ECEN 5623 Exercise 2

Nalin Saxena, Hyounjun Chang

Table of Contents

Nalin Saxena
Hyounjun Chang

Question 1 Answers

1. [5 points] Install Cheddar on a machine of your choice (Windows machine/VM might be easiest). Show a screenshot of it running.

<u>Cheddar screenshot</u>

Question 2 Answers

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.

<u>Overall understanding of paper and key point articulation</u>

The paper talks about a state of the art software system (during the time) developed for NASA's space shuttle program. This system is known as **PASS** or **Primary Avionics Software system.** The software was developed by the IBM Federal Systems Division**.** Pass had redundancy at its core for instance the pre launch sequence had the onboard computer run in a redundant set (set of 4). The computer has a voting mechanism as a mechanism to cross verify the set of values they

were experiencing. The software developed while keeping hardware constraints such as CPU compute, memory latency in mind.

Since the code for pass would be too large to be stored at once in the primary memory of GPC(s)(General purpose computer) therefore the entire code is divided into 8 subsections called OPS-Operational Sequences. The code for this is also organized in an efficient manner-
The first section has code common to all OPS loads, second section contains major functions and common data for multiple common applications and lastly ***OPS overlay*** is the code specific to a certain OPS.Hence in a best case scenario only the OPS overlay would need to be read from the main memory.

One more important thing to note is that the shuttle system is based on the modern fly-by-wire system (unlike the older fly-by-wire-cable) which basically means all the crew inputs will be vetted by the onboard computers before actually performing the necessary action.

Man Machine interface - The man/machine interface is built as a substructure of the OPS which contains 3 main components.

- ***Major-Mode-*** Major modes handle primary tasks within an OPS. It is triggered automatically when entering from one OPS to another and can also be triggered via user input as well. One major mode will be at least entered every time an OPS is executed.
- ***Specialist function*** - Specialist functions also called as ***SPECS*** perform background secondary tasks and a user input is necessary to trigger them. Once a SPEC is executed it runs in a concurrent fashion along with the major modes.
- ***DISP-*** No processing is initiated by the display module. Its primary function is to give an overall picture of the system to the crew with the help of CRT displays.

System-software- Since the requirements of the shuttle program were subject to frequent changes a nonsynchronous approach for the PASS architecture. The application level processes were isolated from external I/O and redundancy systems.

Application Level Software

There are three main application software under PASS which are briefly described below

1. GN&C(Guidance Navigation and control)- The GNC performs critical tasks related to determining vehicle position , velocity and attitude. This uses a multi frequency cyclic closed loop architecture. Every 40 ms critical flight control processing takes place in minor cycles. Tasks are designated frequencies based on their priorities.

2. Vehicle systems management (SM)- Provides the crew with monitoring capabilities. It is also responsible for fault detection and alerting the crew. The SM also allows limited modifications to data tables, reconfiguration of telemetry formats, and control of payload operations and the remote manipulator system
3. Vehicle Checkout (VCO)- Supports testing under the control of ground staff or crew. A unique aspect of the TCS Test Control Supervisor which allows the crew to create custom test sequences.

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

Multi-frequency executive (MFE) is a non-preemptive way of formulating real-time systems.The main idea is to characterize the services which are higher priority to run with a higher frequency. The GN&C module (guidance navigation and control) block of the PASS system uses an MFE architecture.

The higher priority tasks related to basic flight controls are higher priority and hence run at a frequency of 25 Hz compared to lower priority tasks such as display updates which run only at 0.25 Hz. A unique feature of the executive is a ***table driven*** dispatch approach . This technique enables dynamic scheduling of principal functions as required by major modules through a module called the DTU or dispatcher table update module.

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.

Advantages of MFE

1. Since MFE policies do not involve any kind of preemption there is no overhead related to context switching from one service to another. All the services once initiated will run to completion.
2. MFEs and cyclic executives are generally more deterministic compared to other Real time scheduling policies such as RM
3. If the set of services are well defined and "**not**" subject to change MFE(s) are simpler to implement with bugs which are easier to resolve due to lack of multiple threads.
4. No requirement for complex Synchronization- Since all the process is done in a single loop there is no need of dealing with issues of multithreading and there is generally a lesser risk of race condition and deadlocks.

Disadvantages of MFE

1. Lack of flexibility- It is very difficult to make changes or add any new services once the system has been designed. It is easier to accommodate changes in modern real time systems.It also makes MFE systems very ***less scalable***
2. In Real time systems which use threading and RM scheduling policies, preemptiveness is a key feature as high priority tasks can take over execution and stop lower priority task.This is however not allowed in MFEs, where all tasks must run to completion.
3. Inefficient CPU utilization is also one of the key disadvantages of MFE based systems. Since there is no preemption if a task is completed earlier there is no way of dynamically scheduling another task thus leading to poor CPU usage.

Question 3 Answers

3. [5 points] Read the paper "Building Safety-Critical Real-Time Systems with Reusable Cyclic Executives" on Canvas. In other embedded systems classes, you may have 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

Overall understanding of paper and key point articulation

The paper discusses the use of cyclic executives for safety-critical real time tasks. It talks about a reusable cyclic executive implemented in ADA-95 based on a generic architecture for real time systems. The system details the uses of major and minor cycles which are predefined hence ensuring a highly deterministic system. To ensure the system is fault tolerant the concept of recovery groups is used. This ensures that at least the system continues to operate in a known degraded state. This approach has found use in avionics and space systems.

Comparison to Linux and RTOS approaches

Cyclic executive programs simply resolve around a "main loop" which processes tasks "every cycle". It is commonly used in systems without operating systems and used with ISR to handle event requests given through interrupts. Operating systems have control over all running user-programs, and can block user-programs any time it desires, so using a cyclic executive will create issues in real-time scenarios. The lack of overseeing the operating system also means that the programmer must handle everything: scheduling, event handling, and task assignment (if the processor is multi-core).

Linux POSIX RT threads in the meanwhile creates new "threads" or tasks through pthreads, which the OS decides when to execute the program. Linux OS allows users to assign priority (or reassign them) to determine which programs get run. In a cyclic executive program, this has to

be done by the program (in the main loop), and since this becomes complex with a growing number of services, cyclic executives are often used for simpler systems with less tasks to handle.

Question 4 Answers

4. Build the feasibility test example code on your Raspberry Pi. a. Compare the feasibility test.

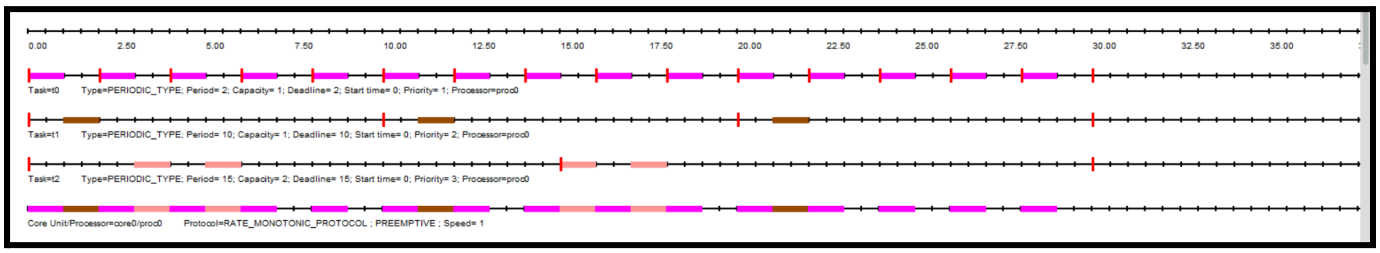
   Brief overview- The code harcodes 5 scenarios with the help of a set of arrays containing WCET(s) or worse case execution times for a set of services along with period. The code then runs a "***Completion Test Feasibility***" on each of these scenarios and checks for scheduling and feasibility using the RMLUB test and prints the result for each scenario Next the code runs a **"Scheduling Point Feasibility**" test for each service and again prints the results. These tests are done for the Rate monotonic policy.

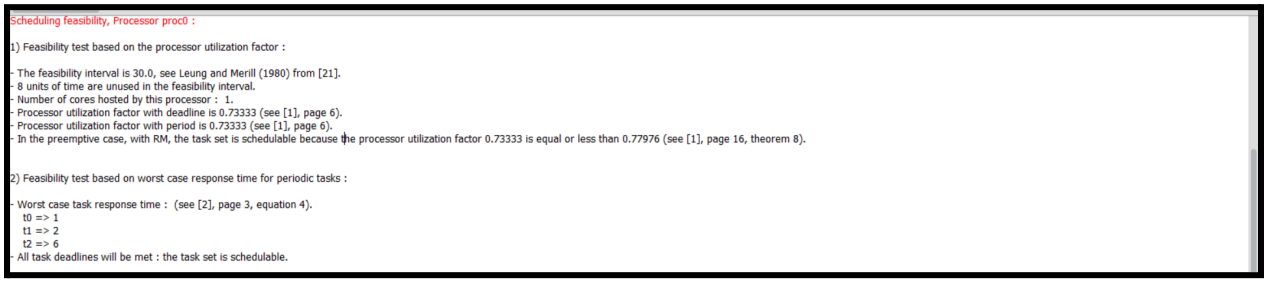
a. Compare the feasibility tests provided by the example code to analysis using Cheddar for the first 5 examples (0-4).
   All source code is linked in *https://github.com/nasa7792/RTES*

