ECEN 5623 – Real Time Embedded Systems

Exercise 2

24th February 2024

Team Members: Chandana Challa, Jithendra Halenahalli Somashekaraiah

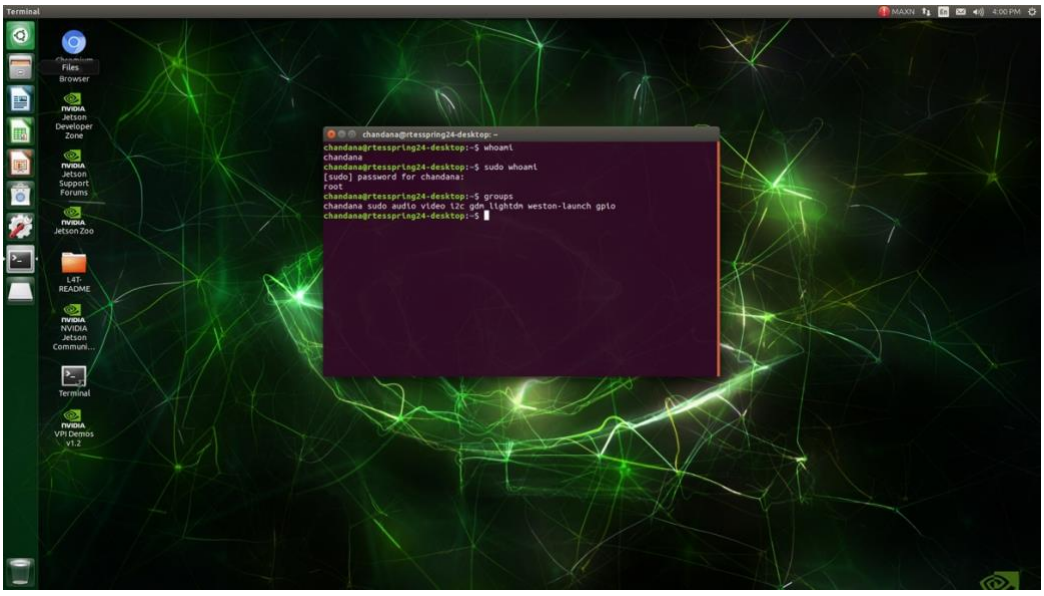
1. [5 points] make yourself an account on your Dev Kit.

a) 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 (you can normally do "sudo apt-get install nano" or any alternative editor you like best). You are welcome to use whatever editor works best for you in this class. Finally, you'll want an SSH tool (e.g. MobaXterm for Windows) and you'll want to enable SSH via the GUI or with headless methods. Do a quick "sudo whoami" to demonstrate success.

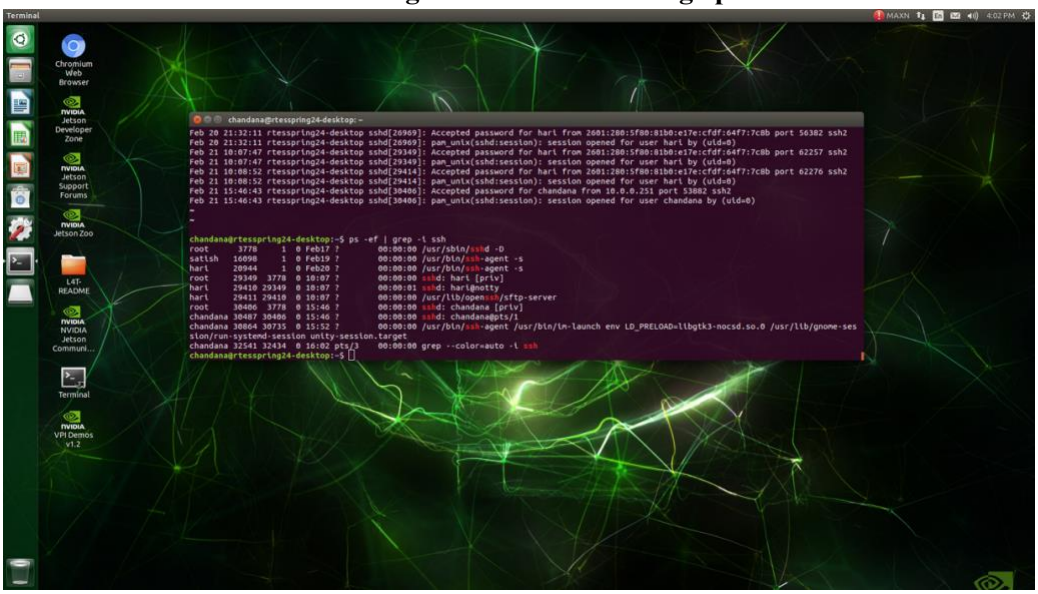
Answer:

Create user account on Jetson Nano and able to enable ssh service. We are able to use vi, vim editors from the terminal and able to edit the files.

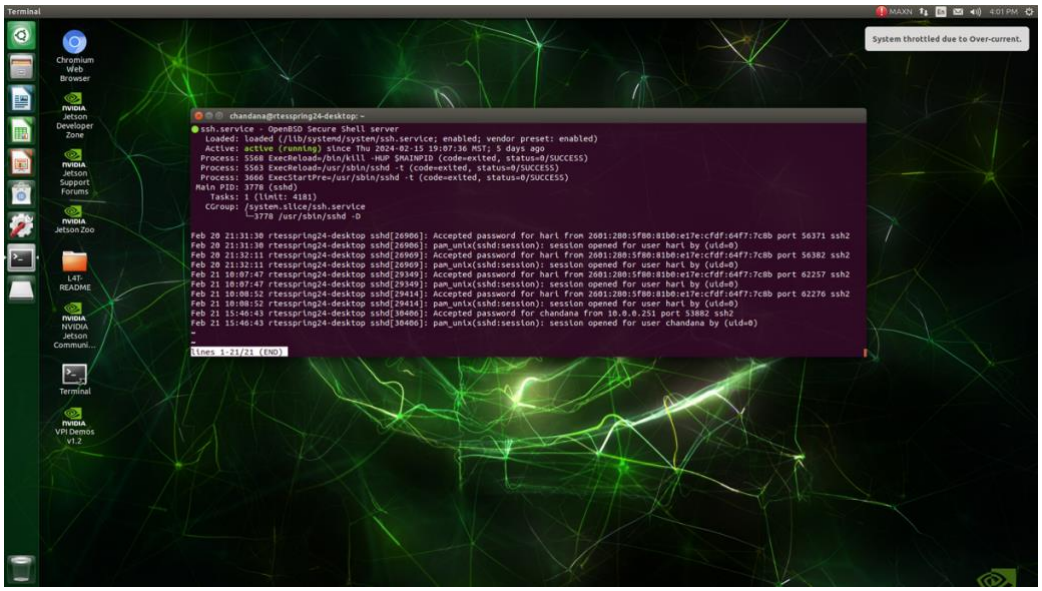
Screenshot of "whoami" command:



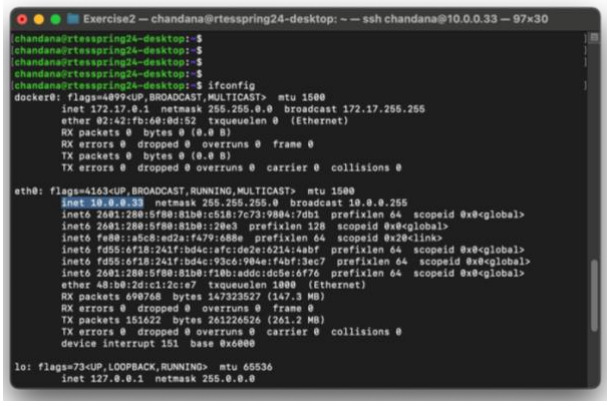
Screenshot of ssh service daemon running on Jetson Nano using "ps" command:



Screenshot of ssh service status running on Jetson Nano using "systemctl" command: sudo systemctl status ssh



Screenshot of remote ssh from laptop:



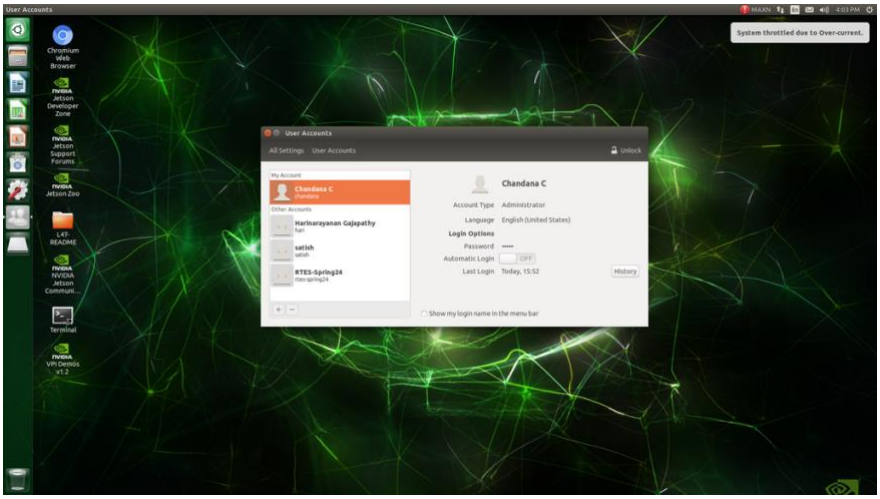
b) Logout of Linux and test your login, then logout. Use Alt+Print-Screen to capture your desktop and save as proof you set up your account. Note that you can always get a terminal with Ctrl+Alt+t key combination. If you don't like the desktop, you can try "GNOME Flashback" and please play around with customizing your account as you wish. Show evidence that you have created a custom login with screenshots or photos from your phone.

Answer:

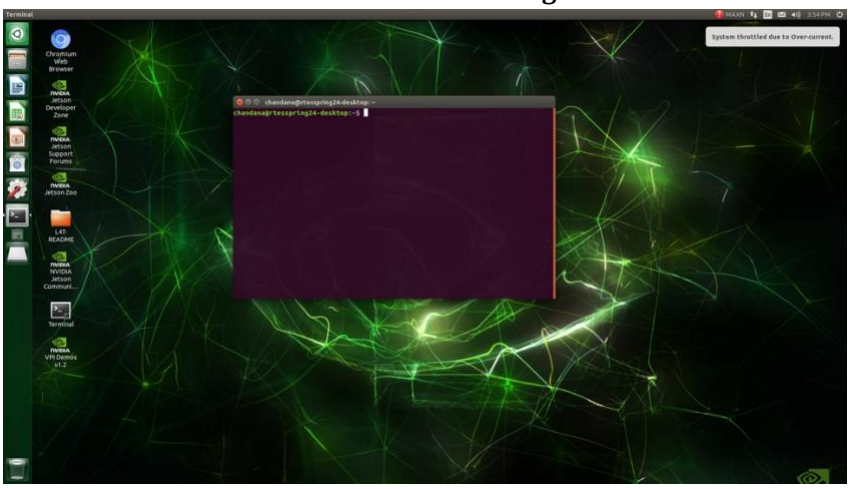
Created custom login account and captured screenshots as mentioned.

Screenshot of custom user account "Chandana C":

Screenshot of custom user account in system settings:



Screenshot of custom user account terminal after login:

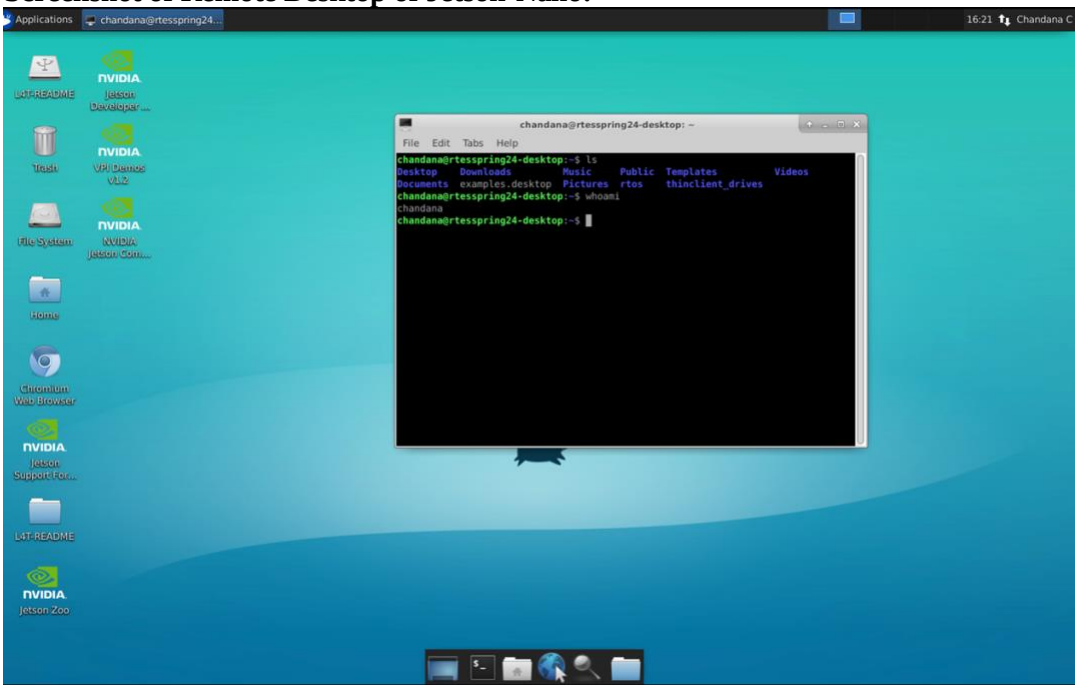


c) Make sure you can access graphical tools with MobaXterm or VNC and show that tools you may need in the future work, including a tool to display graphics in PPM or PGM format. Overall, make sure you are comfortable with development, debug, and compiler tools either native or cross-development and document and demonstrate that you know them.

