.

Observation
The deadline miss for Task 3 under RM scheduling arises because RM does not account for the actual deadlines when assigning priorities; it only considers the periods. This approach can lead to inefficiencies when lower-priority tasks have long execution times or when the distribution of task executions leads to higher-priority tasks preempting lower-priority tasks close to their deadlines.

Why EDF and LLF Do Not Miss Deadlines
EDF: Prioritizes tasks based on their absolute deadlines. Even if Task 3 has a longer period, as its deadline approaches, it would gain priority over other tasks, ensuring it gets CPU time to meet its deadline. LLF: Prioritizes tasks based on their remaining laxity (time until deadline minus remaining execution time). As Task 3's deadline approaches without it having executed, its laxity decreases and I executes first instead of missing the deadline.

5 Question 5

Q: [30 points] Read Chapter 3 of the textbook.

(a) **Q: Briefly describe and 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. Describe whether you think each is reasonable for actual practice or whether you think each is only applicable to and idealized model of practice.**

Answer:

The Liu and Layland paper, a seminal work in real-time systems, introduces the Rate Monotonic (RM) scheduling algorithm and its Least Upper Bound (LUB) for CPU utilization. In this context, certain constraints and assumptions are made for the derivation of the LUB, and understanding these is crucial for applying the theory to practical scenarios.

i. **Constraints in the RM LUB Derivation**

A. **Fixed Task Set with Harmonic Periods** The derivation initially considers only two tasks, under the assumption that if the LUB can be proven for two tasks, it extends to any number of tasks. This simplification assumes a fixed task set, which might not always be the case in dynamic systems where tasks can vary over time.

B. **Worst-Case Execution Time (WCET) Overlap:** It's assumed that the WCET does not perfectly fit within the task periods, allowing for some overlap. This constraint is realistic as it accounts for the variability in execution times and the potential for tasks to run longer than their allocated time slots.

C. **Minimum Interference (I=1):** The derivation assumes that the least upper bound for utilization occurs when higher-priority tasks interfere with lower-priority tasks by the minimum amount possible, exactly once. This assumes a very idealized scenario where task executions are perfectly aligned to minimize interference while still adhering to priority constraints.

ii. **Assumptions in the RM LUB Derivation**

A. **Task Independence:** Tasks are assumed to be independent of each other, with no synchronization or shared resources. This assumption simplifies the analysis but might not hold in complex systems where tasks often interact or share resources.

B. **Static Priority Scheduling:** RM scheduling assigns static priorities based on task periods, with shorter periods implying higher priorities. This assumption neglects the potential need for dynamic priority adjustments in response to runtime conditions.

C. **Complete Utilization of CPU:** The derivation assumes that the system aims to maximize CPU utilization without overloading the processor. In practice, this might not always be desirable as systems might prioritize responsiveness or power efficiency over maximizing utilization.

The LUB for utility increases as the potential for task schedulability under RM becomes more constrained. When I is minimized, it suggests an optimal frequency of interference that allows for the highest possible utility without violating the schedulability constraints of RM. In other words, minimizing I maximizes the efficiency of task scheduling under RM, allowing for the highest total utilization (or LUB) where tasks can still be guaranteed to meet their deadlines.

iii. Applicability to Practice

A. **Fixed Task Set and WCET Overlap:** These constraints are relatively reasonable in systems where the task set is known a priori, and tasks have been analyzed for their worst-case execution scenarios. However, dynamic systems where tasks can change or where execution times are highly variable might not fit well within these constraints.

B. **Minimum Interference:** While aiming for minimal interference is ideal, in practice, the actual interference pattern can be much more complex due to the unpredictable nature of task executions and system load. This assumption serves more as a theoretical ideal than a practical expectation.

C. **Independence and Static Priority:** These assumptions simplify analysis but might not reflect the complexity of modern systems where task interactions and the need for adaptability can't be overlooked. Static priorities might not suffice in systems requiring more flexible scheduling approaches to respond to changing conditions.