Example 0

Time periods=[2,10,15]
WCET=[1,1,2]
LCM=30

Cheddar results





Code results

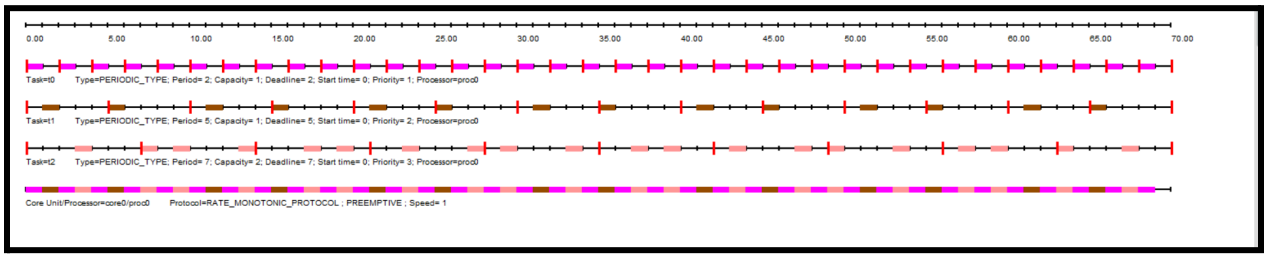
```
nalin@raspberrypi:~/Desktop/Exercise2Files/FeasibilityExampleCode $ ./feasibility_tests
******** Completion Test Feasibility Example
Ex-0 U=73.33% (C1=1, C2=1, C3=2; T1=2, T2=10, T3=15; T=D): CT test FEASIBLE
for 3, utility_sum = 0.000000
for 0, wcet=1.000000, period=2.000000, utility_sum = 0.500000
for 1, wcet=1.000000, period=10.000000, utility_sum = 0.600000
for 2, wcet=2.000000, period=15.000000, utility_sum = 0.733333
utility_sum = 0.733333
LUB = 0.779763
RM LUB FEASIBLE



******** Scheduling Point Feasibility Example
Ex-0 U=73.33% (C1=1, C2=1, C3=2; T1=2, T2=10, T3=15; T=D): FEASIBLE
for 3, utility_sum = 0.000000
for 0, wcet=1.000000, period=2.000000, utility_sum = 0.500000
for 1, wcet=1.000000, period=10.000000, utility_sum = 0.600000
for 2, wcet=2.000000, period=15.000000, utility_sum = 0.733333
utility_sum = 0.733333
LUB = 0.779763
RM LUB FEASIBLE
```

Conclusion- Code results match cheddar results. The service **set is feasible and schedulable**. It passes the completion test and Scheduling point test.

Example 1

Time periods=[2,5,7]
WCET=[1,1,2]
LCM=70

Cheddar results




```
Scheduling simulation, Processor proc0 :

- Number of context switches :  68
- Number of preemptions :  10

- Task response time computed from simulation :
   t0 => 1/worst
   t1 => 2/worst
   t2 => 8/worst ,  missed its deadline (absolute deadline =  7 ; completion time =  8)
- Some task deadlines will be missed : the task set is not schedulable.
Scheduling feasibility, Processor proc0 :

1) Feasibility test based on the processor utilization factor :

- The feasibility interval is 70.0, see Leung and Merill (1980) from [21].
- 1 units of time are unused in the feasibility interval.
- Number of cores hosted by this processor :  1.
- Processor utilization factor with deadline is 0.98571 (see [1], page 6).
- Processor utilization factor with period is 0.98571 (see [1], page 6).
- In the preemptive case, with RM, we cannot prove that the task set is schedulable because the processor utilization factor 0.98571 is more than 0.77976 (see [1], page 16, theorem 8).

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

- Worst case task response time :  (see [2], page 3, equation 4).
   t0 => 1
   t1 => 2
   t2 => 8,  missed its deadline (deadline =  7)
- Some task deadlines will be missed : the task set is not schedulable.
```

Code results

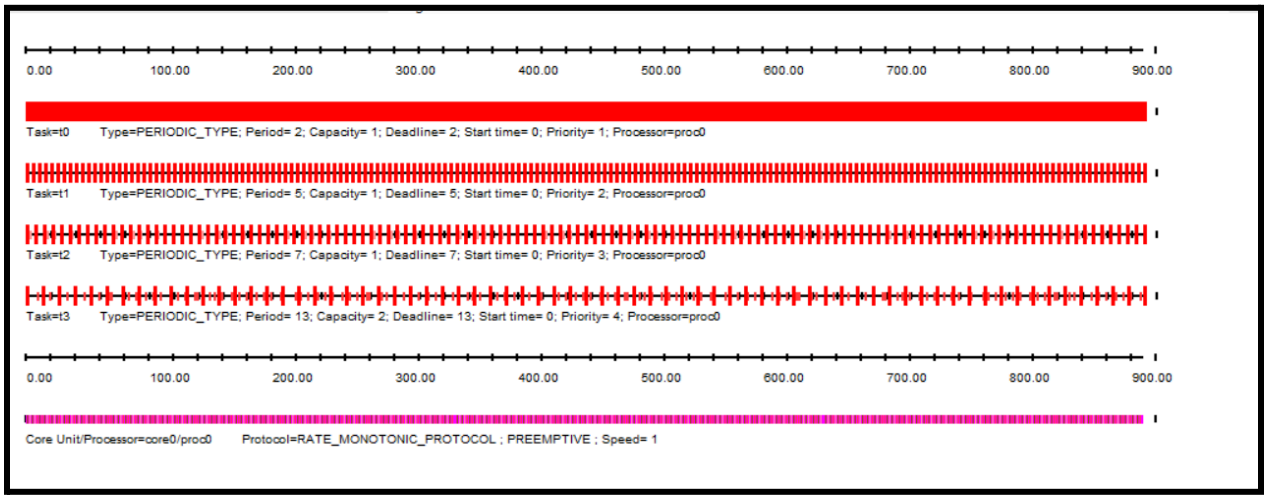
```
nalin@raspberrypi:~/Desktop/Exercise2Files/FeasibilityExampleCode $ ./feasibility_tests
******** Completion Test Feasibility Example
Ex-1 U=98.57% (C1=1, C2=1, C3=2; T1=2, T2=5, T3=7; T=D): INFEASIBLE
for 3, utility_sum = 0.000000
for 0, wcet=1.000000, period=2.000000, utility_sum = 0.500000
for 1, wcet=1.000000, period=5.000000, utility_sum = 0.700000
for 2, wcet=2.000000, period=7.000000, utility_sum = 0.985714
utility_sum = 0.985714
LUB = 0.779763
RM LUB INFEASIBLE



******** Scheduling Point Feasibility Example
Ex-1 U=98.57% (C1=1, C2=1, C3=2; T1=2, T2=5, T3=7; T=D): INFEASIBLE
for 3, utility_sum = 0.000000
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=2.000000, period=7.000000, utility_sum = 0.985714
utility_sum = 0.985714
LUB = 0.779763
```

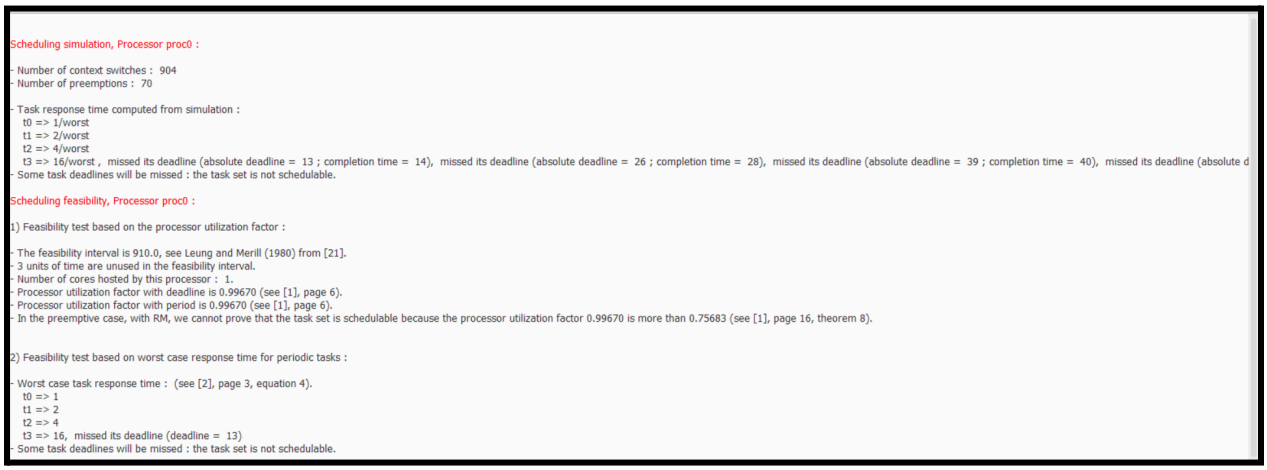
Conclusion- Code results match cheddar results. The service **set is infeasible and not schedulable**. It fails the completion test and Scheduling point test. Task 2 misses its deadline.

Example 2

Time periods=[2,5,7,13]
WCET=[1,1,1,2]
LCM=910

Cheddar results





Code results

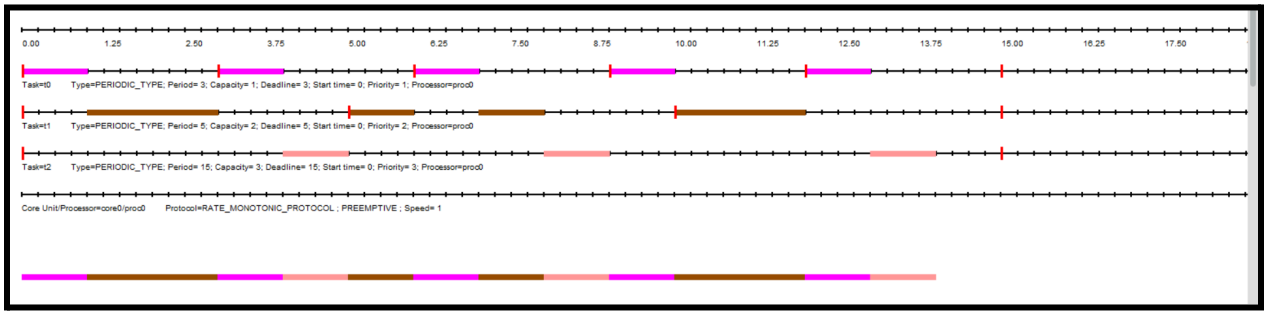
```
******** Completion Test Feasibility Example
Ex-2 U=99.67% (C1=1, C2=1, C3=1, C4=2; T1=2, T2=5, T3=7, T4=13; T=D): INFEASIBLE
for 4, utility_sum = 0.000000
for 0, wcet=1.000000, period=2.000000, utility_sum = 0.500000
for 1, wcet=1.000000, period=5.000000, utility_sum = 0.700000
for 2, wcet=1.000000, period=7.000000, utility_sum = 0.842857
for 3, wcet=2.000000, period=13.000000, utility_sum = 0.996703
utility_sum = 0.996703
LUB = 0.756828
RM LUB INFEASIBLE



******** Scheduling Point Feasibility Example
Ex-2 U=99.67% (C1=1, C2=1, C3=1, C4=2; T1=2, T2=5, T3=7, T4=13; T=D): INFEASIBLE
for 4, utility_sum = 0.000000
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
utility_sum = 0.996703
LUB = 0.756828
RM LUB INFEASIBLE
```

Conclusion- Code results match cheddar results. The service **set is infeasible and not schedulable**. It fails the completion test and Scheduling point test. Task 3 misses its deadline.