Answer:

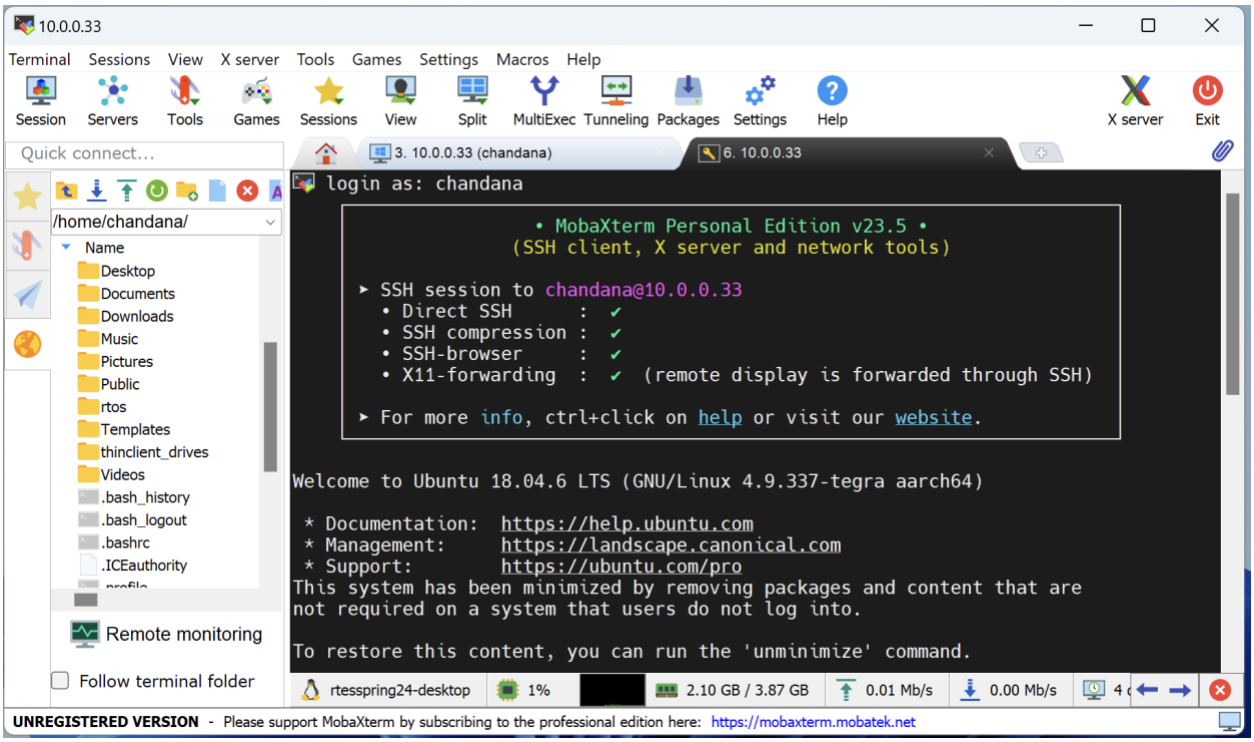
We are able to access Jetson Nano board using Remote Desktop Application from remote device using ip address and user account.

Screenshot of Remote Desktop of Jetson-Nano:

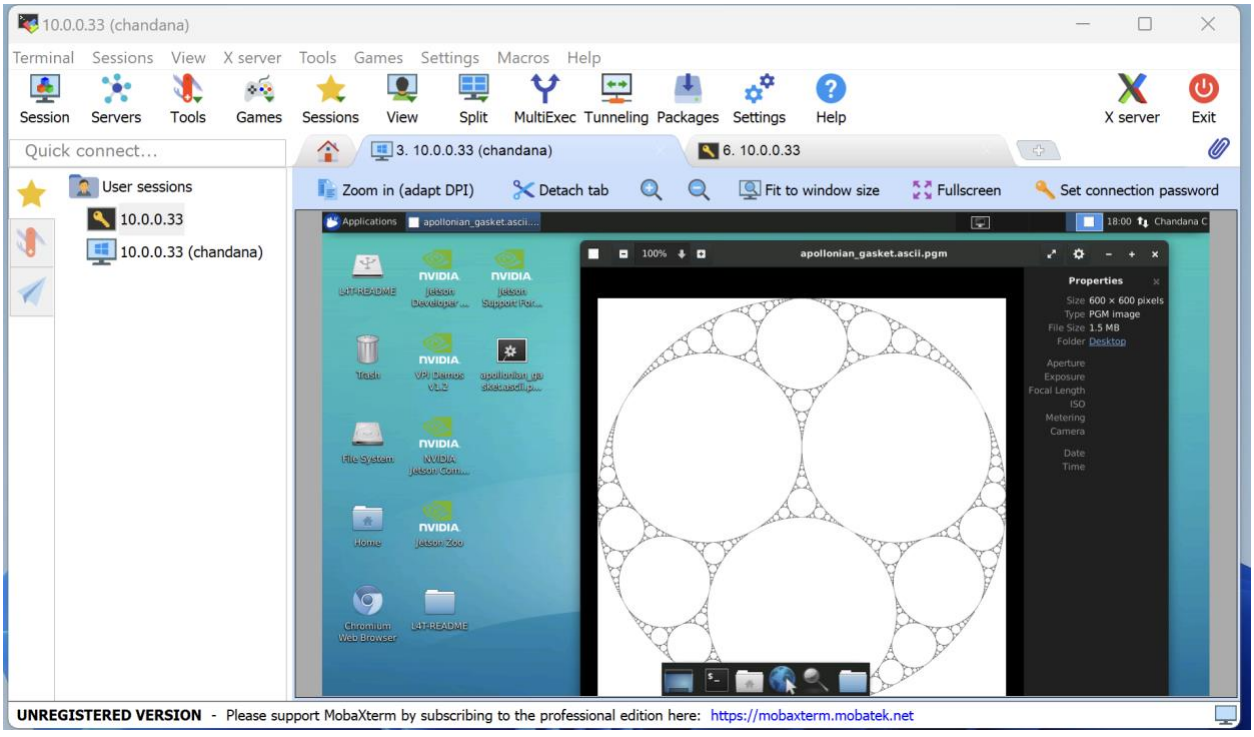


Downloaded MobaXterm on Windows and added two new sessions to access Jetson Nano using SSH and RDP. Able to access folders and open images remotely in SSH session. In RDP session, GUI is accessible and able to open the PGM image on Jetson Nano board.

Screenshot of MobaXterm SSH session:



Screenshot of MobaXterm RDP session:



We are able to access GUI applications from remote desktop and also able to perform ssh using MobaXterm. We are comfortable using IDEs, cross-compilation, native compilation and debugging issues on Jetson Nano board.

2. [10 points] Read the paper "Architecture of the Space Shuttle Primary Avionics Software System" [also available on Canvas, "shuttle_paper.pdf"], by Gene Carlow.

a) Provide an explanation and critique of the frequency executive architecture.

Answer:

Summary of paper:

The software, developed based on contemporary software engineering concepts, followed a top-down integration method, and delivered software incrementally. This software design features critical timing and redundancy management, common in real-time systems. To accommodates eight operation sequences, each comprising major modes, specialist functions, and display functions, they came up with few implementations:

- To address memory constraints, mass memory is utilized to store all the code, loading only the required code to the general-purpose computer main memory.
- Flight control OS manages processes and I/O management and uses redundant processors for data processing. It initializes and configures the DPS and synchronizes the data network.
- User interface to support processed outputs from OPS to display on CRT displays and take keyboard input to support.
- To serve three applications: guidance-navigation-control, system management, and vehicle checkout, executed in cyclic loops with tight timings and phase relationships.
- Critical tasks like GN&C execute at 25Hz, while low-priority tasks like display update at 0.25Hz.

The successful implementation of PASS software is credited to the structural aspects of the software architecture and HAL/S language, alongside interface standardization through FCOS SVCs and control segment grammar. Establishing this software architecture early in the development cycle provided them the insight into potential GPC capacity requirements.

Frequency Executive Architecture:

Frequency executive software architectures reflect a synchronous design approach within which the dispatching of each application process is timed to always occur at a specific point relative to the start of an overall system cycle or loop. To implement this architecture a lot of care must be taken in the implementation to synchronize the start and completion of the different application processes, and their associated I/O, to prevent overruns at both the process and system levels. (Source: "Architecture of the Space Shuttle Primary Avionics Software System")

b) 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:

(Source: "Architecture of the Space Shuttle Primary Avionics Software System")

Advantages:

Deterministic:

Tasks are scheduled to execute at fixed intervals, providing deterministic timing behavior. This predictability is crucial for meeting strict timing constraints in real-time systems.

Simple:

The architecture is relatively simple to understand and implement, making it suitable for applications with well-defined timing requirements and limited complexity.

Latency:

Tasks can be executed with low latency, as there is no need for complex scheduling algorithms or context switching overhead.

Repeatability:
In frequency executive, all the tasks are scheduled in a consistent and predictable way at fixed intervals and runs in a fixed sequence.
Low overhead:
Tasks have minimal context changes and overhead is low when compared to real-time threading which involves context switch latencies with scheduling.

Disadvantages:
Flexibility:
The fixed-frequency scheduling approach may not be suitable for applications with dynamic or unpredictable timing requirements. Tasks are executed at predetermined intervals, which may not always align with the changing needs of the system. It is hard to tune and restructure the code.
Scalability:
Managing a large number of tasks with different timing requirements within a fixed-frequency framework can be challenging and may lead to scheduling conflicts or inefficiencies.
Adaptability:
The architecture may struggle to adapt to varying workload conditions or prioritize tasks based on their criticality. In environments where task priorities or timing constraints change dynamically, the rigid nature of the Frequency Executive approach may limit the system's ability to respond effectively.

3. **[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:

Summary of paper:

In safety-critical scenarios, synchronous architectures like the cyclic executive offer distinct advantages by ensuring predictable system behavior without arbitrary interleaving of operations, simplifying testing procedures. Operating independently of an operating system or runtime support, cyclic executives eliminate complexities and incremental development challenges. Nonetheless, reusable software components such as Ada95 and C++ provide robust frameworks for developing these systems, emphasizing portability and fault tolerance while incorporating error handling mechanisms.

For safety mechanisms, the system should prioritize portability and fault tolerance, incorporating robust error handling methods. In case of discrepancies, the process should abort and restart to maintain system integrity. Implementing exceptions is essential for handling error and anomalous situations effectively.

The generic architecture of a cyclic executive consists of structures, executive modules, and shells, decomposing the system into object classes that adhere to the principles of abstraction, encapsulation, and inheritance. So, this modular approach facilitates code reuse and enhances maintainability.

Concept of Cyclic Executive:

In cyclic executive, tasks are managed in a synchronous scheduling strategy within a fixed time frame. The scheduler operates in cycles, with major cycles containing multiple minor cycles. Each minor cycle has its own execution time and subdivides into frames. For instance, the main program represents the major cycle, while tasks are encapsulated within minor cycles under the main program's while loop. Each task (minor cycle) consists of several functions (frames), executed in a cyclic order and repeating due to the while loop structure. This structured approach ensures deterministic execution and simplifies system design and management.

Comparison between Cyclic Executive, Linux POSIX and RTOS:

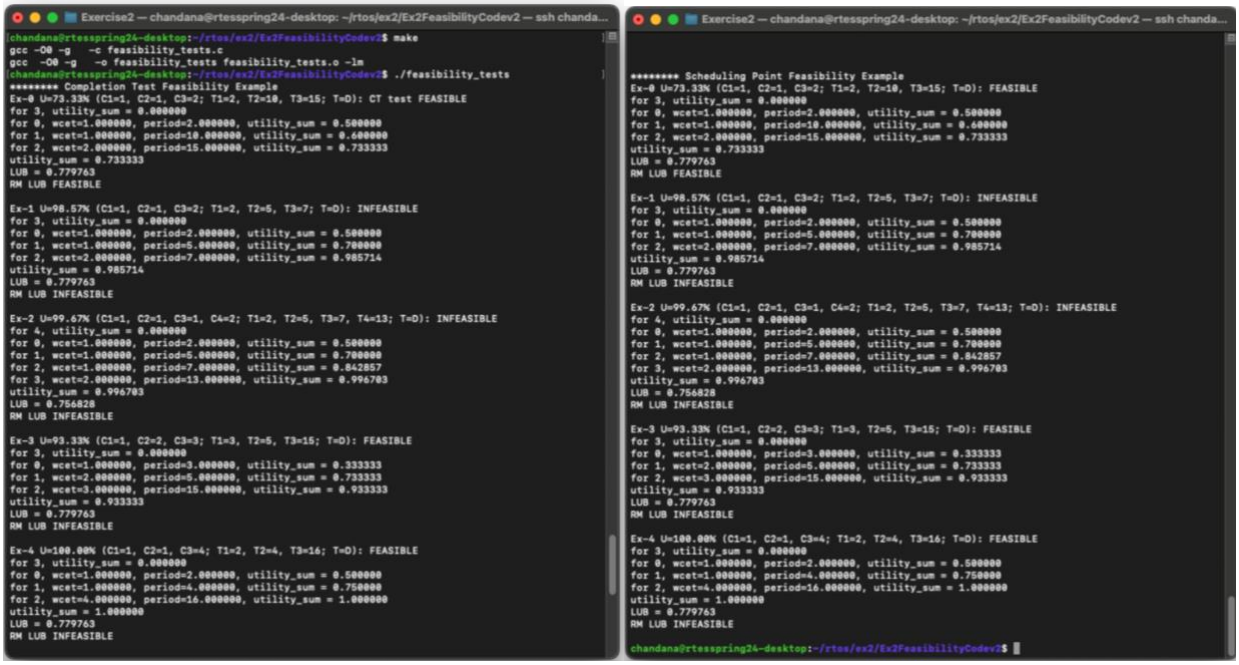
	Cyclic Executive	**Linux POSIX**	**RTOS**
Task execution	Synchronous- Wait for current task to complete before moving on to the next one	Asynchronous- run multiple tasks concurrently	Asynchronous
Deterministic	Timing of task is known and predictable	Not predictable due to frequent context switching	Not predictable due to frequent context switching
Flexibility	Less flexible – adding new tasks will jeopardize the schedular	Supports fixed and dynamic priority according to requirement	Supports fixed and dynamic priority according to requirement
Complexity	Suitable for few tasks based real time system	Can become more complex due to multiple tasks.	This can become more complex due to multiple tasks.
Scalability	low	High	High
Resource useability	Less due to sleeps and pooling implementations	Medium	Provide better memory and CPU utilization