(b) **Q: 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.r deterministic and compare your results.**

Answer: Case 1 represents that the utilization monotonically decreases with increasing C1 when T1>T2. Case 2 represents that the utilization monotonically increases with increasing C1 when T1>T2. RM policy assumes that T2 > T1.

i. In the derivation, we want to determine the utility when C1 exceeds or is less than the critical time zone. It is mentioned that is determined by subtracting case 1 and case 2 utility data with varying T2 and C1. Exactly how it is achieved was not clear. According to our understanding, derivation is a mathematical approach for studying and understanding the relations between the parameters in utility for different situations. Finding the in which the system can function at the same degree of utility is the aim, since it offers important information about whether real-time scheduling is feasible or not.

ii. It is mentioned that the cases are valid only when services intersect and this can occur only when C1 is equal. If C1 is not equal, it implies that there is no common critical instant where the utility expressions for Case 1 and Case 2 intersect. In such a scenario, the analysis of the utility equation might not yield a valid LUB for the system's utility. It was difficult to understand this case.

iii. The derivation also assumes that the least upper bound for utilization occurs when higher-priority tasks interfere with lower-priority tasks by the minimum amount possible, i.e. exactly one. Assumption made was unclear. The assumption is that the smallest possible value for I is 1, indicating that there is exactly one instance of S1 occurring during the execution of S2. This assumption simplifies the analysis and allows for the derivation of the least upper bound for utilization.

6 Challenges Faced

1. Understanding the key assumptions made in the derivation and the reason why these assumptions are necessary was difficult to understand.

2. Learning cheddar tool and getting used to it was initially challenging but later on was fun.

7 Conclusion

1. We understood the concept of the Cyclic Executive and its comparison with Linux POSIX RT threading and RTOS.

2. Implemented and analyzed custom feasibility test code for different scheduling policies (RM, EDF, LLF) using Cheddar.

3. Moreover, learned the constraints, assumptions, and derivation steps in Rate Monotonic (RM) Least Upper Bound (LUB) as mentioned in Chapter 3 of the textbook.

8 References

1. ECEN 5623 Lecture slides material and example codes.
2. REAL-TIME EMBEDDED COMPONENTS AND SYSTEMS with LINUX and RTOS, Sam Siewert John Pratt (Chapter 3, 4 & 5).
3. Exercise 2 requirements included links and documentation.