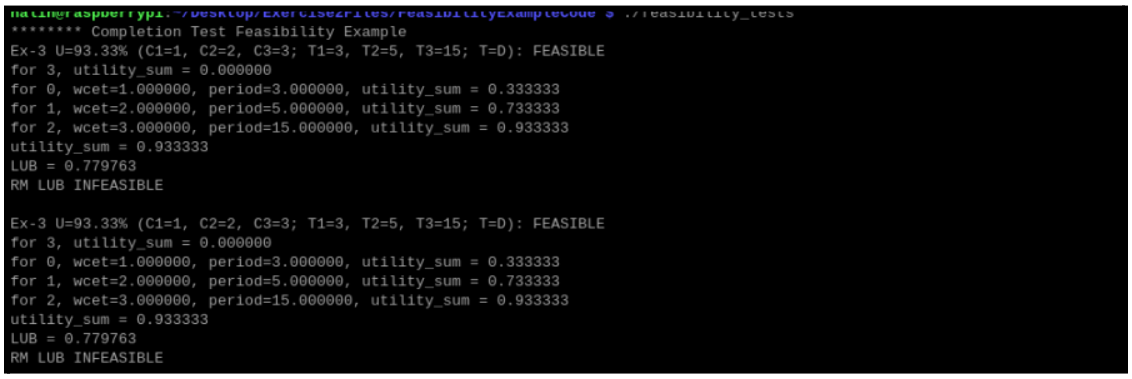
Example 3

Time periods=[3,5,15]
WCET=[1,2,3]
LCM=15

Cheddar results



Code results



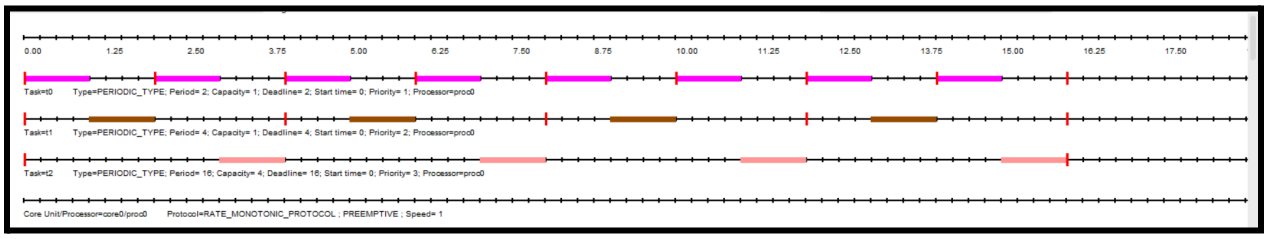
Conclusion- Code results match cheddar results. The service fails the RMLUB test however since **no deadline** is missed the schedule is still considered feasible.

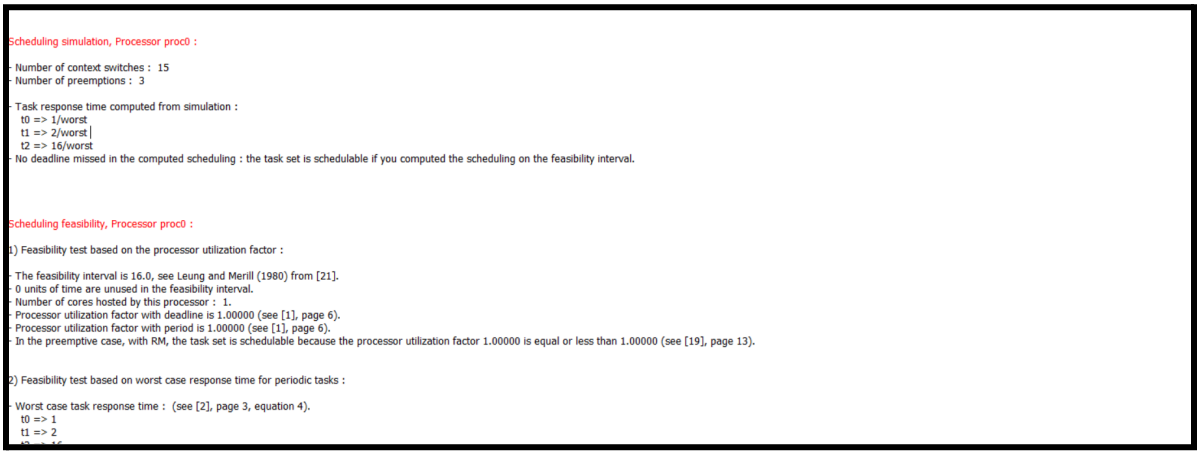
Example 4

Time periods=[2,4,16]
WCET=[1,1,4]
LCM=16

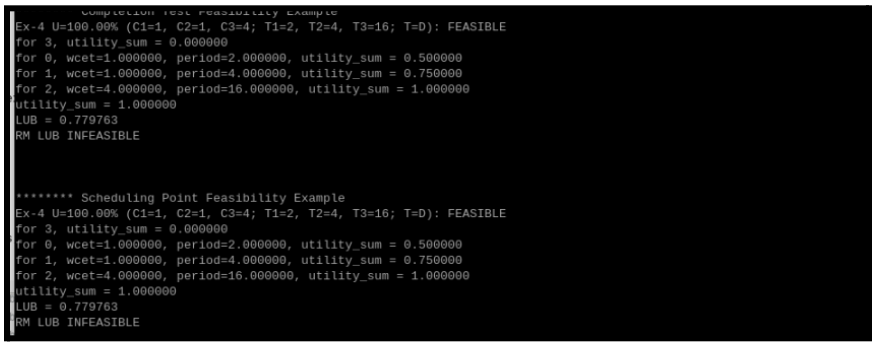
This is the case of harmonic RM so even if utilization is 100 % this scenario would be schedulable.

Cheddar results





Code results



Conclusion- Code results **do not match** cheddar results. The service set should be RM infeasible since the value of utilization is above the RMLUB. However, cheddar output suggests that it has passed the RMLUB test which is incorrect.

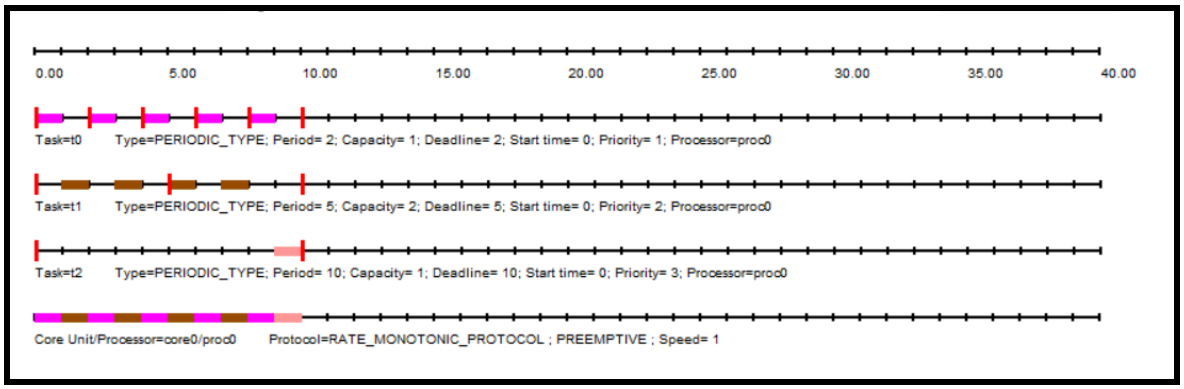
b. Now, implement the remaining examples [5 more, 5-9] that we reviewed in class (and on Canvas) by modifying the example code to include the other examples. Complete analysis for all three policies using Cheddar (RM, EDF, LLF) and by adding EDF and LLF feasibility tests to the code, except for example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

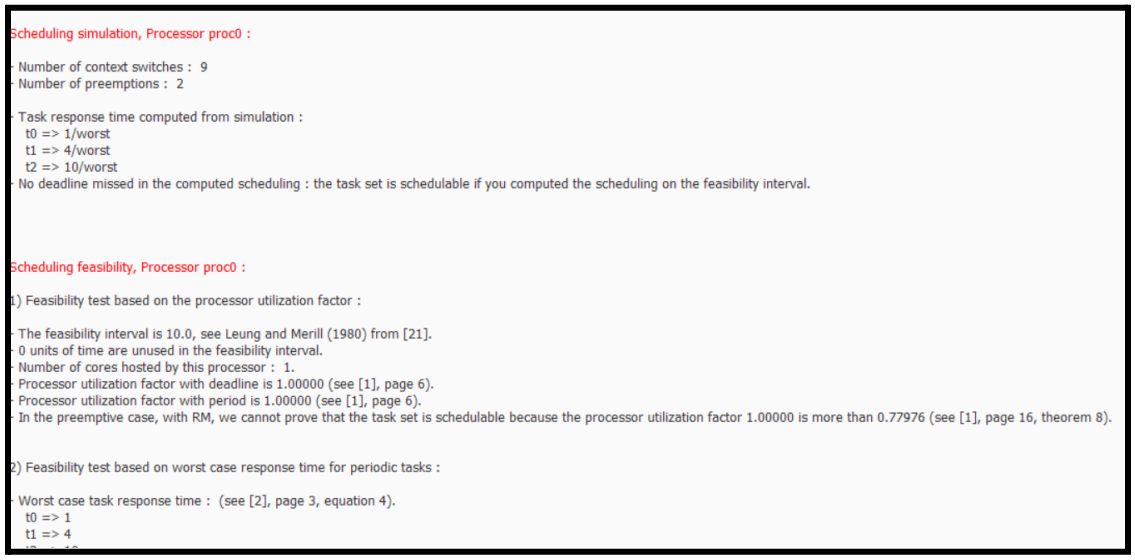
the RM LUB and scheduling-point/completion-test algorithms, referred to as “Worst Case Analysis”