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

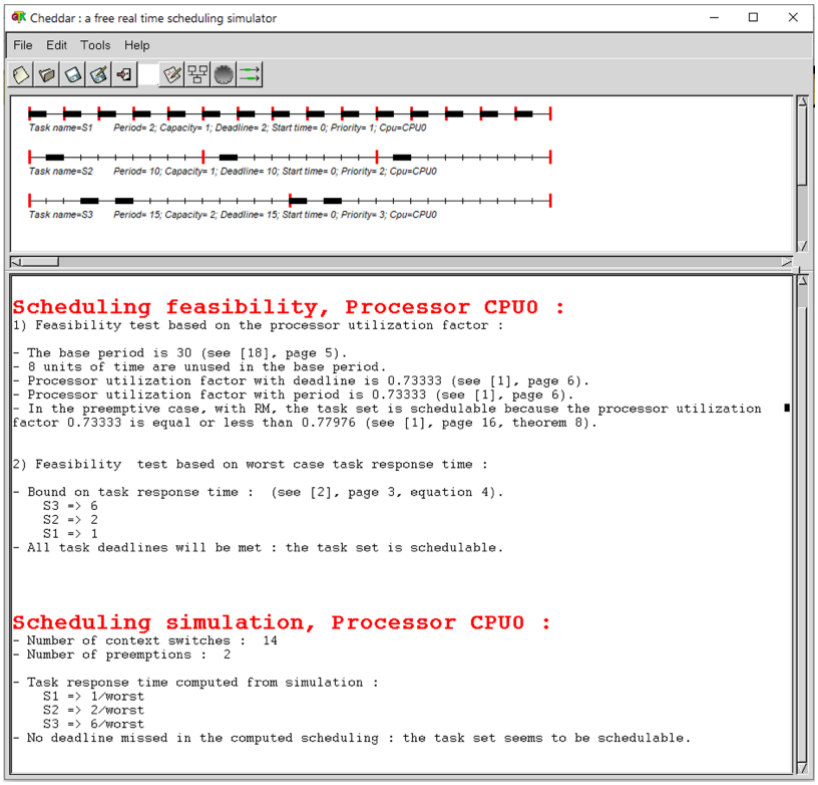
Answer:

Downloaded Feasibility example v2 code provided and built it on Jetson Nano board. Able to resolve warnings and compile it using Makefile.

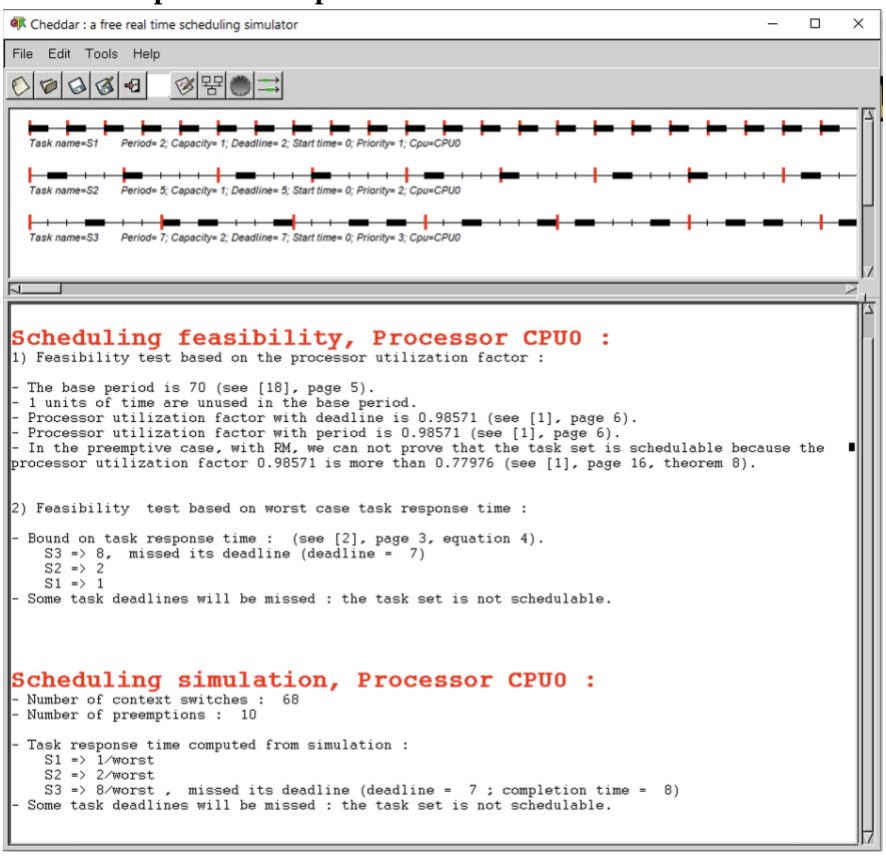
Screenshots of feasibility tests output on Jetson Nano board:



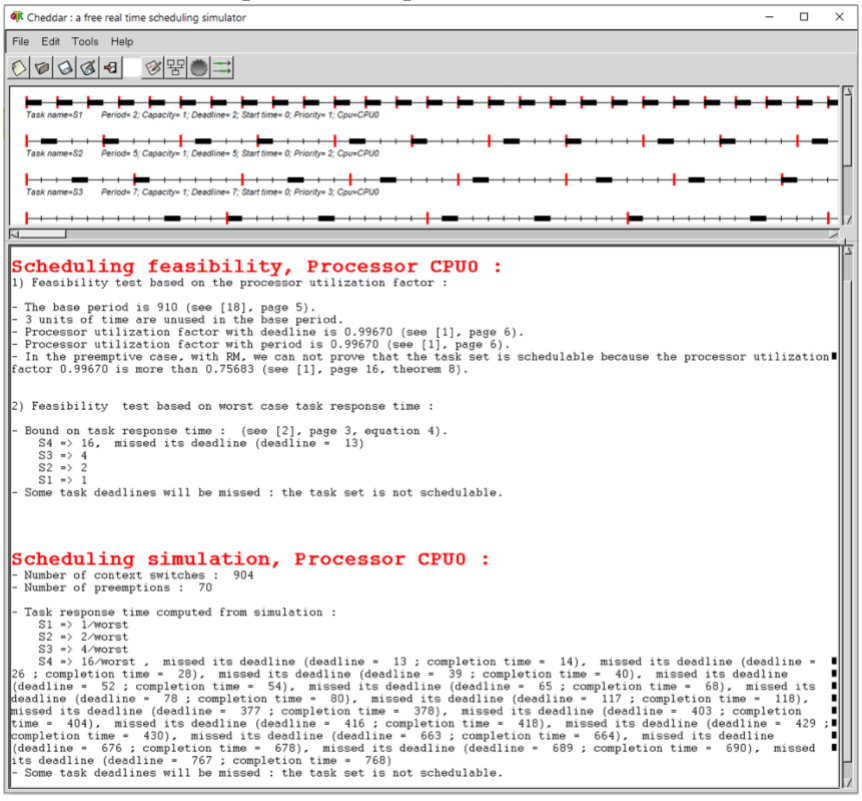
Screenshot of Cheddar output for example-0:



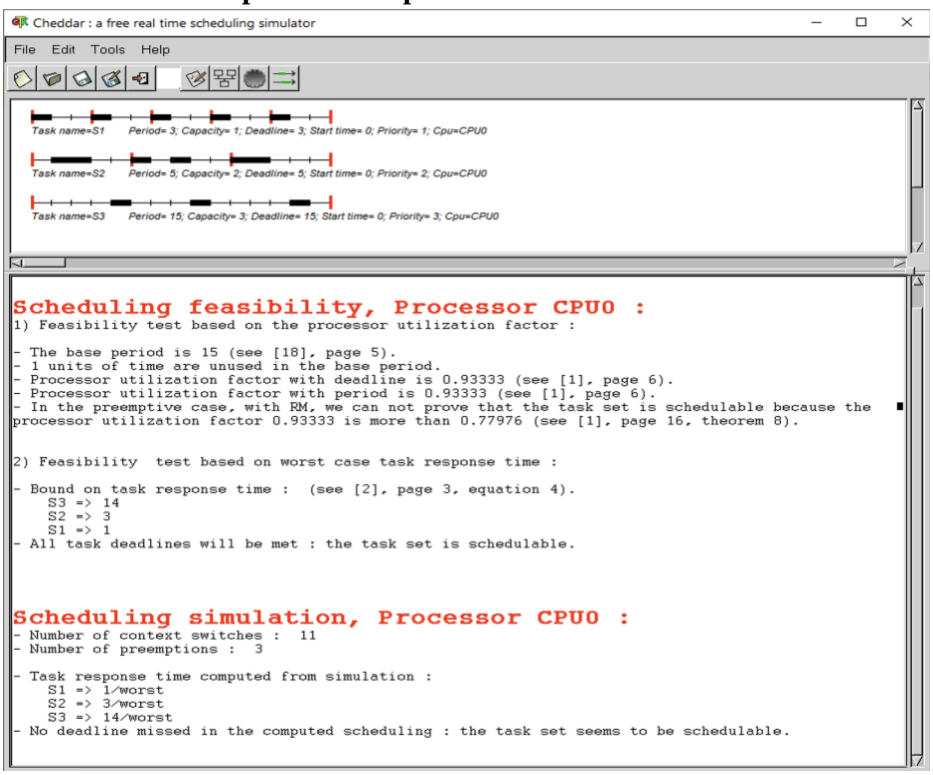
Screenshot of Cheddar output for example-1:



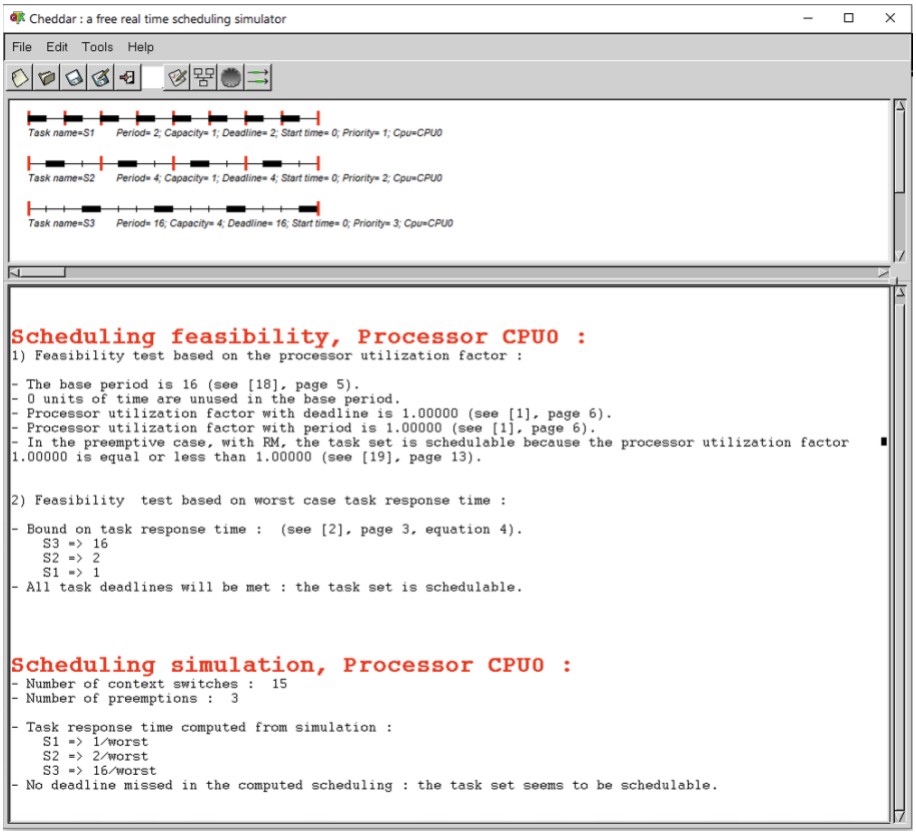
Screenshot of Cheddar output for example-2:



Screenshot of Cheddar output for example-3:



Screenshot of Cheddar output for example-4:



Comparison Analysis:

	Code RM output			Cheddar RM output		
	Completion test	**Scheduling point test**	**RM LUB test**	**Worst case analysis tests (completion/scheduling point tests)**	**RM LUB**	**Simulation**
Example-0	Feasible	Feasible	Feasible	Feasible	Feasible	Feasible
Example-1	Infeasible	Infeasible	Infeasible	Infeasible	Infeasible	Infeasible
Example-2	Infeasible	Infeasible	Infeasible	Infeasible	Infeasible	Infeasible
Example-3	Feasible	Feasible	Infeasible	Feasible	Infeasible	Feasible
Example-4	Feasible	Feasible	Infeasible	Feasible	Feasible	Feasible

In case of example-0, example-1, example-2 and example-3 both the example code RM LUB output results and cheddar results for RM analysis **match**.

In example-3, after considering the simulation outputs and as no deadlines are missed, Cheddar decides RM test is feasible while code only checks RM LUB and decides as infeasible.

The worst-case analysis tests (scheduling point/completion tests) results from example code for example 0-4 **matches** with Cheddar RM output results.