Appendices

A C Code for the Implementation

feasibility_tests.c

```c
/*****************************************************************************
 * Copyright (C) 2023 by Sam Siewert, Parth Thakkar
 *
 * Redistribution, modification or use of this software in source or binary
 * forms is permitted as long as the files maintain this copyright. Users are
 * permitted to modify this and use it to learn about the field of embedded
 * software. Parth Thakkar and the University of Colorado are not liable for
 * any misuse of this material.
 *
*****************************************************************************/

/**
 * @file    feasibility_tests.c
 * @brief  This example code provides feasibiltiy decision tests for single core fixed
 * priority rate mononTic systems only (not dyanmic priority such as deadline
 * driven EDF and LLF). These are standard algorithms which either estimate
 * feasibility (as the RM LUB does) or automate exact analysis (scheduling point,
 * completion test) for a set services sharing one CPU core. This can be emulated on Linux SMP
 * multi-core systemes by use of POSIX thread affinity, to "pin" a thread to a
 * specific core.  Coded based upon standard definition of:
 * 1) RM LUB based upon model by Liu and Layland
 * 2) Scheduling Point - an exact feasibility algorithm based upon Lehoczky, Sha, and Ding exact analysis
 * 3) Completion Test - an exact feasibility algorithm
 *
 *
 * @author  Parth Thakkar, Sam Siewert
 * @date    20th Sept 2023
 *
 */

#include <math.h>
#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>

#define TRUE 1
#define FALSE 0

#define U32_T unsigned int
#define EXAMPLES 10

#define MAX_TASKS 4

typedef struct
{
    U32_T wcet[MAX_TASKS];
    U32_T period[MAX_TASKS];
    U32_T deadline[MAX_TASKS];
    U32_T num_tasks;
} task_set_t;

// Example tasks
```

```
52
53  task_set_t tasks[EXAMPLES] = {
54      {
55          // 0
56          .period = {2, 10, 15},   // Example Periods
57          .wcet = {1, 1, 2},       // Example Worst Case Execution Times
58          .deadline = {2, 10, 15}, // Example Deadlines
59          .num_tasks = 3           // Number of tasks in this set
60      },
61
62      {
63          // 1
64          .period = {2, 5, 7},   // Example Periods
65          .wcet = {1, 1, 2},     // Example Worst Case Execution Times
66          .deadline = {2, 5, 7}, // Example Deadlines
67          .num_tasks = 3         // Number of tasks in this set
68      },
69
70      {
71          // 2
72          .period = {2, 5, 7, 13},   // Example Periods
73          .wcet = {1, 1, 1, 2},      // Example Worst Case Execution Times
74          .deadline = {2, 5, 7, 13}, // Example Deadlines
75          .num_tasks = 4             // Number of tasks in this set
76      },
77
78      {
79          // 3
80          .period = {3, 5, 15},   // Example Periods
81          .wcet = {1, 2, 3},      // Example Worst Case Execution Times
82          .deadline = {3, 5, 15}, // Example Deadlines
83          .num_tasks = 3          // Number of tasks in this set
84      },
85
86      {
87          // 4
88          .period = {2, 4, 16},   // Example Periods
89          .wcet = {1, 1, 4},      // Example Worst Case Execution Times
90          .deadline = {2, 4, 16}, // Example Deadlines
91          .num_tasks = 3          // Number of tasks in this set
92      },
93      {
94          // 5
95          .period = {2, 4, 16},   // Example Periods
96          .wcet = {1, 1, 4},      // Example Worst Case Execution Times
97          .deadline = {2, 4, 16}, // Example Deadlines
98          .num_tasks = 3          // Number of tasks in this set
99      },
100
101     {
102         // 6
103         .period = {2, 5, 7, 13},   // Example Periods
104         .wcet = {1, 1, 1, 2},      // Example Worst Case Execution Times
105         .deadline = {2, 3, 7, 15}, // Example Deadlines
106         .num_tasks = 4             // Number of tasks in this set
107     },
108
109     {
110         // 7
```

```c
        .period = {3, 5, 15},   // Example Periods
        .wcet = {1, 2, 4},      // Example Worst Case Execution Times
        .deadline = {3, 5, 15}, // Example Deadlines
        .num_tasks = 3          // Number of tasks in this set
    },

    {
        // 8
        .period = {2, 5, 7, 13},   // Example Periods