Example 5

Time periods=[2,5,10]
WCET=[1,2,1]
LCM=10

Rate monotonic Cheddar results-



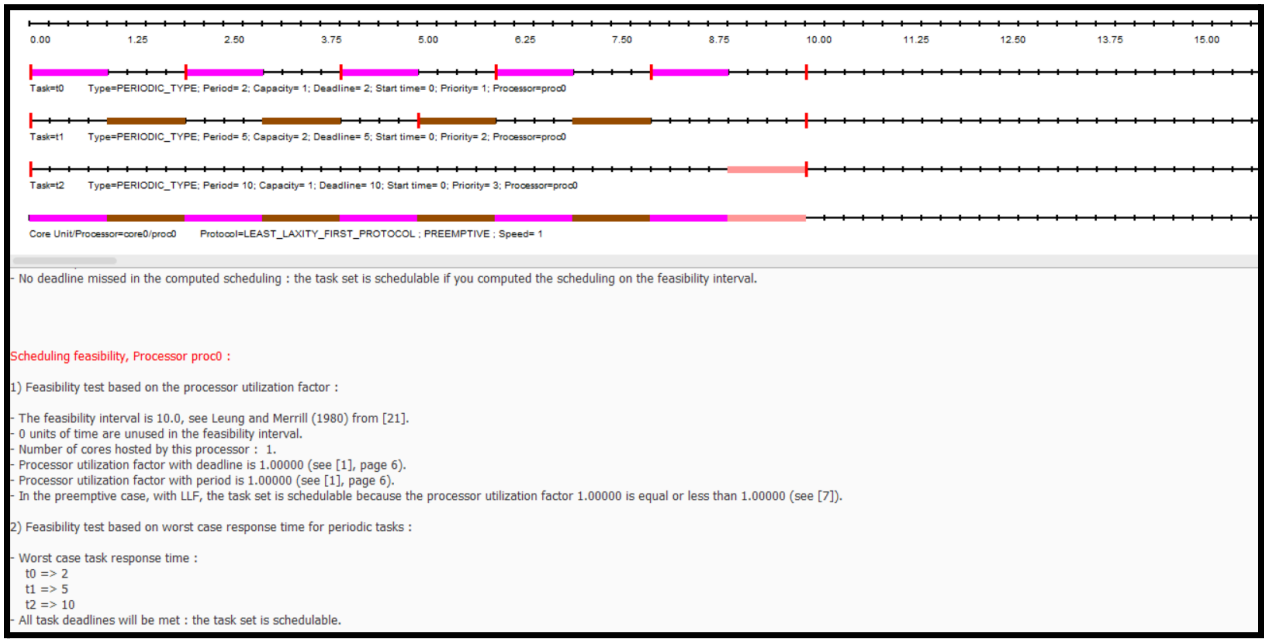


Code results

```
******** Completion Test Feasibility Example
Ex-5 U=100.00%  (C1=1, C2=2, C3=1; T1=2, T2=5, T3=10; T=D): FEASIBLE
for 3, utility_sum = 0.000000
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
utility_sum = 1.000000
LUB = 0.779763
RM LUB INFEASIBLE

******** Scheduling Point Feasibility Example
Ex-5 U=100.00%  (C1=1, C2=2, C3=1; T1=2, T2=5, T3=10; T=D): FEASIBLE
for 3, utility_sum = 0.000000
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
utility_sum = 1.000000
LUB = 0.779763
RM LUB INFEASIBLE
```

EDF cheddar results-

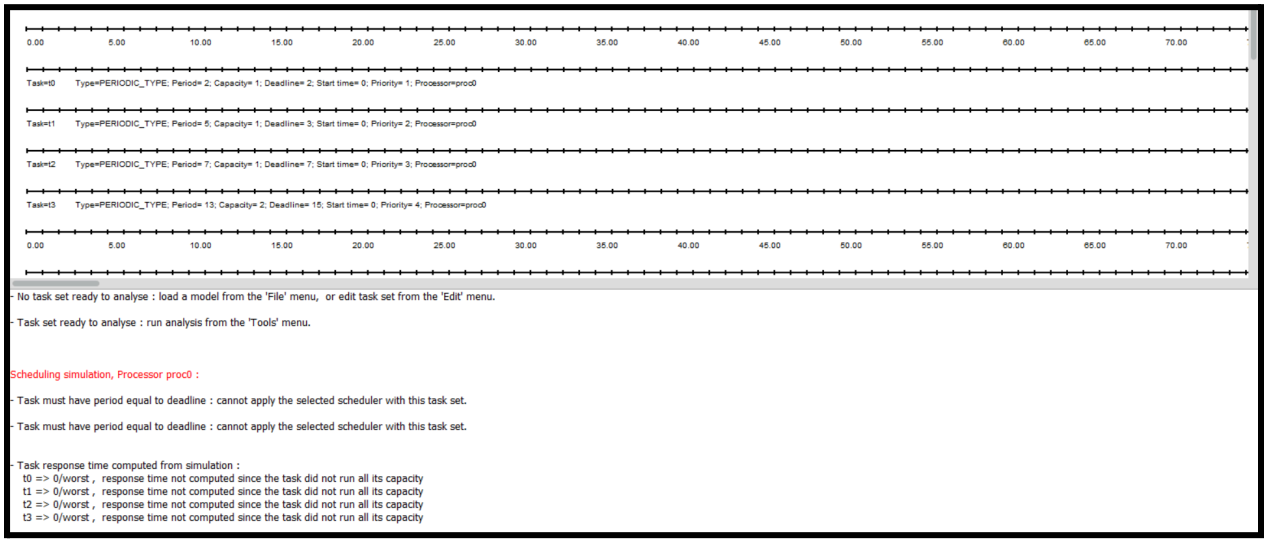
LLF Cheddar results



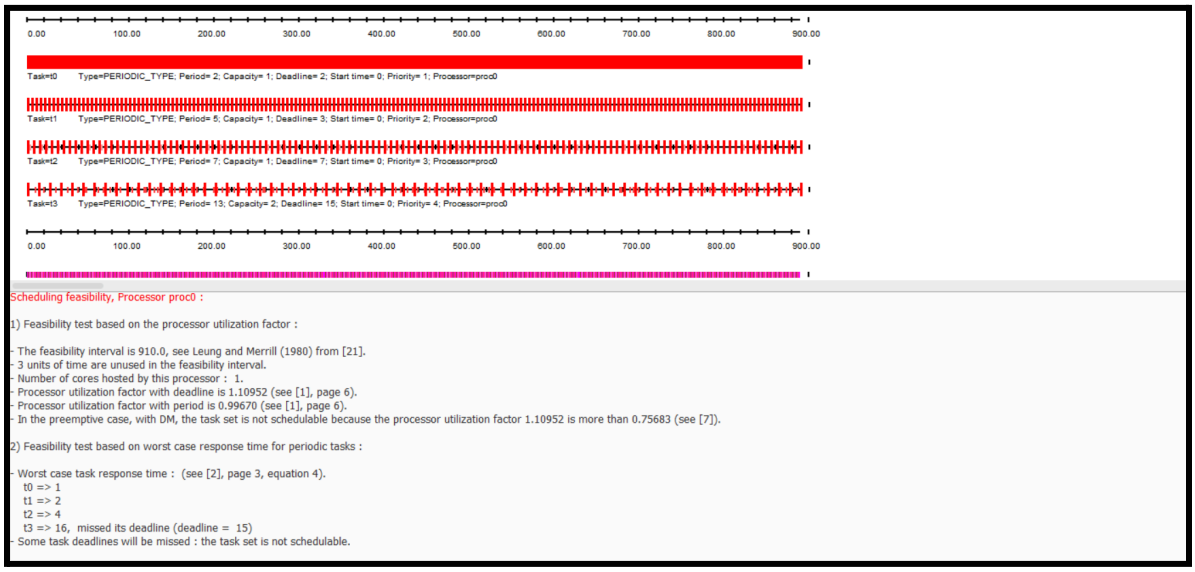
Conclusion - The task fails the RMLUB but it does not miss any deadlines. Both EDF and LLF policies are feasible.