In case of example-4, example code RM LUB output results and cheddar results for RM LUB analysis **does not match** as mentioned above. Cheddar RM LUB test also considered whether **periods are hormonic** and if utilization is 100%, then RM LUB mentions as schedulable. In this case, the periods are harmonic, and utilization is 100%, hence cheddar mentions as feasible, while in code we only check utilization with LUB, hence it mentions as infeasible.

Example code provides the results based only on RM feasibility test formula (RM LUB), while Cheddar provides results by checking both RM feasibility test, worst-case analysis and simulating the schedule over LCM of periods of services and checking the missed deadlines. Hence, I believe Cheddar results are appropriate.

Results:

Example-0 is Feasible
Example-1 is Infeasible
Example-2 is Infeasible
Example-3 is Feasible
Example-4 is Feasible

b) 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:

Implemented Dynamic priority Scheduling (EDF/LLF) and Deadline Monotonic feasibility tests and attached the code in Appendix Section 1.

Dynamic priority Scheduling (EDF/LLF) feasibility test:

This test checks whether utilization is less than or equal to 1 and declares that the schedule is feasible or not.

Deadline Monotonic feasibility test:

In this test, it is considered that deadlines are not equal to the periods of the services. Interference of each service is calculated, and each service is checked whether it is feasible. If all the services are feasible, then the system is considered as feasible.

Worst-case Interference calculation formula:

$$I_i = \sum_{j=1}^{i-1} \left\lceil \frac{D_i}{T_j} \right\rceil C_j$$

Service feasibility test formula (Audsley):

$$\forall i : 1 \le i \le n : \frac{C_i}{D_i} + \frac{I_i}{D_i} \le 1.0$$

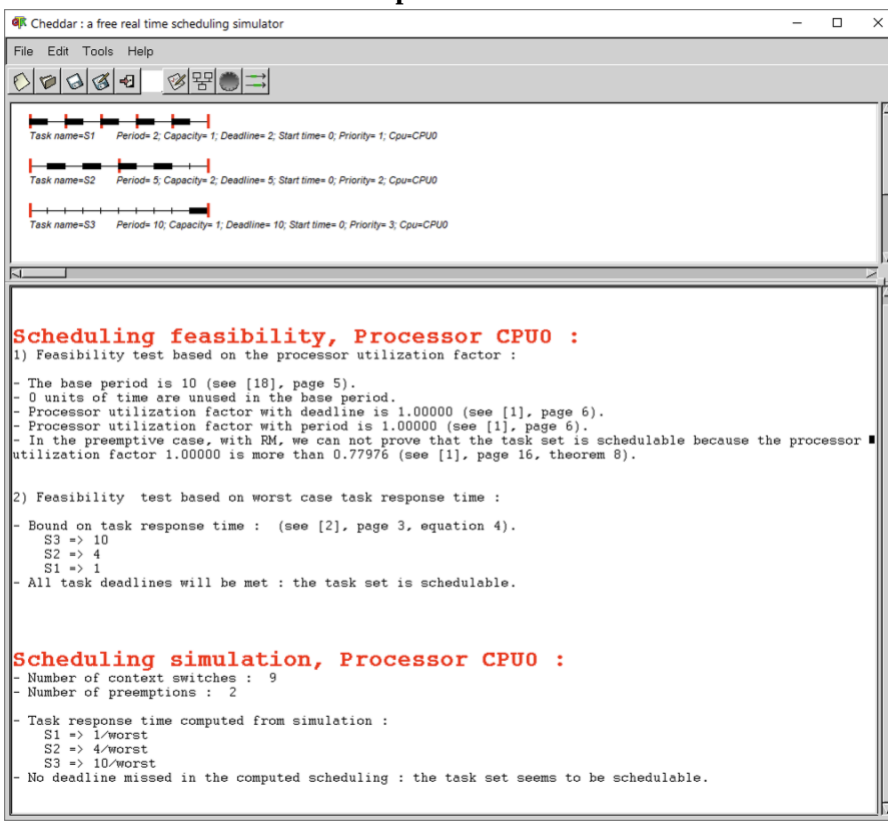
Example-5:

Screenshot of code RM, EDF, LLF feasibility tests output:

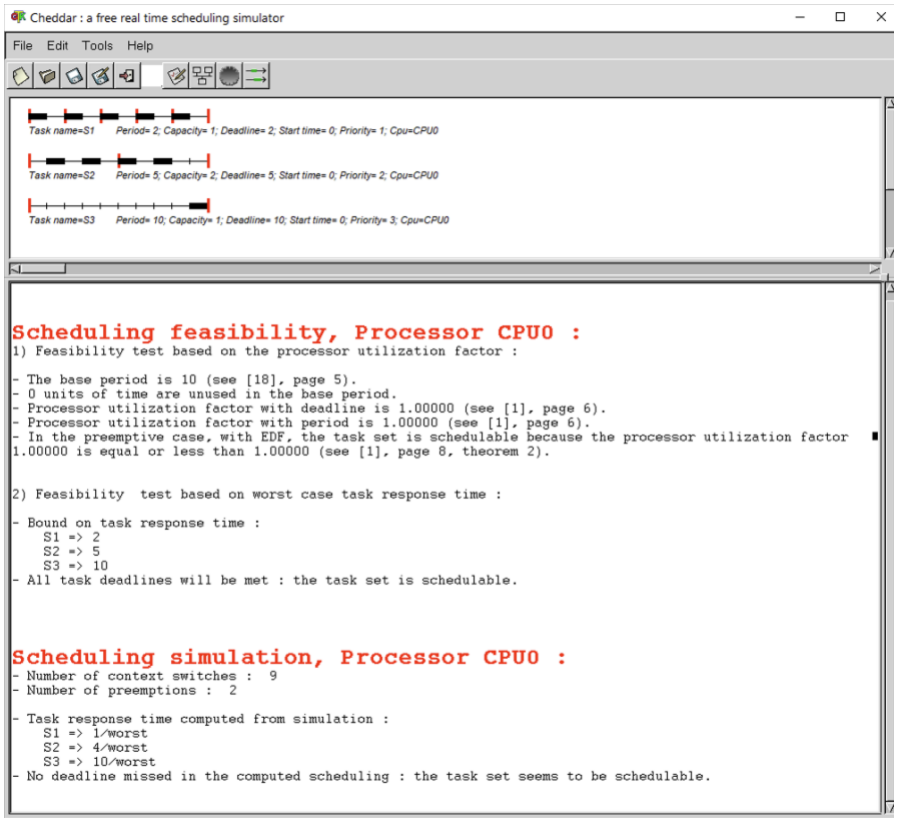
```
chandana@rtesspring24-desktop:~/rtos/ex2/Ex2FeasibilityCodev2$ make clean; make
rm -f *.o *.d
rm -f feasibility_tests
gcc -O0 -g   -c feasibility_tests.c
gcc  -O0 -g   -o feasibility_tests feasibility_tests.o -lm
chandana@rtesspring24-desktop:~/rtos/ex2/Ex2FeasibilityCodev2$ ./feasibility_tests
**** Example-5 ****
U=100.00% (C1=1, C2=2, C3=1; T1=2, T2=5, T3=10; T=D):
CT test FEASIBLE
SP test FEASIBLE
for 0, wcet=1.000000, period=2.000000, utility_sum = 0.500000
for 1, wcet=2.000000, period=5.000000, utility_sum = 0.900000
for 2, wcet=1.000000, period=10.000000, utility_sum = 1.000000
RM utility_sum = 1.000000
LUB = 0.779763
RM LUB INFEASIBLE

for 0, wcet=1.000000, period=2.000000, utility_sum = 0.500000
for 1, wcet=2.000000, period=5.000000, utility_sum = 0.900000
for 2, wcet=1.000000, period=10.000000, utility_sum = 1.000000
EDF/LLF utility_sum = 1.000000
EDF/LLF FEASIBLE
```

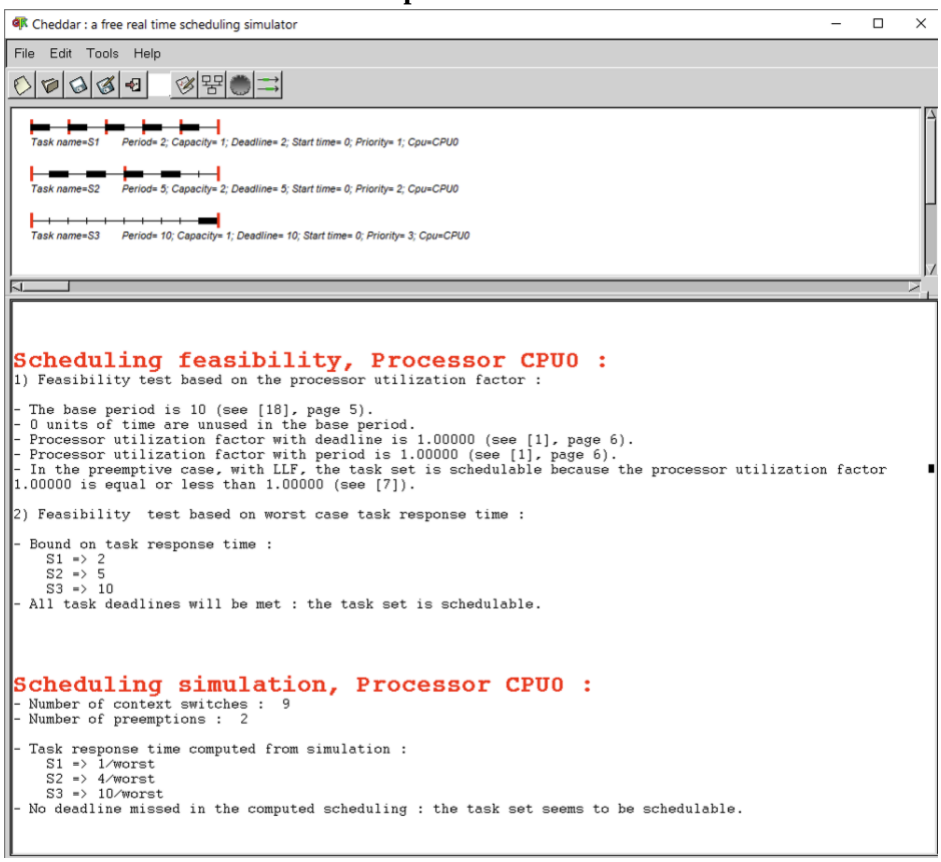
Screenshot of Cheddar RM output:



Screenshot of Cheddar EDF output:



Screenshot of Cheddar LLF output:



Analysis:

Code output				Cheddar			
Completion Test	**Scheduling point test**	**RM LUB**	**EDF/LLF**	**Worst-case analysis**	**RM**	**EDF**	**LLF**
Feasible	Feasible	Infeasible	Feasible	Feasible	Feasible	Feasible	Feasible

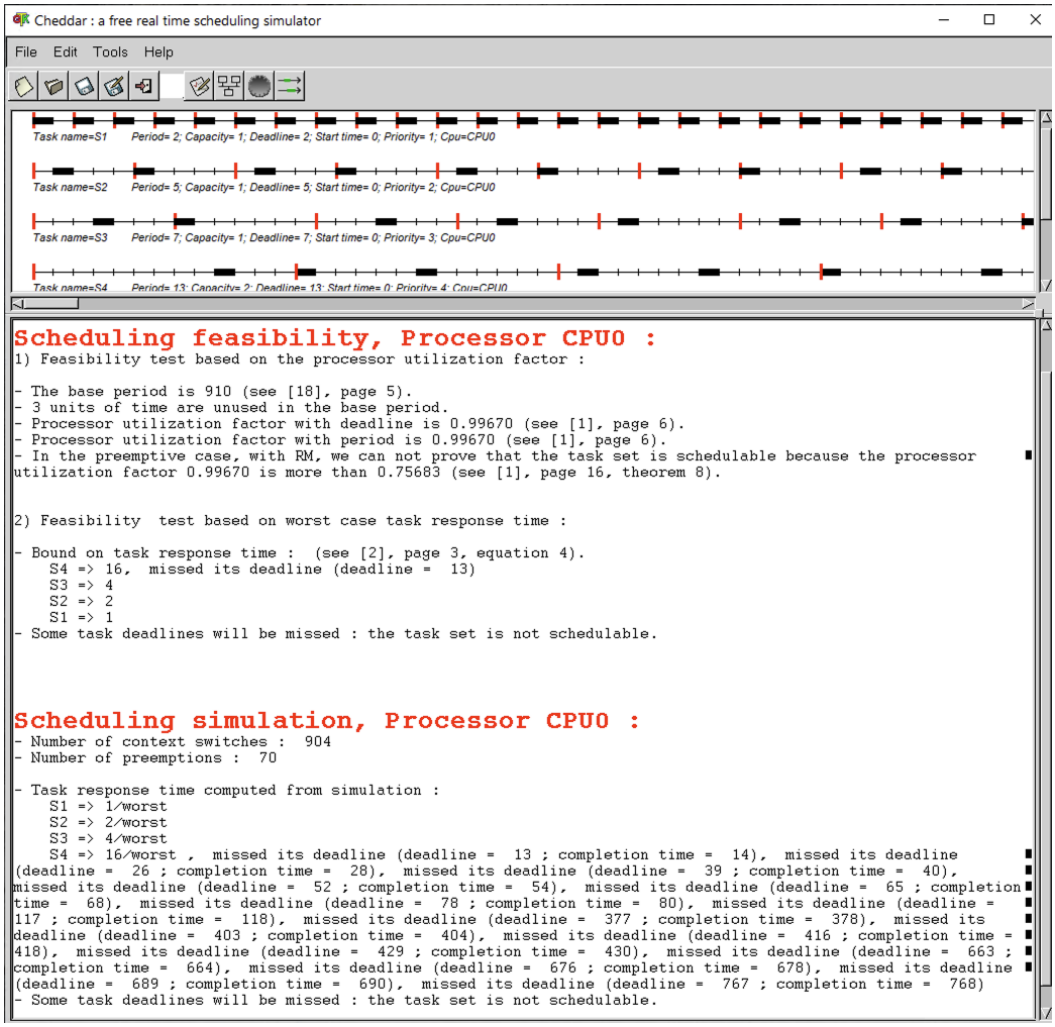
All code results match with Cheddar except RM test. For RM test analysis, Cheddar uses RM LUB, worst-case analysis tests (completion/scheduling tests), scheduling the services over LCM of periods and checks for missed deadlines. If there are missed deadlines, then Cheddar mentions the schedule as not schedulable (Infeasible). While in RM test feasibility code, we had used only RM LUB to decide the feasibility. If Completion test and Scheduling point test results from code are considered in RM test code along with RM LUB, it will match the Cheddar output results.