        .wcet = {1, 1, 1, 2},      // Example Worst Case Execution Times
        .deadline = {2, 5, 7, 13}, // Example Deadlines
        .num_tasks = 4             // Number of tasks in this set
    },
    {
        // 9
        .period = {6, 8, 12, 24},   // Example Periods
        .wcet = {1, 2, 4, 6},       // Example Worst Case Execution Times
        .deadline = {6, 8, 12, 24}, // Example Deadlines
        .num_tasks = 4              // Number of tasks in this set
    }};

// Feasibility test functions
bool completion_time_feasibility(task_set_t *task_set);
bool scheduling_point_feasibility(task_set_t *task_set);
bool rate_monotonic_least_upper_bound(task_set_t *task_set);
int edf_feasibility(task_set_t *task_set, bool deadline);
int llf_feasibility(task_set_t *task_set, bool deadline);
bool deadline_monotonic_feasibility(task_set_t *task_set);

int main(void)
{

    for (int i = 0; i < EXAMPLES; i++)
    {
        printf("\n****************\n");
        printf("Example %d\n", i);

        // Calculate and print total utilization
        double U = 0.0;
        for (int j = 0; j < tasks[i].num_tasks; j++)
        {
            U += (double)tasks[i].wcet[j] / tasks[i].period[j];
        }
        printf("C: ");
        for (int j = 0; j < tasks[i].num_tasks; j++)
        {
            printf("%d ", tasks[i].wcet[j]);
        }
        printf("\nT: ");
        for (int j = 0; j < tasks[i].num_tasks; j++)
        {
            printf("%d ", tasks[i].period[j]);
            if(tasks[i].period[j] ==0){
                printf("Pseriod is zero\n");
                exit(0);
            }
        }
        printf("\nD: ");
        int dm = 0;
```

```c
        for (int j = 0; j < tasks[i].num_tasks; j++)
        {
            printf("%d ", tasks[i].deadline[j]);
            if(tasks[i].deadline[j] ==0){
                printf("Deadline is zero\n");
                exit(0);
            }
            if (tasks[i].deadline[j] != tasks[i].period[j])
            {
                dm++;
            }
        }
        // printf("\nUtility : %4.2f%%\n", U * 100);

        // Perform and print feasibility tests
        printf("RM LUB: %s\n", rate_monotonic_least_upper_bound(&tasks[i]) ? "Feasible" : "Infeasible");
        printf("Completion time feasibility: %s\n", completion_time_feasibility(&tasks[i]) ? "Feasible" : "Infeasible");
        printf("Scheduling point feasibility: %s\n", scheduling_point_feasibility(&tasks[i]) ? "Feasible" : "Infeasible");

        if (dm != 0)
        {
            printf("Deadline monotonic: %s\n", deadline_monotonic_feasibility(&tasks[i]) ? "Feasible" : "Infeasible");

            printf("\n(Period)");
            printf("EDF on Period: %s\n", edf_feasibility(&tasks[i], false) ? "Feasible" : "Infeasible");
            printf("LLF on Period: %s\n", llf_feasibility(&tasks[i], false) ? "Feasible" : "Infeasible");

            printf("\n(Deadline)");
            printf("EDF on Deadline: %s\n", edf_feasibility(&tasks[i], true) ? "Feasible" : "Infeasible");
            printf("LLF on Deadline: %s\n", llf_feasibility(&tasks[i], true) ? "Feasible" : "Infeasible");
        }
        else if (i > 4)
        {
            printf("\n(Period)");
            printf("EDF: %s\n", edf_feasibility(&tasks[i], false) ? "Feasible" : "Infeasible");
            printf("LLF: %s\n", llf_feasibility(&tasks[i], false) ? "Feasible" : "Infeasible");
        }

        // Add other feasibility tests here

        printf("\n");
    }
}

bool rate_monotonic_least_upper_bound(task_set_t *task_set)
{
    double utility_sum = 0.0, lub = 0.0;
    int idx;

    // Sum the C(i) over the T(i) for utility calculation
    printf("\n\n");
```

```c
    for (idx = 0; idx < task_set->num_tasks; idx++)
    {
        utility_sum += ((double)task_set->wcet[idx] / (double)task_set->period[idx]);
        printf("Task %d, WCET=%u, Period=%u, Utility Sum = %lf\n", idx, task_set->wcet[idx], task_set->period[idx], utility_sum);
    }
    printf("\nTotal Utility Sum = %lf\n", utility_sum);

    // Compute LUB for the number of services
    lub = (double)task_set->num_tasks * ((pow(2.0, (1.0 / (double)task_set->num_tasks))) - 1.0);
    printf("LUB = %lf\n", lub);

    // Compare the utility sum to the bound and return feasibility
    if (utility_sum <= lub)
        return TRUE;
    else
        return FALSE;
}