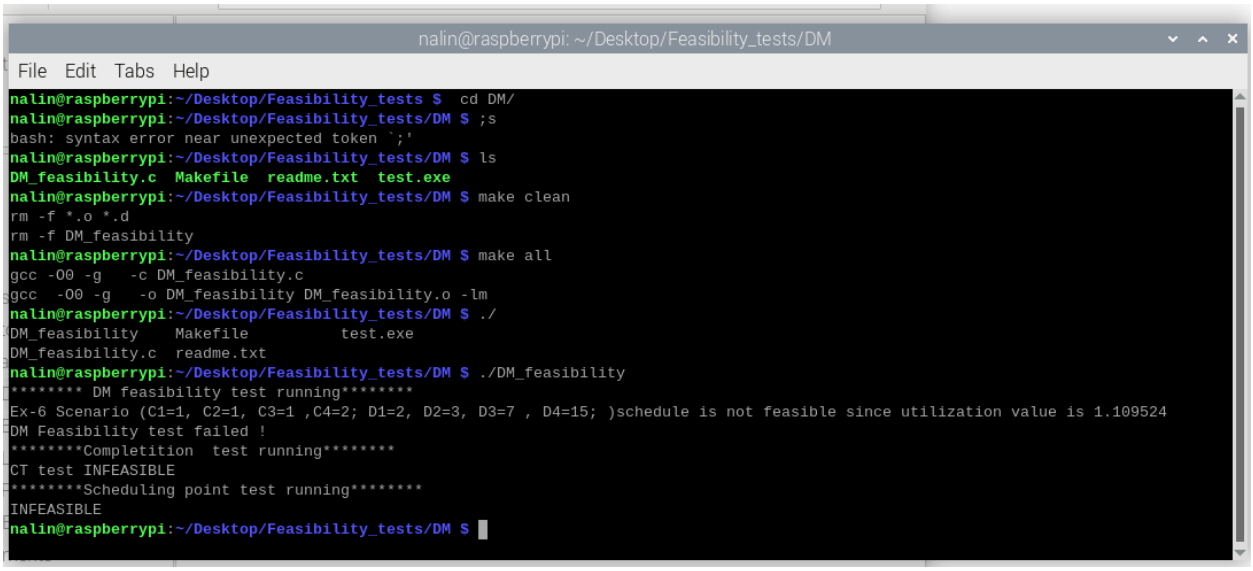
Example 6

Time periods=[2,5,7,13]
Deadline = [2,3,7,15]
WCET=[1,1,1,2]
LCM=910
Deadline is not same as period so we cannot use RM policy
Rate monotonic Cheddar results-



Deadline Monotonic Results

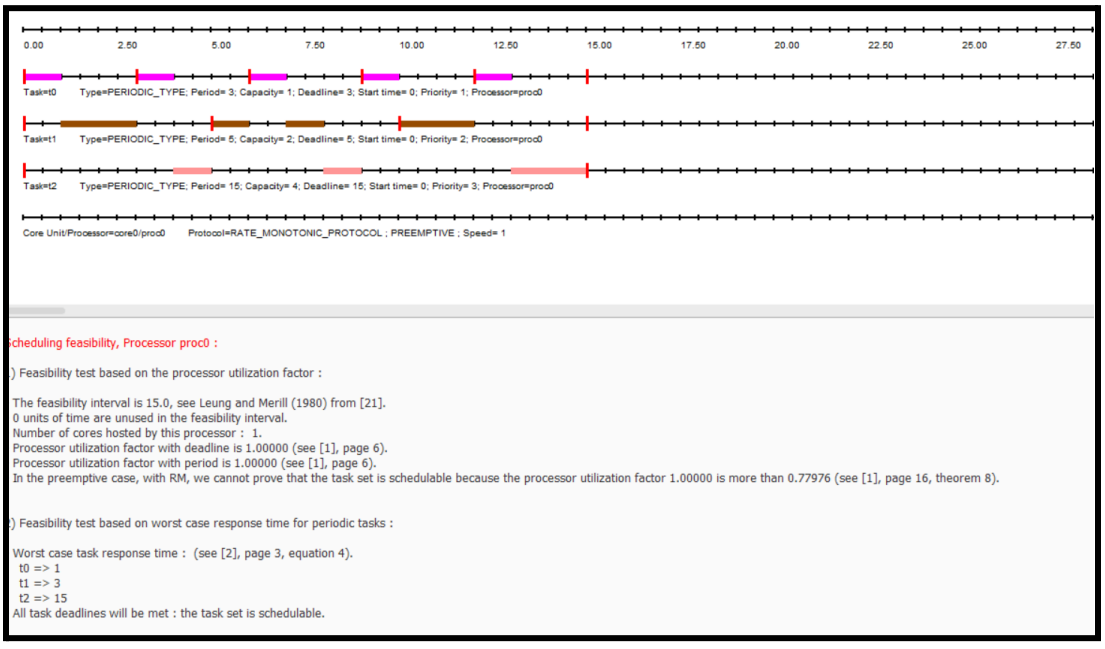




Conclusion - The task is not applicable for RM policy since the deadline is not the same as the period.

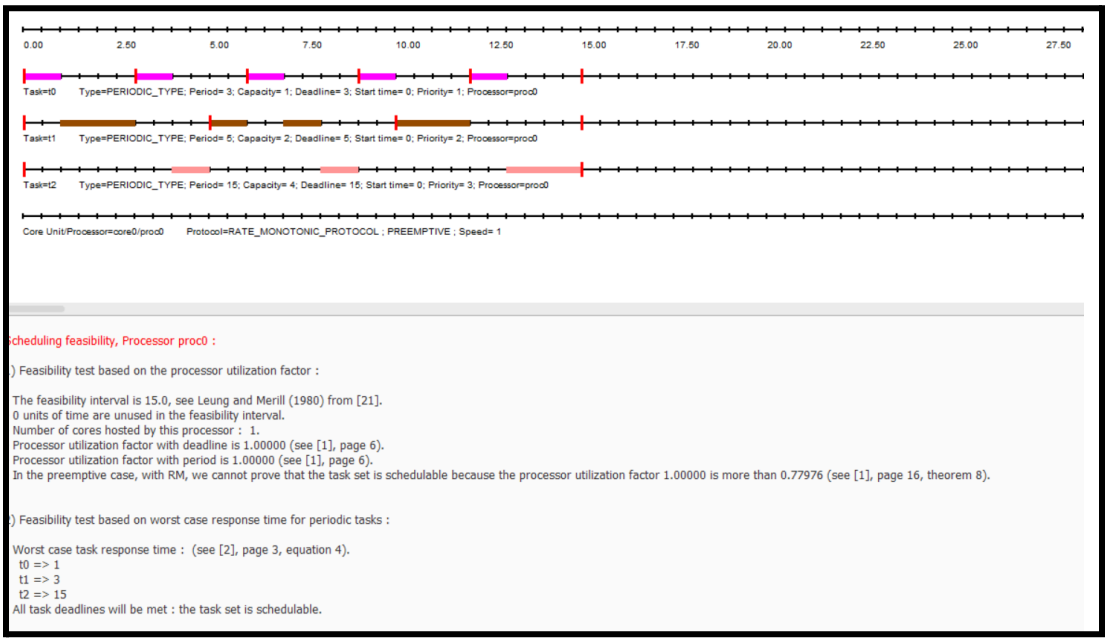
Example 7

Time periods=[3,5,15]
WCET=[1,2,4]
LCM=15
Rate monotonic Cheddar results-



```
root@raspberrypi:/home/nalin/Desktop/res/Exercise2Files/FeasibilityExampleCode# ./feasibility_tests
******** Completion Test Feasibility Example
Ex-7 U=100.00%  (C1=1, C2=2, C3=4; T1=3, T2=5, T3=15; T=D): FEASIBLE
for 3, utility_sum = 0.000000
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
utility_sum = 1.000000
LUB = 0.779763
RM LUB INFEASIBLE

******** Scheduling Point Feasibility Example
Ex-7 U=100.00%  (C1=1, C2=2, C3=4; T1=3, T2=5, T3=15; T=D): FEASIBLE
for 3, utility_sum = 0.000000
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
utility_sum = 1.000000
LUB = 0.779763
RM LUB INFEASIBLE
```