Example-6:

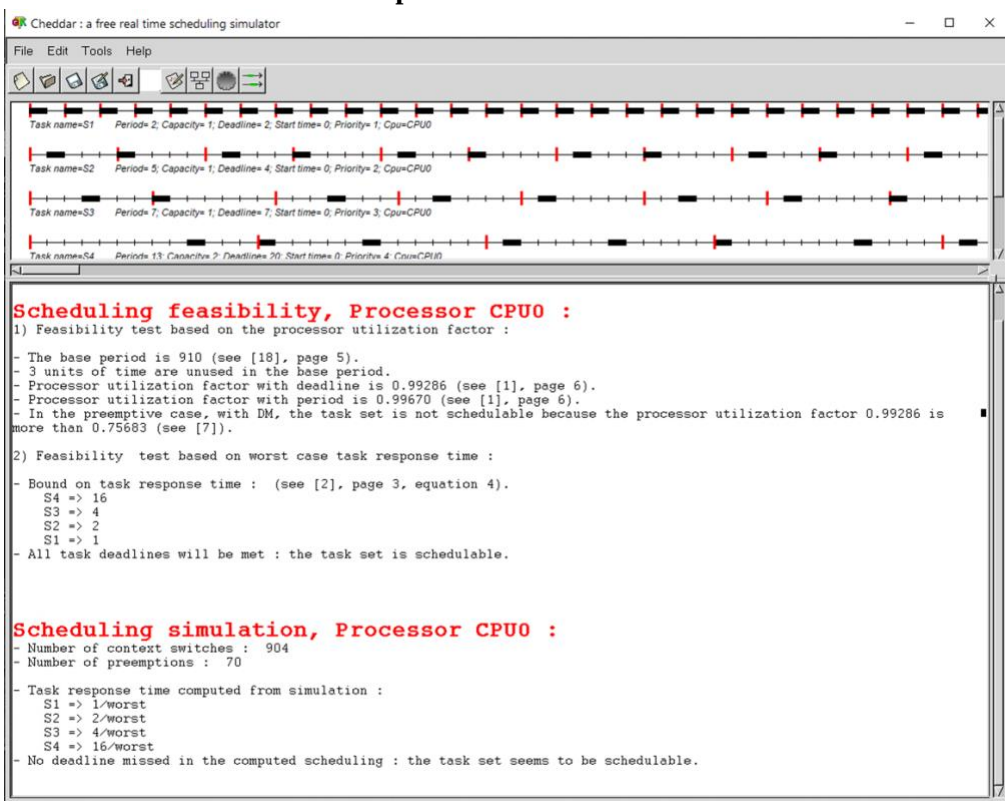
Screenshot of code RM, DM feasibility tests output:

```
**** Example-6 ****
U=99.67% (C1=1, C2=1, C3=1, C4=2; T1=2, T2=5, T3=7, T4=13; T!=D):
CT test INFEASIBLE
SP test INFEASIBLE
for 0, wcet=1.000000, period=2.000000, utility_sum = 0.500000
for 1, wcet=1.000000, period=5.000000, utility_sum = 0.700000
for 2, wcet=1.000000, period=7.000000, utility_sum = 0.842857
for 3, wcet=2.000000, period=13.000000, utility_sum = 0.996703
RM utility_sum = 0.996703
LUB = 0.756828
RM LUB INFEASIBLE
for 0, wcet=1.000000, period=2.000000, interference_time = 0.000000, service_sum = 0.500000
for 1, wcet=1.000000, period=5.000000, interference_time = 2.000000, service_sum = 0.750000
for 2, wcet=1.000000, period=7.000000, interference_time = 6.000000, service_sum = 1.000000
for 3, wcet=2.000000, period=13.000000, interference_time = 17.000000, service_sum = 0.950000
DM utility_sum = 0.992857
DM FEASIBLE
```

Screenshot of Cheddar RM output:



Screenshot of Cheddar DM output:



Analysis:

Observation: LCM mentioned in excel sheet is incorrect, it should be 910.

Code output				Cheddar		
Completion Test	**Scheduling point test**	**RM LUB**	**DM**	**Worst-case analysis of DM**	**RM**	**DM**
Infeasible	Infeasible	Infeasible	Feasible	Feasible	Infeasible	Feasible

All test results match with code and Cheddar.

In this case, RM fails over DM as it had missed deadlines over the LCM of periods and utilization is more than LUB. In DM, deadlines are considered along with periods of service and interference time of each service is taken into consideration and checks whether service is schedulable to 100% utilization.

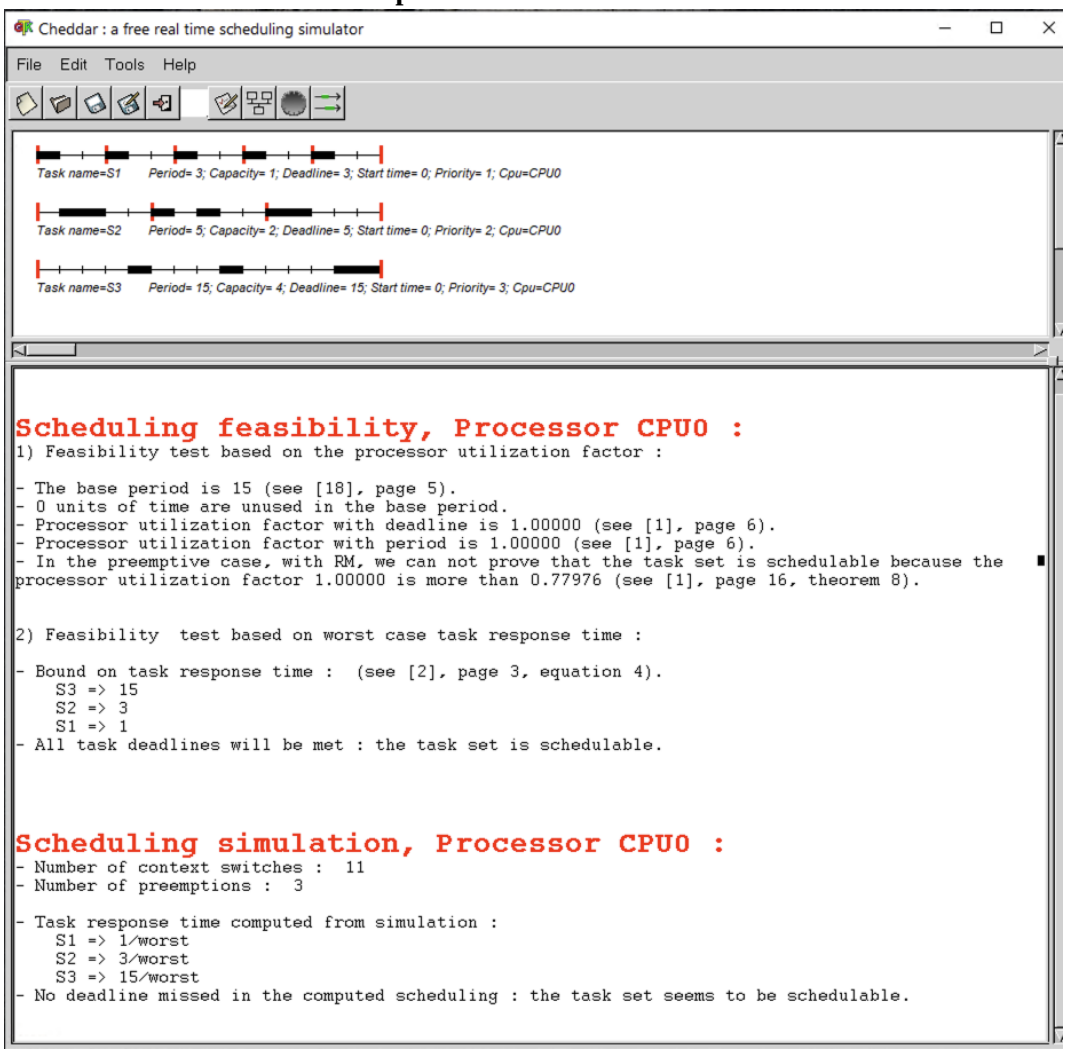
NOTE: Cheddar DM uses LUB instead of checking each service feasibility using Audsley formula.

Example-7:

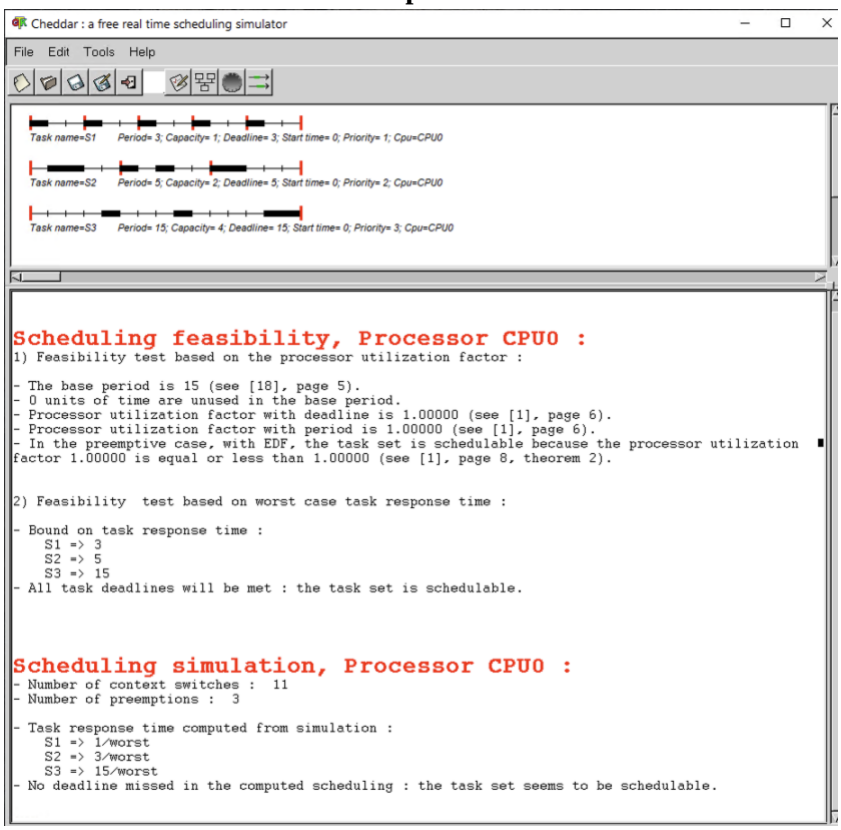
Screenshot of code RM, EDF, LLF feasibility tests output:

```
**** Example-7 ****
U=100.00% (C1=1, C2=2, C3=4; T1=3, T2=5, T3=15; T=D):
CT test FEASIBLE
SP test FEASIBLE
for 0, wcet=1.000000, period=3.000000, utility_sum = 0.333333
for 1, wcet=2.000000, period=5.000000, utility_sum = 0.733333
for 2, wcet=4.000000, period=15.000000, utility_sum = 1.000000
RM utility_sum = 1.000000
LUB = 0.779763
RM LUB INFEASIBLE
for 0, wcet=1.000000, period=3.000000, utility_sum = 0.333333
for 1, wcet=2.000000, period=5.000000, utility_sum = 0.733333
for 2, wcet=4.000000, period=15.000000, utility_sum = 1.000000
EDF/LLF utility_sum = 1.000000
EDF/LLF FEASIBLE
```

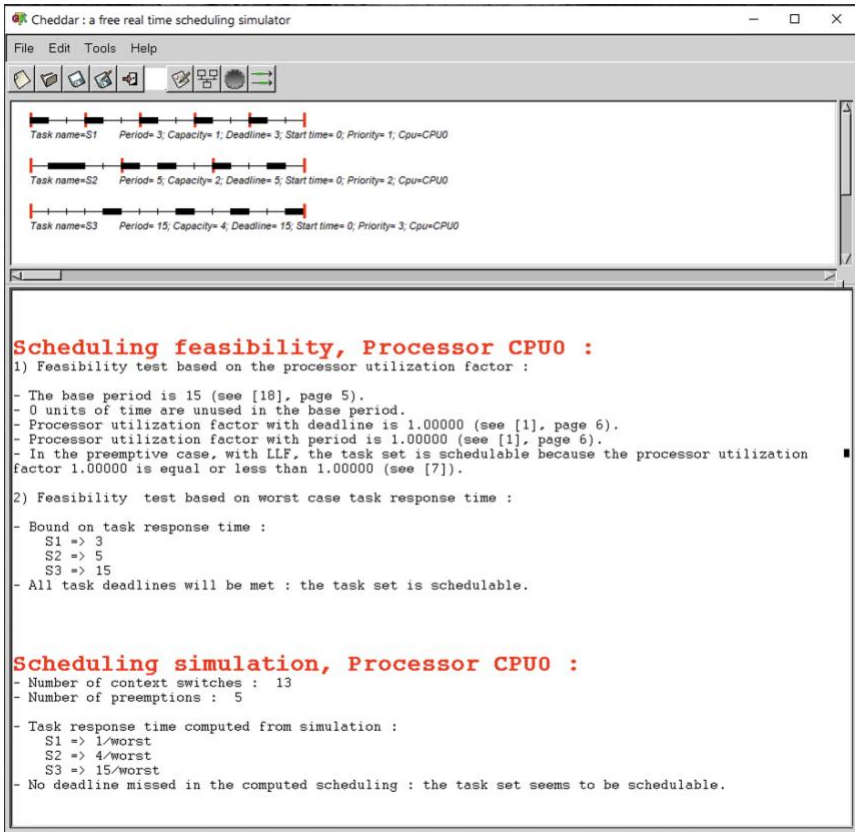
Screenshot of Cheddar RM output:



Screenshot of Cheddar EDF output:



Screenshot of Cheddar LLF output:



Analysis:

Code output				Cheddar			
Completion Test	**Scheduling point test**	**RM LUB**	**EDF/LLF**	**Worst-case analysis**	**RM**	**EDF**	**LLF**
Feasible	Feasible	Infeasible	Feasible	Feasible	Feasible	Feasible	Feasible

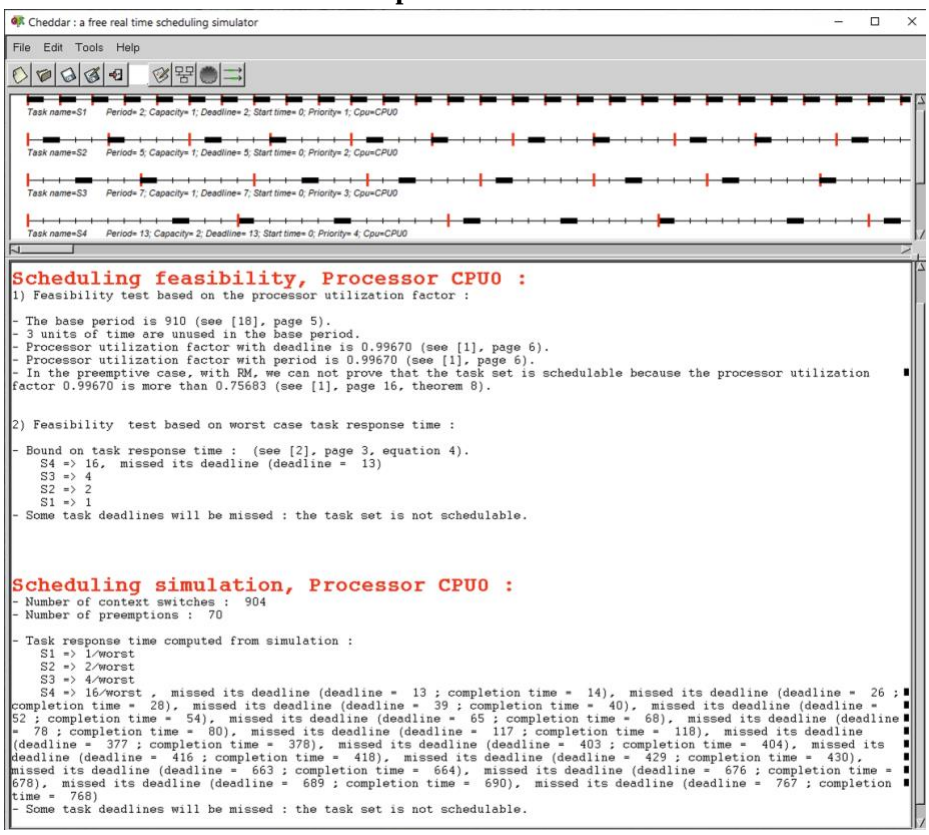
All code results match with Cheddar except RM test. For RM test analysis, Cheddar uses RM LUB, worst-case analysis tests (completion/scheduling tests), scheduling the services over LCM of periods and checks for missed deadlines. If there are missed deadlines, then Cheddar mentions the schedule as not schedulable (Infeasible). While in RM test feasibility code, we had used only RM LUB to decide the feasibility. If Completion test and Scheduling point test results from code are considered in RM test code along with RM LUB, it will match the Cheddar output results.

Example-8:

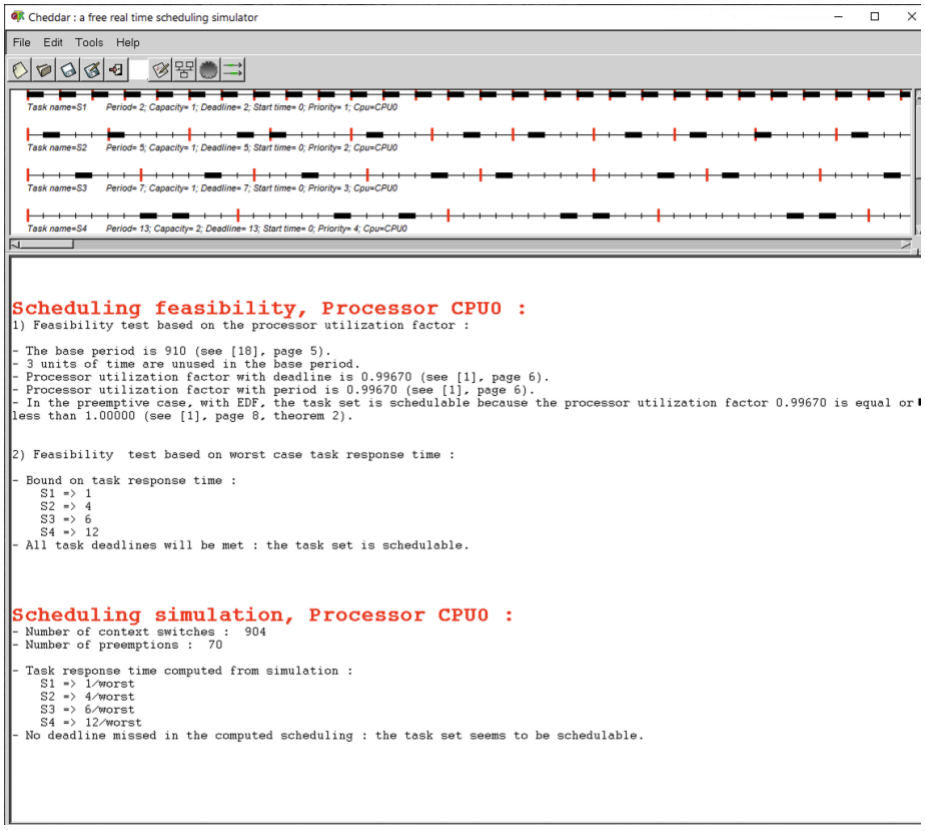
Screenshot of code RM, EDF, LLF feasibility tests output:

```
**** Example-8 ****
U=99.67% (C1=1, C2=1, C3=1, C4=2; T1=2, T2=5, T3=7, T4=13; T=D):
CT test INFEASIBLE
SP test INFEASIBLE
for 0, wcet=1.000000, period=2.000000, utility_sum = 0.500000
for 1, wcet=1.000000, period=5.000000, utility_sum = 0.700000
for 2, wcet=1.000000, period=7.000000, utility_sum = 0.842857
for 3, wcet=2.000000, period=13.000000, utility_sum = 0.996703
RM utility_sum = 0.996703
LUB = 0.756828
RM LUB INFEASIBLE
for 0, wcet=1.000000, period=2.000000, utility_sum = 0.500000
for 1, wcet=1.000000, period=5.000000, utility_sum = 0.700000
for 2, wcet=1.000000, period=7.000000, utility_sum = 0.842857
for 3, wcet=2.000000, period=13.000000, utility_sum = 0.996703
EDF/LLF utility_sum = 0.996703
EDF/LLF FEASIBLE
```

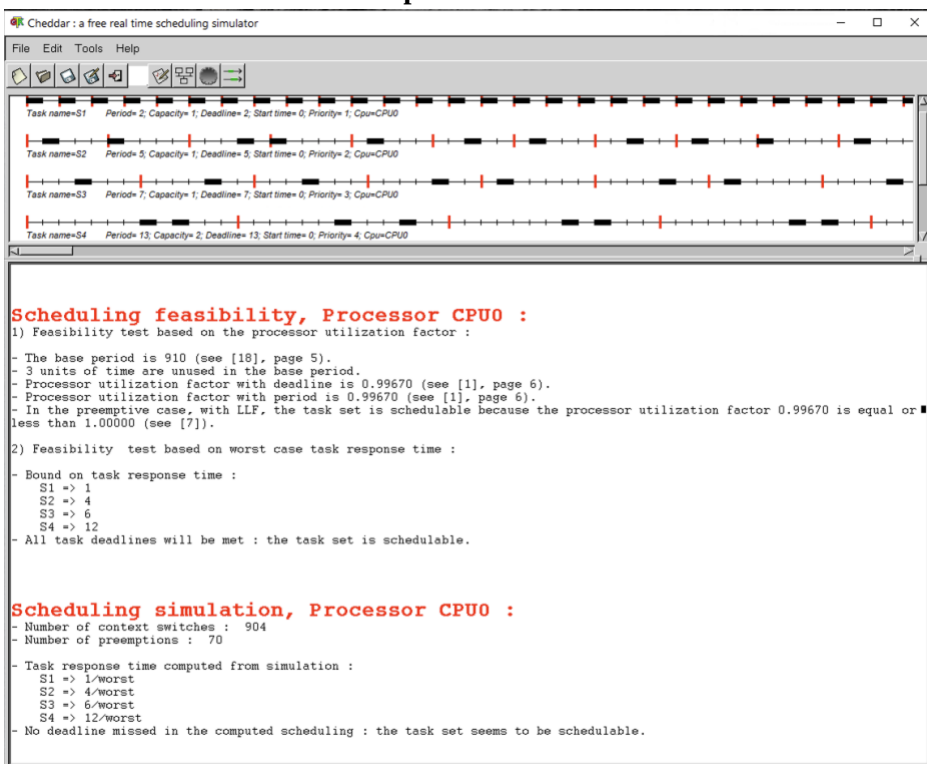
Screenshot of Cheddar RM output:



Screenshot of Cheddar EDF output:



Screenshot of Cheddar LLF output:



Analysis:

Observation: LCM mentioned in excel sheet is incorrect, it should be 910.

In excel sheet, C2 is assumed as 2 for scheduling analysis. We considered C2=1 and provided analysis.

Code output				Cheddar			
Completion Test	**Scheduling point test**	**RM LUB**	**EDF/LLF**	**Worst-case analysis**	**RM**	**EDF**	**LLF**
Infeasible	Infeasible	Infeasible	Feasible	Feasible	Infeasible	Feasible	Feasible

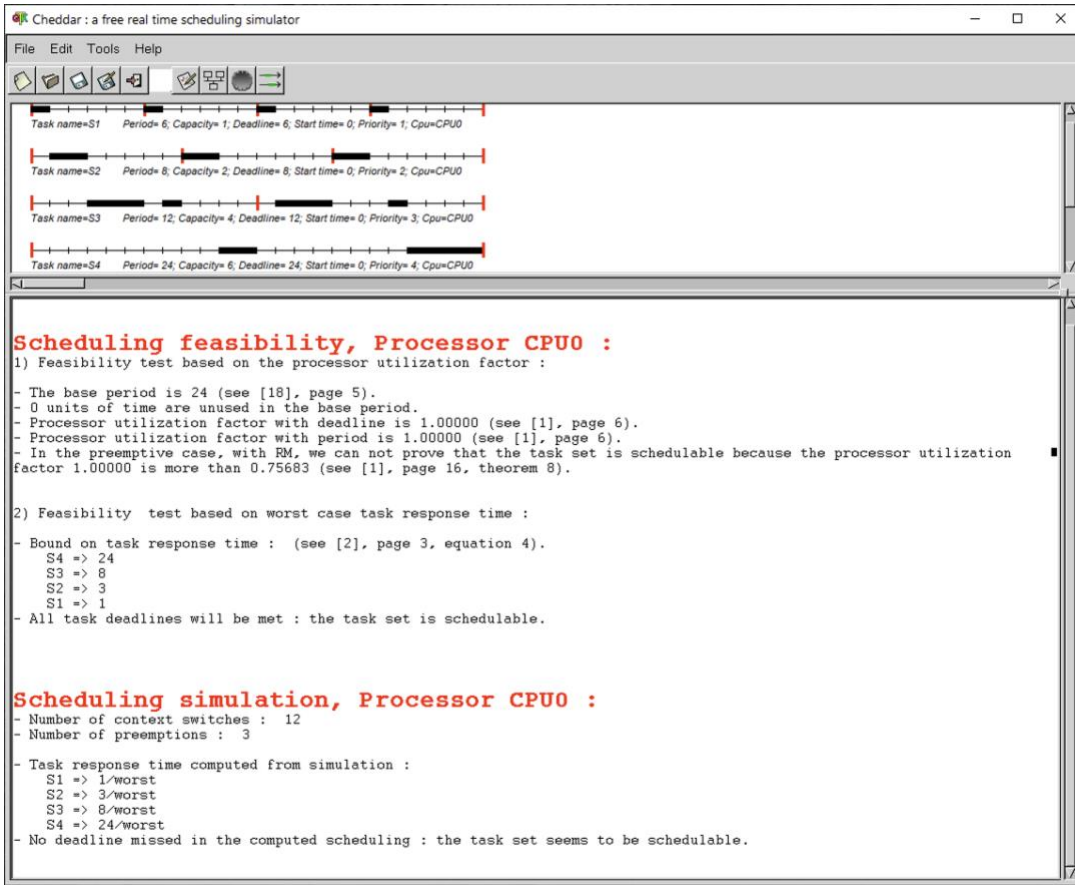
All test results match with code and Cheddar.

In this case from Cheddar Analysis, **RM fails but EDF/LLF succeeds**. RM uses static priority scheduling and checks if utilization is less than or equal to RM LUB and expect not to schedule services up to 100% of CPU and try to schedule the service over LCM of periods. While EDF/LLF are dynamic priority scheduling by calculating the remaining deadlines, computation times in run-time and changes the priorities of service to schedule up to 100% of CPU. Hence, in this case EDF/LLF succeeds over RM as the utilization of services is less than 1 and doesn't miss any deadlines in simulation compared to RM.