bool completion_time_feasibility(task_set_t *task_set)
{
    int i, j;
    U32_T an, anext;
    int set_feasible = TRUE;

    // For all tasks in the analysis
    for (i = 0; i < task_set->num_tasks; i++)
    {
        an = 0;
        anext = 0;

        for (j = 0; j <= i; j++)
        {
            an += task_set->wcet[j];
        }

        while (1)
        {
            anext = task_set->wcet[i];

            for (j = 0; j < i; j++)
                anext += ceil((double)an / (double)task_set->period[j]) * task_set->wcet[j];

            if (anext == an)
                break;
            else
                an = anext;
        }

        if (an > task_set->period[i])
        {
            set_feasible = FALSE;
        }
    }

    return set_feasible;
}
```

```
277
278 bool scheduling_point_feasibility(task_set_t *task_set)
279 {
280     int rc = TRUE, i, j, k, l, status, temp;
281
282     // For all tasks in the analysis
283     for (i = 0; i < task_set->num_tasks; i++)
284     { // iterate from highest to lowest priority
285         status = 0;
286
287         // Look for all available CPU minus what has been used by higher
    priority tasks
288         for (k = 0; k <= i; k++)
289         {
290             // find available CPU windows and take them
291             for (l = 1; l <= (floor((double)task_set->period[i] / (double)
    task_set->period[k])); l++)
292             {
293                 temp = 0;
294
295                 for (j = 0; j <= i; j++)
296                     temp += task_set->wcet[j] * ceil((double)l * (double)
    task_set->period[k] / (double)task_set->period[j]);
297
298                 // Can we get the CPU we need or not?
299                 if (temp <= (l * task_set->period[k]))
300                 {
301                     // insufficient CPU during our period, therefore infeasible
302                     status = 1;
303                     break;
304                 }
305             }
306             if (status)
307                 break;
308         }
309
310         if (!status)
311             rc = FALSE;
312     }
313     return rc;
314 }
315
316 int llf_feasibility(task_set_t *task_set, bool deadline)
317 {
318     double totalU = 0.0;
319     if (!deadline)
320     {
321         for (int i = 0; i < task_set->num_tasks; i++)
322         {
323             totalU += (double)task_set->wcet[i] / task_set->period[i];
324         }
325     }
326     else
327     {
328         for (int i = 0; i < task_set->num_tasks; i++)
329         {
330             totalU += (double)task_set->wcet[i] / task_set->deadline[i];
331         }
332     }
```

```c
    printf("Total utility in LLF: %f ", totalU);
    if (totalU <= 1.0)
    {
        printf("Which is less than 1.0 \n");
        return TRUE;
    }
    else
    {
        printf("Which is less than 1.0 \n");
        return FALSE;
    }
}

bool deadline_monotonic_feasibility(task_set_t *task_set)
{
    //Ensure tasks are sorted by their deadlines before running this feasibility test.
    int status = 0;
    for (int i = 0; i < task_set->num_tasks; i++)
    {
        float interference = 0;
        float utilization = 0;
        for (int j = 0; j < i; j++)
        {
            interference += (ceil((float)task_set->deadline[i] / (float)task_set->period[j])) * (float)task_set->wcet[j];
        }
        utilization = ((float)(task_set->wcet[i]) / (float)task_set->deadline[i]) + (interference / (float)task_set->deadline[i]);
        if (utilization > 1)
        {
            status = 1;
            break;
        }
    }
    if (status == 1)
        return FALSE;
    else
        return TRUE;
}

int edf_feasibility(task_set_t *task_set, bool deadline)
{
    double totalU = 0.0;
    if (!deadline)
    {
        for (int i = 0; i < task_set->num_tasks; i++)
        {
            totalU += (double)task_set->wcet[i] / task_set->period[i];
        }
    }
    else
    {
        for (int i = 0; i < task_set->num_tasks; i++)
        {
            totalU += (double)task_set->wcet[i] / task_set->deadline[i];
        }
    }
    printf("\nTotal utility in EDF: %f ", totalU);
```

```
389     if (totalU <= 1.0)
390     {
391         printf("Which is less than 1.0 \n");
392         return TRUE;
393     }
394     else
395     {
396         printf("Which is less than 1.0 \n");
397         return FALSE;
398     }
399 }
400
```