EDF Cheddar Results

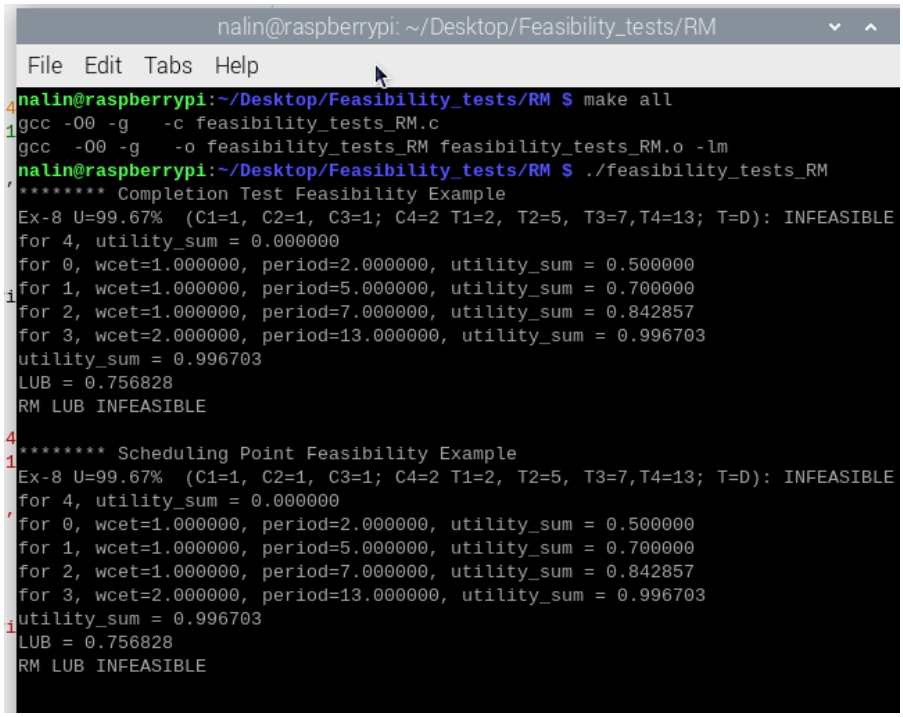


LLF Cheddar Results

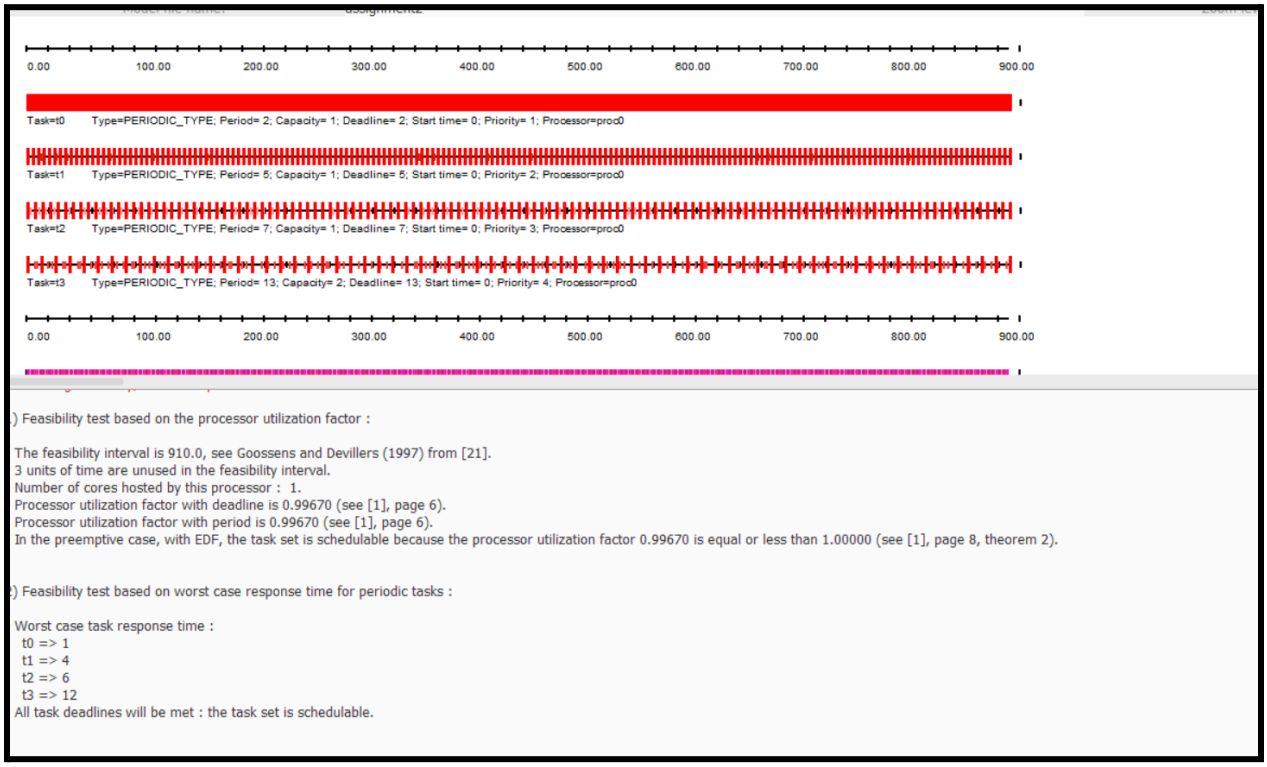
Conclusion - Service set fails RMLUB but does not miss any deadlines under RM. Feasible under LFF and EDF.

Example 8

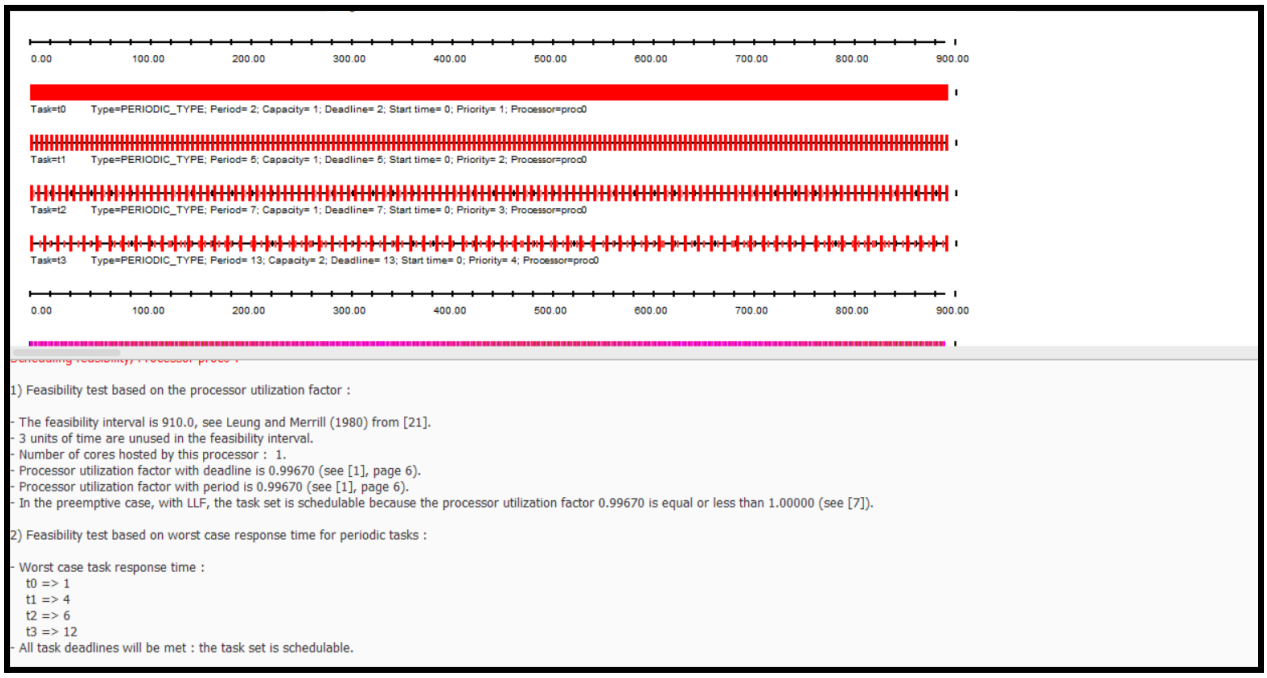
Time periods=[2,5,7,13]
WCET=[1,1,1,2]
LCM=910
Rate monotonic Cheddar results



EDF Cheddar Results



LLF Cheddar Results



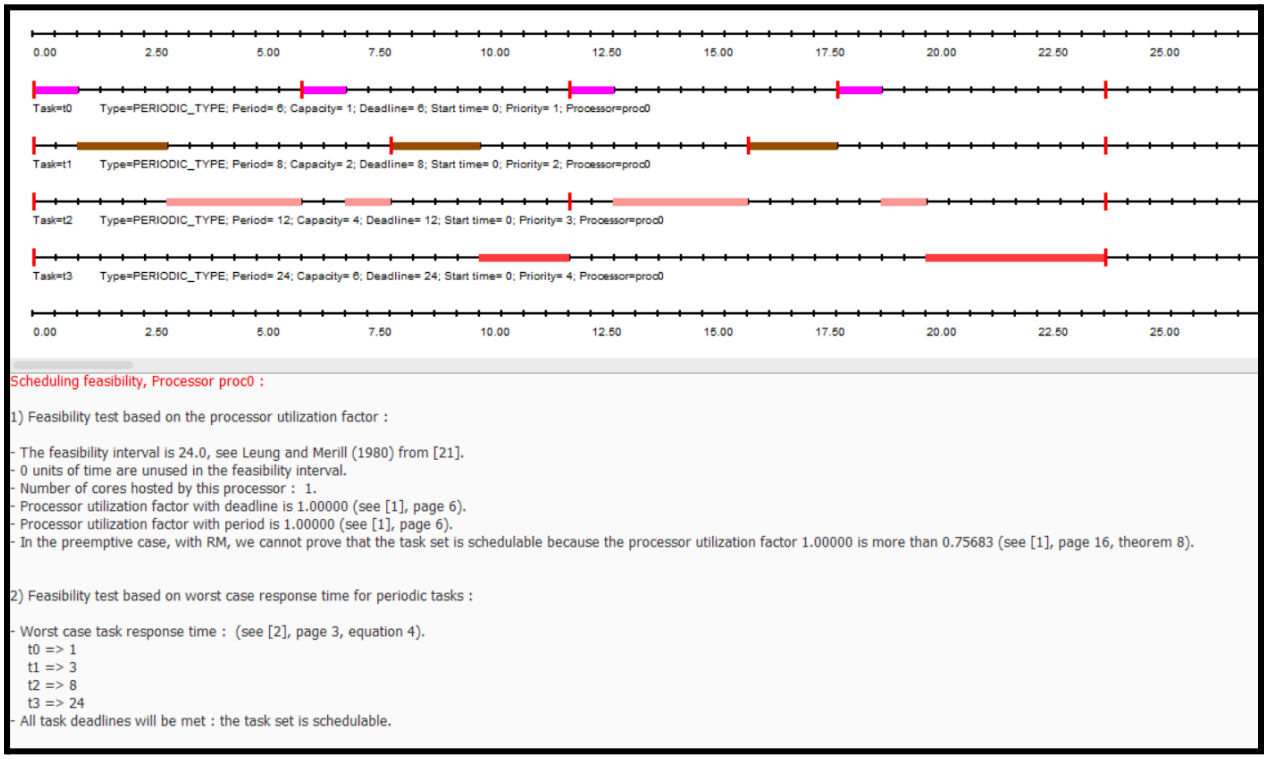
Conclusion - RMLUB test fails and the task set misses a few deadlines. However the task is feasible under LLF or EDF policy.

Why does RM fail and EDF/LLF works?

Under RM highest priority is given to tasks with the highest frequency. Here in this case task3 is having the lowest priority and misses its deadline as it is being preempted by other higher priority tasks.Under Edf task priority is assigned based on the deadlines dynamically and in a sense tasks which are the most “urgent” or have the shortest deadlines are service first. This results in a better cpu utilization and fewer missed deadlines.