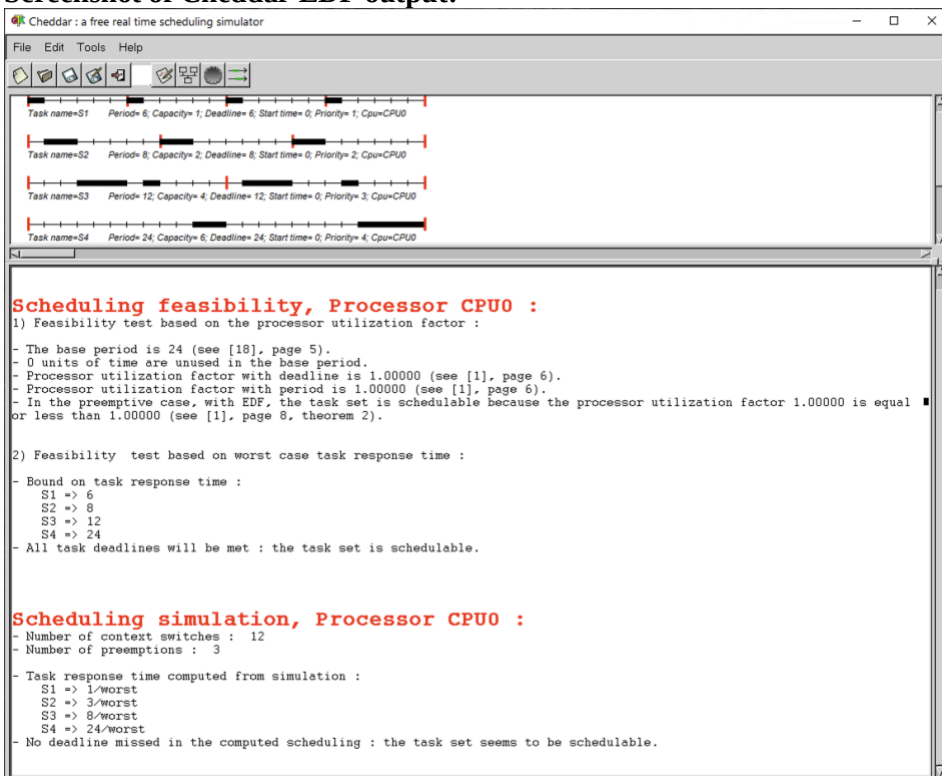
Example-9:
Screenshot of code RM, EDF, LLF feasibility tests output:

```
**** Example-9 ****
U=100.00% (C1=1, C2=2, C3=4, C3=6; T1=6, T2=8, T3=12, T4=24; T=D):
CT test FEASIBLE
SP test FEASIBLE
for 0, wcet=1.000000, period=6.000000, utility_sum = 0.166667
for 1, wcet=2.000000, period=8.000000, utility_sum = 0.416667
for 2, wcet=4.000000, period=12.000000, utility_sum = 0.750000
for 3, wcet=6.000000, period=24.000000, utility_sum = 1.000000
RM utility_sum = 1.000000
LUB = 0.756828
RM LUB INFEASIBLE
for 0, wcet=1.000000, period=6.000000, utility_sum = 0.166667
for 1, wcet=2.000000, period=8.000000, utility_sum = 0.416667
for 2, wcet=4.000000, period=12.000000, utility_sum = 0.750000
for 3, wcet=6.000000, period=24.000000, utility_sum = 1.000000
EDF/LLF utility_sum = 1.000000
EDF/LLF FEASIBLE
```

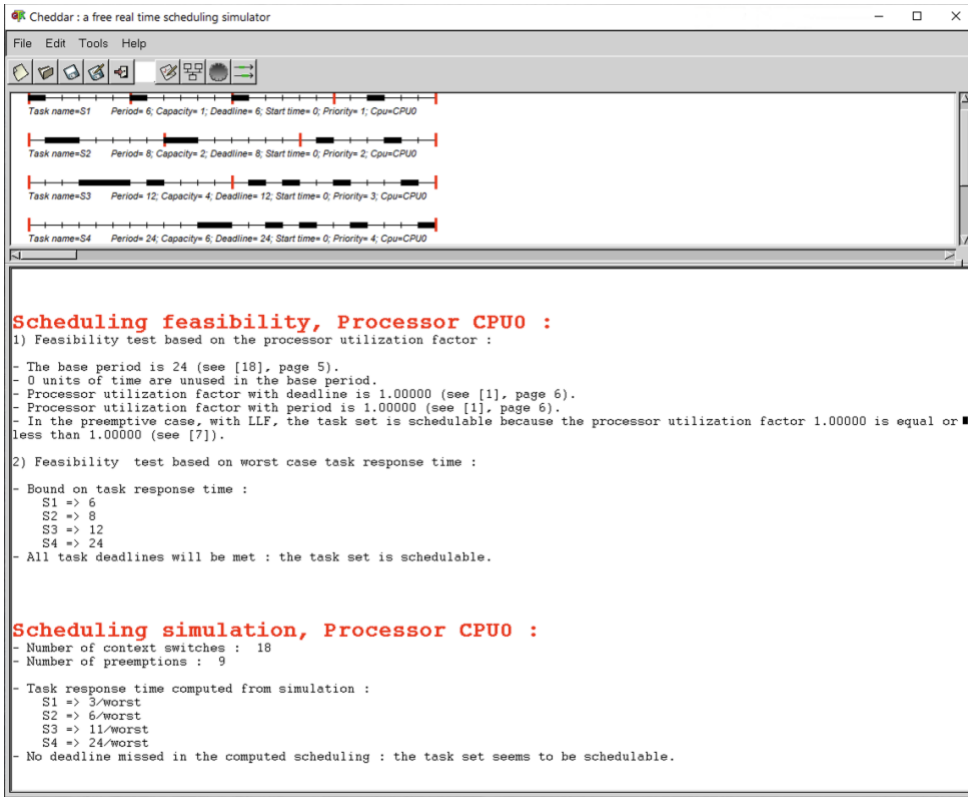
Screenshot of Cheddar RM output:



Screenshot of Cheddar EDF output:



Screenshot of Cheddar LLF output:



Analysis:

Code output				Cheddar		
Completion Test	**Scheduling point test**	**RM LUB**	**EDF/LLF**	**RM**	**EDF**	**LLF**
Feasible	Feasible	Infeasible	Feasible	Feasible	Feasible	Feasible

All code results match with Cheddar except RM test. For RM test analysis, Cheddar uses RM LUB, worst-case analysis tests (completion/scheduling tests), scheduling the services over LCM of periods and checks for missed deadlines. If there are missed deadlines, then Cheddar mentions the schedule as not schedulable (Infeasible). While in RM test feasibility code, we had used only RM LUB to decide the feasibility. If Completion test and Scheduling point test results from code are considered in RM test code along with RM LUB, it will match the Cheddar output results.

c) Does your modified Feasibility code agree with Cheddar analysis in all 5 additional cases? Why or why not?

Answer:

Modified Feasibility code DM and EDF/LLF tests analysis agrees with Cheddar analysis for all 5 additional cases.

In example-6 and example-8, all the outputs Completion test, scheduling test, RM, EDF/LLF, DM from code matches/agrees with Cheddar analysis.

In example-5, example-7 and example-9, EDF/LLF results match/agree but RM test did not match.

For RM test analysis, Cheddar uses RM LUB, worst-case analysis tests (completion/scheduling tests), scheduling the services over LCM of periods and checks for missed deadlines. If there are missed deadlines, then Cheddar mentions the schedule as not schedulable (Infeasible). While in RM test feasibility code, we had used only RM LUB to decide the feasibility. If Completion test and Scheduling point test results from code are considered in RM test code along with RM LUB, it will match the Cheddar output results.

NOTE: Cheddar DM uses LUB instead of checking each service feasibility using Audsley formula.

5. [30 points] Read Chapter 3 of the textbook.

a) 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:

Constraints:

1. **C1: Deadline is equal to period of the task:** In RM LUB derivation, they considered that the task will repeat after a certain period. Which means that the task can have time till that time for completion/execution without being considered as failure. So that we can consider the period of task equal to deadline.
   Analysis:
   This is not feasible if we expect the task to be completed at certain time from its spawn, which may be less than the period of the task.

2. **C2: Fixed priority:** The priority for a task is based on frequency of that task and which is defined at the beginning of dispatching (critical instant) will remain same over time.
   Analysis:
   This will not work for dynamic environments, where new tasks need to be created and scheduled.

3. **Preemptive:** During dispatching of tasks, the task with higher priority is allowed to preempt the lower priority task so that the task with higher frequency and priority to complete.
   Analysis:
   Sometimes preemptive can cause starving to the task with lower priority, due to fixed priorities and period.

Assumptions:

1. **A1: All tasks are periodic:** Assumed that the task considered to be schedule will repeat over and over at particular interval of time.
   Analysis:
   In real-time systems, all the tasks are not periodic. There will be asynchronous tasks which run based on events/interrupts from IO. The real systems are not always independent, so there will be slight overlap between execution of tasks, intern can lead to drift in run-time, which affects the periodicity. In this case, we think this assumption will **not be applicable for actual practice**.

2. **A3: Tasks are independent:** Tasks under scheduling are not dependent on resources such as memory, I/O result related to other tasks which can hinder the execution of task.
   Analysis:
   As we described in the above point, tasks in real systems are not always independent. Because most of the task depends on the shared memory variables. Considering above factors, we think this assumption will **not be applicable for actual practice**.

3. **A4: Runtime for each task is deterministic/constant:** Considered the task execution time will remain same without any deviation due to any resource dependency.
   Analysis:
   In real-time systems, run-time of tasks may vary based upon task context switching, dynamic priorities, preemption, dependency on other resources. If loops are part of task and if it varies

from case to case, the execution time will definitely vary. Considering above factors, we think this assumption will **not be applicable for actual practice**.

b) 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:

Key derivation Step-1: Execution Time calculation

In case 1, C_1 is considered as short enough to fit all three releases in T_2.

To calculate the execution time of service 1, it is assumed that all three S_1 releases will fit in T_2 which is described as equation 1. This expression considers the minimum number of times that T_1 can occur in T_2 period that is $T_1[T_2/T_1]$ and floor function is applied to it.

To calculate the execution time of service 2, all the time not used by S_1 is considered. That is the remaining time of T_2 in which S_1 is not executing is expressed as equation-2 and ceil function is applied to calculate the number of times S_1 is executed.

$$Eq\text{–}1 : C_1 \le T_2 - T_1\left\lfloor T_2/T_1 \right\rfloor$$

$$Eq\text{–}2 : C_2 = T_2 - C_1\left\lceil T_2/T_1 \right\rceil$$

We did not understand exactly in which situation a ceil/floor function should be used and why it is used in specific way in case 1 and we consider this calculation as tricky math.

From the scheduling diagram, it describes that the increasing C1 by keeping C2 constant is not feasible but increasing C2 can be done by keeping C1 constant. This is not quite clear to us how increasing C2 is feasible.

Key derivation Step-2: Fractional Interference assumption

After considering case 1 and case 2 and substituting the execution times of S_1 and S_2 as C_1 and C_2 in the utility equation below,

$$U = \frac{C_1}{T_1} + \frac{C_2}{T_2}$$

We derive below Utility equation,

$$U = 1 - (T_1/T_2)\left[\lceil T_2/T_1 \rceil - (T_2/T_1)\right]\left[(T_2/T_1) - \lfloor T_2/T_1 \rfloor\right] \quad (3.12)$$

After this step, the interference I is considered as

$$I = \lfloor T_2/T_1 \rfloor$$

And fractional interference is considered as

$$f = (T_2/T_1) - \lfloor T_2/T_1 \rfloor$$

After substituting fractional interference in above 3.12 equation of utility, we derive below equation,

$$U = 1 - \left(\frac{f(1-f)}{(T_2 / T_1)} \right) \qquad (3.13)$$

We did not understand how interference and fractional interference are considered as mentioned above and how those expressions are derived.

Key derivation Step-3: Fractional Interference derivation

After we consider getting equation of U with respect to fractional interference f, a derivative w.r.t. f is applied on U on both sides of equation. Here they try to solve for f and reduced the equation directly in one step which we felt as a very tricky math and hard to understand.

The final equation is as shown below,

$$U = 1 - \left(\frac{(f - f^2)}{(1+f)} \right)$$

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:

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

This equation of fractional interference is used to map to the RM LUB derivation considering the number of services as 2. Hence, we felt it's important to understand above derivation step in detail.

Key derivation Step-4: Intersection of Slopes

For case1 and case2 we derived the equation for C1 and C2 from S1 and S2 data. Then combined the equations to find the RM upper bound. We are missing the information about why we considering the intersection of slopes from case1 and case2 and finally why we end up considering only the equation from case1 for C2.

APPENDIX

Section 1:

Feasibilitytests.c Code:

```
/ Sam Siewert, August 2020
//
// 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
//
// All 3 are also covered in RTECS with Linux and RTOS p. 84 to p. 89
//
// Original references for single core AMP systems:
//
// 1) RM LUB - Liu, Chung Laung, and James W. Layland. "Scheduling algorithms for multiprogramming in a hard-real-time environment." Journal of the ACM (JACM) 20.1 (1973): 46-61.
// 2) Scheduling Point - Lehoczky, John, Lui Sha, and Yuqin Ding. "The rate monotonic scheduling algorithm: Exact characterization and average case behavior." RTSS. Vol. 89. 1989.
// 3) Completion Test - Joseph, Mathai, and Paritosh Pandya. "Finding response times in a real-time system." The Computer Journal 29.5 (1986): 390-395.
//
// References for mulit-core systems:
//
// 1) Bertossi, Alan A., Luigi V. Mancini, and Federico Rossini. "Fault-tolerant rate-monotonic first-fit scheduling in hard-real-time systems."
//    IEEE Transactions on Parallel and Distributed Systems 10.9 (1999): 934-945.
```

```
// 2) Burchard, Almut, et al. "New strategies for assigning real-time tasks to multiprocessor systems." IEEE transactions on computers 44.12 (1995): 1429-1442.
// 3) Dhall, Sudarshan K., and Chung Laung Liu. "On a real-time scheduling problem." Operations research 26.1 (1978): 127-140.
//
//
// Deadline Montonic (not implemented in this example, but covered in class and notes):
//
// 1) Audsley, Neil C., et al. "Hard real-time scheduling: The deadline-monotonic approach." IFAC Proceedings Volumes 24.2 (1991): 127-132.
//
// Note that Deadline Monotoic simply uses the deadine interval, D(i) to assign priority, rather than the period interval, T(i) and relaxes T=D constraint.  Anlaysis can
// be done as it is done for RM, but with evaluation of feasbility based upon modified D(i) and with modified fixed priorities.  This is covered by manual analysis examples.
//
// For a more interactive tool, students can use Cheddar:
//
// http://beru.univ-brest.fr/~singhoff/cheddar/
//
// This open source tool handles single and multi-core and allows for modeling of the platform hardware, RTOS/OS, and scheduler with a particular fixed priority or dynamic
// priority policy.
//
// This code is provided primarily so students can learn the methods of worst case analysis and compare exact and estimated feasibility decision testing.
//

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

#define TRUE 1
#define FALSE 0
#define U32_T unsigned int

// U=1.0
```

```
U32_T ex0_period[] = {2, 5, 10};
U32_T ex0_wcet[] = {1, 2, 1};

// U=0.996
U32_T ex1_period[] = {2, 5, 7, 13};
U32_T ex1_wcet[] = {1, 1, 1, 2};
U32_T ex1_deadline[] = {2, 4, 7, 20};

// U=1.0
U32_T ex2_period[] = {3, 5, 15};
U32_T ex2_wcet[] = {1, 2, 4};

// U=0.996
U32_T ex3_period[] = {2, 5, 7, 13};
U32_T ex3_wcet[] = {1, 1, 1, 2};

// U=1.0
U32_T ex4_period[] = {6, 8, 12, 24};
U32_T ex4_wcet[] = {1, 2, 4, 6};

int completion_time_feasibility(U32_T numServices, U32_T period[], U32_T wcet[], U32_T deadline[]);
int scheduling_point_feasibility(U32_T numServices, U32_T period[], U32_T wcet[], U32_T deadline[]);
int rate_monotonic_least_upper_bound(U32_T numServices, U32_T period[], U32_T wcet[], U32_T
deadline[]);
int dynamic_priority_feasibility_test(U32_T numServices, U32_T period[], U32_T wcet[], U32_T
deadline[]);
int deadline_monotonic_feasibility_test(U32_T numServices, U32_T period[], U32_T wcet[], U32_T
deadline[]);
int main(void)
{
   int i;
   U32_T numServices;

   printf("**** Example-5 ****\n");
   printf("U=%4.2f%% (C1=1, C2=2, C3=1; T1=2, T2=5, T3=10; T=D): \n",
        ((1.0/2.0)*100.0 + (2.0/5.0)*100.0 + (1.0/10.0)*100.0));
```

```
numServices = 3;
if(completion_time_feasibility(numServices, ex0_period, ex0_wcet, ex0_period) == TRUE)
   printf("CT test FEASIBLE\n");
else
   printf("CT test INFEASIBLE\n");

if(scheduling_point_feasibility(numServices, ex0_period, ex0_wcet, ex0_period) == TRUE)
   printf("SP test FEASIBLE\n");
else
   printf("SP test INFEASIBLE\n");

if(rate_monotonic_least_upper_bound(numServices, ex0_period, ex0_wcet, ex0_period) == TRUE)
   printf("RM LUB FEASIBLE\n");
else
   printf("RM LUB INFEASIBLE\n");
printf("\n");

if(dynamic_priority_feasibility_test(numServices, ex0_period, ex0_wcet, ex0_period) == TRUE)
   printf("EDF/LLF FEASIBLE\n");
else
   printf("EDF/LLF INFEASIBLE\n");
printf("\n");

printf("**** Example-6 ****\n");
printf("U=%4.2f%% (C1=1, C2=1, C3=1, C4=2; T1=2, T2=5, T3=7, T4=13; T!=D): \n",
     ((1.0/2.0)*100.0 + (1.0/5.0)*100.0 + (1.0/7.0)*100.0 + (2.0/13.0)*100.0));
numServices = 4;
if(completion_time_feasibility(numServices, ex1_period, ex1_wcet, ex1_period) == TRUE)
   printf("CT test FEASIBLE\n");
else
   printf("CT test INFEASIBLE\n");

if(scheduling_point_feasibility(numServices, ex1_period, ex1_wcet, ex1_period) == TRUE)
   printf("SP test FEASIBLE\n");
else
   printf("SP test INFEASIBLE\n");
```

```
if(rate_monotonic_least_upper_bound(numServices, ex1_period, ex1_wcet, ex1_period) == TRUE)
   printf("RM LUB FEASIBLE\n");
else
   printf("RM LUB INFEASIBLE\n");

if(deadline_monotonic_feasibility_test(numServices, ex1_period, ex1_wcet, ex1_deadline) == TRUE)
   printf("DM FEASIBLE\n");
else
   printf("DM INFEASIBLE\n");
printf("\n");

printf("**** Example-7 ****\n");
printf("U=%4.2f%% (C1=1, C2=2, C3=4; T1=3, T2=5, T3=15; T=D): \n",
     ((1.0/3.0)*100.0 + (2.0/5.0)*100.0 + (4.0/15.0)*100.0));
numServices = 3;
if(completion_time_feasibility(numServices, ex2_period, ex2_wcet, ex2_period) == TRUE)
   printf("CT test FEASIBLE\n");
else
   printf("CT test INFEASIBLE\n");

if(scheduling_point_feasibility(numServices, ex2_period, ex2_wcet, ex2_period) == TRUE)
   printf("SP test FEASIBLE\n");
else
   printf("SP test INFEASIBLE\n");

if(rate_monotonic_least_upper_bound(numServices, ex2_period, ex2_wcet, ex2_period) == TRUE)
   printf("RM LUB FEASIBLE\n");
else
   printf("RM LUB INFEASIBLE\n");

if(dynamic_priority_feasibility_test(numServices, ex2_period, ex2_wcet, ex2_period) == TRUE)
   printf("EDF/LLF FEASIBLE\n");
else
   printf("EDF/LLF INFEASIBLE\n");
printf("\n");
```

```
printf("**** Example-8 ****\n");
printf("U=%4.2f%% (C1=1, C2=1, C3=1, C4=2; T1=2, T2=5, T3=7, T4=13; T=D): \n",
      ((1.0/2.0)*100.0 + (1.0/5.0)*100.0 + (1.0/7.0)*100.0 + (2.0/13.0)*100.0));
numServices = 4;
if(completion_time_feasibility(numServices, ex3_period, ex3_wcet, ex3_period) == TRUE)
   printf("CT test FEASIBLE\n");
else
   printf("CT test INFEASIBLE\n");

if(scheduling_point_feasibility(numServices, ex3_period, ex3_wcet, ex3_period) == TRUE)
   printf("SP test FEASIBLE\n");
else
   printf("SP test INFEASIBLE\n");

if(rate_monotonic_least_upper_bound(numServices, ex3_period, ex3_wcet, ex3_period) == TRUE)
   printf("RM LUB FEASIBLE\n");
else
   printf("RM LUB INFEASIBLE\n");

if(dynamic_priority_feasibility_test(numServices, ex3_period, ex3_wcet, ex3_period) == TRUE)
   printf("EDF/LLF FEASIBLE\n");
else
   printf("EDF/LLF INFEASIBLE\n");
printf("\n");

printf("**** Example-9 ****\n");
printf("U=%4.2f%% (C1=1, C2=2, C3=4, C3=6; T1=6, T2=8, T3=12, T4=24; T=D): \n",
      ((1.0/6.0)*100.0 + (2.0/8.0)*100.0 + (4.0/12.0)*100.0 + (6.0/24.0)*100.0));
numServices = 4;
if(completion_time_feasibility(numServices, ex4_period, ex4_wcet, ex4_period) == TRUE)
   printf("CT test FEASIBLE\n");
else
   printf("CT test INFEASIBLE\n");

if(scheduling_point_feasibility(numServices, ex4_period, ex4_wcet, ex4_period) == TRUE)
```

```
      printf("SP test FEASIBLE\n");
   else
      printf("SP test INFEASIBLE\n");

   if(rate_monotonic_least_upper_bound(numServices, ex4_period, ex4_wcet, ex4_period) == TRUE)
      printf("RM LUB FEASIBLE\n");
   else
      printf("RM LUB INFEASIBLE\n");

   if(dynamic_priority_feasibility_test(numServices, ex4_period, ex4_wcet, ex4_period) == TRUE)
      printf("EDF/LLF FEASIBLE\n");
   else
      printf("EDF/LLF INFEASIBLE\n");
   printf("\n");

}


int rate_monotonic_least_upper_bound(U32_T numServices, U32_T period[], U32_T wcet[], U32_T
deadline[])
{
  double utility_sum=0.0, lub=0.0;
  int idx;

  //printf("for %d, utility_sum = %lf\n", numServices, utility_sum);

  // Sum the C(i) over the T(i)
  for(idx=0; idx < numServices; idx++)
  {
    utility_sum += ((double)wcet[idx] / (double)period[idx]);
    printf("for %d, wcet=%lf, period=%lf, utility_sum = %lf\n", idx, (double)wcet[idx], (double)period[idx],
utility_sum);
  }
  printf("RM utility_sum = %lf\n", utility_sum);

  // Compute LUB for number of services
```

```
  lub = (double)numServices * (pow(2.0, (1.0/((double)numServices))) - 1.0);
  printf("LUB = %lf\n", lub);

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


int dynamic_priority_feasibility_test(U32_T numServices, U32_T period[], U32_T wcet[], U32_T
deadline[])
{
  double utility_sum=0.0, lub=0.0;
  int idx;

  //printf("for %d, utility_sum = %lf\n", numServices, utility_sum);

  // Sum the C(i) over the T(i)
  for(idx=0; idx < numServices; idx++)
  {
    utility_sum += ((double)wcet[idx] / (double)period[idx]);
    printf("for %d, wcet=%lf, period=%lf, utility_sum = %lf\n", idx, (double)wcet[idx], (double)period[idx],
utility_sum);
  }
  printf("EDF/LLF utility_sum = %lf\n", utility_sum);

  // Compare the utilty to the bound and return feasibility
  if(utility_sum <= 1)
    return TRUE;
  else
    return FALSE;
}
```

```
int deadline_monotonic_feasibility_test(U32_T numServices, U32_T period[], U32_T wcet[], U32_T
deadline[])
{
  double utility_sum=0.0, lub=0.0, service_sum = 0.0;
  double interference_time = 0.0;
  int idx;
  int j;
  int isFeasible = FALSE;

  // Sum the C(i) over the T(i)
  for(idx=0; idx < numServices; idx++)
  {
    // Sum the C(i) over the T(i)
    utility_sum += ((double)wcet[idx] / (double)deadline[idx]);
    interference_time = 0.0;
    for(j=0; j < idx; j++)
    {
      interference_time += ((double)wcet[j] * (ceil((double)deadline[idx] / (double)period[j])));
    }
    service_sum = (((double)wcet[idx] + (double)interference_time) / (double)deadline[idx]);
    printf("for %d, wcet=%lf, period=%lf, interference_time = %lf, service_sum = %lf\n", idx,
(double)wcet[idx], (double)period[idx], interference_time, service_sum);

    // Compare the each service sum for feasibility
    if(service_sum <= 1)
    {
      isFeasible = TRUE;
    }
    else
    {
      isFeasible = FALSE;
      break;
    }
  }
  printf("DM utility_sum = %lf\n", utility_sum);
```

```
 return (isFeasible ? TRUE: FALSE);
}

int completion_time_feasibility(U32_T numServices, U32_T period[], U32_T wcet[], U32_T deadline[])
{
 int i, j;
 U32_T an, anext;

 // assume feasible until we find otherwise
 int set_feasible=TRUE;

 //printf("numServices=%d\n", numServices);

 // For all services in the analysis
 for (i=0; i < numServices; i++)
 {
    an=0; anext=0;

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

    //printf("i=%d, an=%d\n", i, an);

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

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

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

```
        //printf("an=%d, anext=%d\n", an, anext);
    }

    //printf("an=%d, deadline[%d]=%d\n", an, i, deadline[i]);

    if (an > deadline[i])
    {
      set_feasible=FALSE;
    }
 }

 return set_feasible;
}


int scheduling_point_feasibility(U32_T numServices, U32_T period[],
            U32_T wcet[], U32_T deadline[])
{
  int rc = TRUE, i, j, k, l, status, temp;

  // For all services in the analysis
  for (i=0; i < numServices; i++) // iterate from highest to lowest priority
  {
    status=0;

    // Look for all available CPU minus what has been used by higher priority services
    for (k=0; k<=i; k++)
    {
    // find available CPU windows and take them
      for (l=1; l <= (floor((double)period[i]/(double)period[k])); l++)
      {
          temp=0;

          for (j=0; j<=i; j++) temp += wcet[j] * ceil((double)l*(double)period[k]/(double)period[j]);
```

```
        // Can we get the CPU we need or not?
            if (temp <= (l*period[k]))
            {
                // insufficient CPU during our period, therefore infeasible
                status=1;
                break;
            }
        }
        if (status) break;
    }

    if (!status) rc=FALSE;
  }
  return rc;
}
```

Citations:

- Liu and Layland Paper
- RTOS Textbook