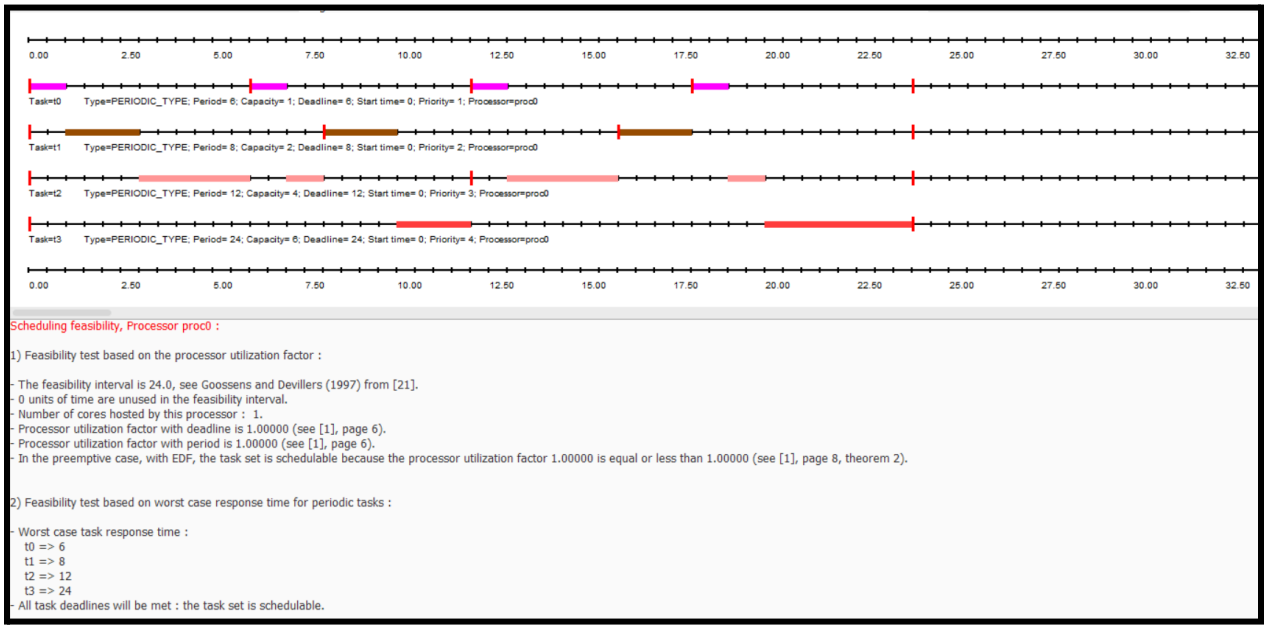
Example 9

Time periods=[6,8,12,24]
WCET=[1,2,4,6]
LCM=24
Rate monotonic Cheddar results-

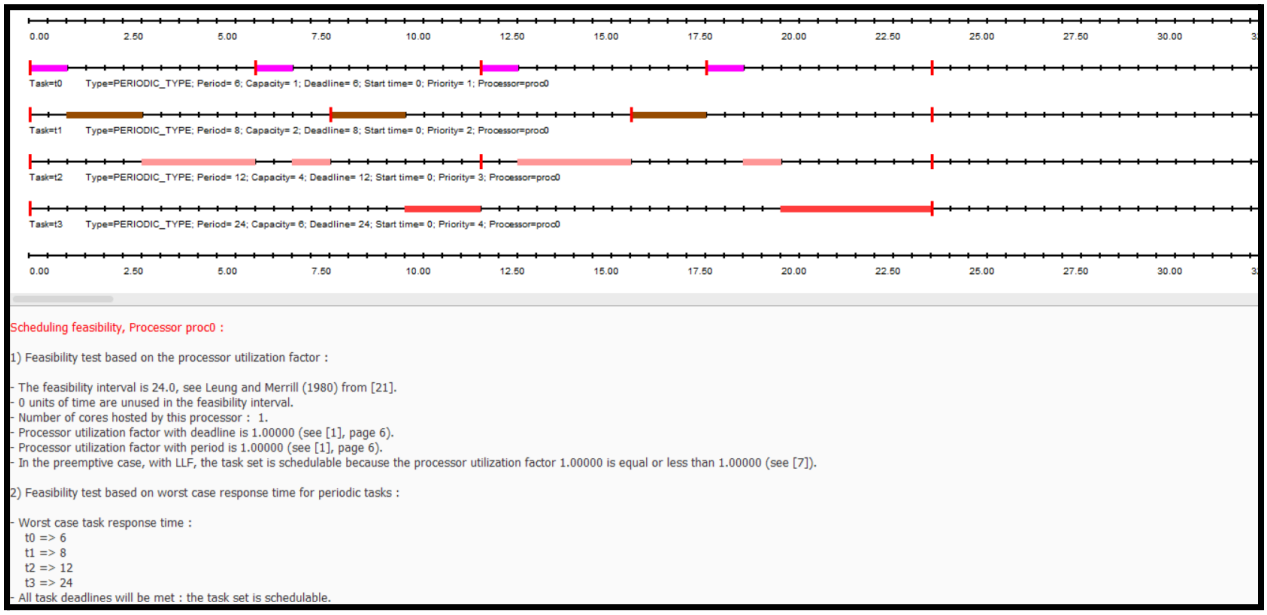


```
Completion Test Feasibility Example
Ex-9 U=100.00%  (C1=1, C2=2, C3=4; C4=6 T1=6, T2=8, T3=12,T4=24; T=D): FEASIBLE
for 4, utility_sum = 0.000000
for 0, wcet=1.000000, period=6.000000, utility_sum = 0.166667
for 1, wcet=2.000000, period=8.000000, utility_sum = 0.416667
for 2, wcet=4.000000, period=12.000000, utility_sum = 0.750000
for 3, wcet=6.000000, period=24.000000, utility_sum = 1.000000
utility_sum = 1.000000
LUB = 0.756828
RM LUB INFEASIBLE


Ex-9 U=100.00%  (C1=1, C2=2, C3=4; C4=6 T1=6, T2=8, T3=12,T4=24; T=D): FEASIBLE
for 4, utility_sum = 0.000000
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
utility_sum = 1.000000
LUB = 0.756828
RM LUB INFEASIBLE
```

EDF Cheddar Results



LLF Cheddar Results



Conclusion - Service set fails RMLUB but does not miss any deadlines under RM. Feasible under LFF and EDF. (for the same reason as explained under example 8)

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

All cases under EDF and LLF match the code of utilization test and cheddar

.

Interestingly- There appears to be some discrepancy in example4 where cheddar results do not match with code. (explained under example 4 conclusion)

Overall results

Example Number	RMLUB TEST	Completion test	Scheduling point test	Cheddar test (RM)	Utilization (EDF& LLF)	Cheddar EDF	Cheddar LLF
Example	Feasible	Feasible	Feasible	Schedulable	*N/A*	*N/A*	*N/A*

0							
Example 1	Infeasible	Infeasible	Infeasible	Not Schedulable	*N/A*	*N/A*	*N/A*
Example 2	Infeasible	Infeasible	Infeasible	Not Schedulable	*N/A*	*N/A*	*N/A*
Example 3	Infeasible	Feasible	Feasible	Schedulable	*N/A*	*N/A*	*N/A*
Example 4	Infeasible	Feasible	Feasible	Schedulable	*N/A*	*N/A*	*N/A*
Example 5	Infeasible	Feasible	Feasible	Schedulable	Feasible	Schedulable	Schedulable
Example 6	*N/A*	*N/A*	*N/A*	Not Schedulable (DM)	*N/A*	*N/A*	*N/A*
Example 7	Infeasible	Feasible	Feasible	Schedulable	Feasible	Schedulable	Schedulable
Example 8	Infeasible	Infeasible	Infeasible	Not Schedulable	Feasible	Schedulable	Schedulable
Example 9	Infeasible	Feasible	Feasible	Schedulable	Feasible	Schedulable	Schedulable

Overall EDF Code results

```
nalin@raspberrypi:~/Desktop/Feasibility_tests/EDF $ ./edf_feasibility
******** EDF feasibility test running********
Ex-5 U=100.00%  (C1=1, C2=2, C3=1; T1=2, T2=5, T3=10; T=D):
simulation period is 10
EDF Feasibility test passed !


Ex-7 U=100.00%  (C1=1, C2=2, C3=4; T1=3, T2=5, T3=15; T=D):
simulation period is 15
EDF Feasibility test passed !


Ex-8 U=99.67%  (C1=1, C2=1, C3=1; C4=2 T1=2, T2=5, T3=7,T4=13; T=D):
simulation period is 910
EDF Feasibility test passed !


Ex-9 U=100.00%  (C1=1, C2=2, C3=4; C4=6 T1=6, T2=8, T3=12,T4=24; T=D):
simulation period is 24
EDF Feasibility test passed !
nalin@raspberrypi:~/Desktop/Feasibility_tests/EDF $
```

Overall LLF Code results

```
nalin@raspberrypi: ~/Desktop/Feasibility_tests/LLF
File  Edit  Tabs  Help
Ex-8 U=99.67%  (C1=1, C2=1, C3=1; C4=2 T1=2, T2=5, T3=7,T4=13; T=D): LLF Feasibility test failed !


Ex-9 U=100.00%  (C1=1, C2=2, C3=4; C4=6 T1=6, T2=8, T3=12,T4=24; T=D): LLF Feasibility test failed !
nalin@raspberrypi:~/Desktop/Feasibility_tests/LLF $ make clean
rm -f *.o *.d
rm -f llf_feasibility
nalin@raspberrypi:~/Desktop/Feasibility_tests/LLF $ make all
gcc -O0 -g   -c llf_feasibility.c
gcc  -O0 -g   -o llf_feasibility llf_feasibility.o -lm
nalin@raspberrypi:~/Desktop/Feasibility_tests/LLF $ ./llf_feasibility
******** LLF feasibility test running********
Ex-5 U=100.00%  (C1=1, C2=2, C3=1; T1=2, T2=5, T3=10; T=D): LLF Feasibility test passed !


Ex-7 U=100.00%  (C1=1, C2=2, C3=4; T1=3, T2=5, T3=15; T=D): LLF Feasibility test passed !


Ex-8 U=99.67%  (C1=1, C2=1, C3=1; C4=2 T1=2, T2=5, T3=7,T4=13; T=D): LLF Feasibility test passed !


Ex-9 U=100.00%  (C1=1, C2=2, C3=4; C4=6 T1=6, T2=8, T3=12,T4=24; T=D): LLF Feasibility test passed !
nalin@raspberrypi:~/Desktop/Feasibility_tests/LLF $
```

Question 5 Answers

5. [24 points] Read Chapter 3 of the textbook again.

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.

First assumption made by Liu and Layland's paper is that service requests are periodic. This is a reasonable assumption given that many systems use polling techniques to constantly process input. This assumption makes sense for some systems such as live-video/audio processing units, but does not apply to all systems such as an airbag deployment system. Not all requests are periodic, but may need to be handled multiple times at different deadlines/frequencies.

Second assumption in the paper is that all service requests are independent of each other. This assumes that there isn't an order in which tasks have to be finished, and output of a service request does not depend on response from another request. This is reasonable for most systems, since any task which has dependencies on another request can be "treated" as one request. However, some of these dependent requests may have different deadlines for each task (rather than one deadline as a whole), so this assumption is not valid for systems in which previous outputs need to be stored for the next response. For other real-life systems, this is a reasonable assumption when designing a scheduler.

Third assumption is that run-time for all tasks is constant, and worst-case execution time is known for each service. This is a reasonable assumption to make for a real-time system, but a very hard task to perform in real-life, and therefore idealistic. Modern day computer architecture and operating systems are much more complex than before. Compiler optimizations modify code and often perform branch predictions, which may cause faults. In addition to this, some operating systems (such as Linux for soft-real time) do not have clear scheduling parameters since the kernel can block any user-space applications, making it much harder to predict run-time. In addition to this, due to data cache misses and context switch times (from scheduling), finding the theoretical worst-case run-time is a very difficult task. So in reality, worst-case execution time is estimated from testing, which may not work in all real-life situations.

First constraint with the Rate-Monotonic scheduling is that the deadline of a request is the same as the period by definition. This simply means that a task needs to be processed before the next request is sent. This is also only applicable to specific systems, and oftentimes for tasks with long periods, the deadline of the service is before the next request. For example, a motor for a car can't be controlled immediately, and additional time is required for the control system. Even if the self-driving system request is made every 50 milliseconds, the data will need to be processed before the 50ms mark since controlling the motor is not instantaneous. Deadline of a request is not always the same as the time until the next request in many real life systems.

Second constraint is that all tasks have fixed priority. For Rate-Monotonic algorithm this means that frequency of the service stays constant. This is a realistic constraint for many systems which are meant to provide service to a constant stream of inputs, such as video processing units. Systems which require change in polling rates or service frequency still do exist - for example, a

user may want to change the bit-rate of video depending on the internet speed. However, many systems with tasks with changing request periods are usually more complex, and won't likely use Rate Monotonic policy, so this is a realistic constraint.

Third constraint is that scheduling is preemptive, but runs-to-completion without any requests. This is a realistic constraint, as blocking a higher priority task to make a lower priority task does not make sense when it comes to scheduling. Without preemption, it is much harder to meet all deadlines given a limited number of resources. The only thing that a programmer needs to watch out for preemptive scheduling is that there would be more overheads from context switching. Given that modern day operating systems do this constantly, this is a very realistic constraint.

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.

Key Derivation Step of RM LUBs:

1. Definition of "critical time zone": the time between T2 and last deadline of S1
    - Critical time = T2 - Floor(T2/T1) by definition
    - For Case 1 (critical time does fit), C1 <= T2 - Floor(T2/T1)
    - For Case 2 (critical time doesn't fit), C1 > T2 - Floor(T2/T1)
    - For harmonic requests, critical time is zero

2. Deriving Utilization Rate Equation: explaining why they monotonically increase/decrease
    - U = (C1/T1) + (C2/T2) for both cases
    - For Case 1, We use the maximum value of C2 to meet deadline, which is C2 = T2 - C1 * Ceiling(T2/T1)

With substitution for Case 1, we get

At this point, let's plug the expression for C2 in Equation 3.2 into Equation 3.3, which creates Equation 3.4:

$$U = \frac{C_1}{T_1} + \frac{\left[T_2 - C_1\lceil T_2 / T_1 \rceil\right]}{T_2} \quad (3.4)$$

Now, simplify by the following algebraic steps:

$$U = \frac{C_1}{T_1} + \frac{T_2}{T_2} + \frac{\left[-C_1\lceil T_2 / T_1 \rceil\right]}{T_2} \text{ (pull out } T_2 \text{ term)}$$

$$U = \frac{C_1}{T_1} + 1 + \frac{\left[-C_1\lceil T_2 / T_1 \rceil\right]}{T_2} \text{ (note that } T_2 \text{ term is 1)}$$

$$U = 1 + C_1\left[(1/T_1) - \frac{\lceil T_2 / T_1 \rceil}{T_2}\right] \text{ (combine } C_1 \text{ terms)}$$

This gives you Equation 3.5:

$$U = 1 + C_1\left[(1/T_1) - \frac{\lceil T_2 / T_1 \rceil}{T_2}\right] \quad (3.5)$$

- With C1 increasing, Utilization actually decreases, since T2 > T1 for RM policy. The term Ceiling(T2/T1) / T2 is always between 1 and 2, for T >= 1. We can set T1 = 1 (shorter time deadline is 1 time unit), and this is monotonically decreasing

- For Case 2, the maximum value of C2 is different. Only the time period of T1 * Floor(T2/T1) matters because T1 is always running at critical time zone
  C2 = T1 * Floor(T2/T1) - C1 * Floor (T2/T1)

$$C_2 = T_1\lfloor T_2 / T_1 \rfloor - C_1\lfloor T_2 / T_1 \rfloor \quad (3.7)$$

Substituting Equation 3.7 into the utility Equation 3.3 again as before, we get

Equation 3.8:

$$U = \frac{C_1}{T_1} + \frac{\left[T_1\lfloor T_2 / T_1 \rfloor - C_1\lfloor T_2 / T_1 \rfloor\right]}{T_2} \quad (3.8)$$

Now simplifying by the following algebraic steps:

$$U = (T_1 / T_2)\lfloor T_2 / T_1 \rfloor + \frac{C_1}{T_1} + \frac{\left[-C_1\lfloor T_2 / T_1 \rfloor\right]}{T_2} \text{ (separating terms)}$$

$$U = (T_1 / T_2)\lfloor T_2 / T_1 \rfloor + C_1\left[(1/T_1) - (1/T_2)\lfloor T_2 / T_1 \rfloor\right] \text{ (pulling out common } C_1 \text{ term)}$$

This gives us Equation 3.9:

$$U = (T_1 / T_2)\lfloor T_2 / T_1 \rfloor + C_1\left[(1/T_1) - (1/T_2)\lfloor T_2 / T_1 \rfloor\right] \quad (3.9)$$

- With C1 increasing, Utilization increases.
  - (1/T2) * floor(T2/T1) <= 1 for all scenarios where T1 >= 1
  - Let T1 = 1 (time unit = shorter deadline), then, 1/T1 = 1, and since (1/T2) *floor(T2/T1) <=1, this is monotonically decreasing
- We do not need to worry about T1 != 1, since we can divide all other intervals (C1, C2, T2) by factor of T1 to make T1_new = 1

3. Combining the 2 Equations for different cases

- Case 1 and Case 2 shows different utilization usage with increasing C1 (C2 will decrease by definition of maximum C2)
- We only care about the “higher utilization” of the 2 cases only because the policy with higher utilization will run, so the intersection of the 2 functions is only where it matters
- Intersection only occurs when C1 is equal for both cases!
    - So compute value of C1 (which is equivalent in both cases), and after simplification we get

      Solving for f, we get:

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

      And, plugging f back into U, we get:

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

- And this process can be iteratively done to get the lower-bound of U = n * (2^1/n - 1) where n is number of services

Code appendix.

File- Rate monotonic tests

```
// 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.
// Code modified to add more test cases
//

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

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

// U=0.7333
U32_T ex0_period[] = {2, 10, 15};
U32_T ex0_wcet[] = {1, 1, 2};

// U=0.9857
U32_T ex1_period[] = {2, 5, 7};
U32_T ex1_wcet[] = {1, 1, 2};

// U=0.9967
U32_T ex2_period[] = {2, 5, 7, 13};
U32_T ex2_wcet[] = {1, 1, 1, 2};

// U=0.93
U32_T ex3_period[] = {3, 5, 15};
```

```
U32_T ex3_wcet[] = {1, 2, 3};

// U=1.0
U32_T ex4_period[] = {2, 4, 16};
U32_T ex4_wcet[] = {1, 1, 4};

// example 5
U32_T ex5_period[] = {2, 5, 10};
U32_T ex5_wcet[] = {1, 2, 1};

// example 6 deadline not same as period

U32_T ex6_period[] = {2, 5, 7, 13};
U32_T ex6_wcet[] = {1, 1, 1, 2};
U32_T ex6_deadline[] = {2, 3, 7, 15};

// example 7
U32_T ex7_period[] = {3, 5, 15};
U32_T ex7_wcet[] = {1, 2, 4};

// example 8
U32_T ex8_period[] = {2, 5, 7, 13};
U32_T ex8_wcet[] = {1, 1, 1, 2};

// example 9
U32_T ex9_period[] = {6, 8, 12, 24};
U32_T ex9_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 main(void)
{
    int i;
    U32_T numServices;

    printf("******** Completion Test Feasibility Example\n");

        printf("Ex-0 U=%4.2f\% (C1=1, C2=1, C3=2; T1=2, T2=10, T3=15; T=D): ",
               ((1.0 / 2.0) * 100.0 + (1.0 / 10.0) * 100.0 + (2.0 / 15.0) * 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 (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");

        printf("Ex-1 U=%4.2f\% (C1=1, C2=1, C3=2; T1=2, T2=5, T3=7; T=D): ",
               ((1.0 / 2.0) * 100.0 + (1.0 / 5.0) * 100.0 + (2.0 / 7.0) * 100.0));
        numServices = 3;
        if (completion_time_feasibility(numServices, ex1_period, ex1_wcet, ex1_period) == TRUE)
            printf("FEASIBLE\n");
        else
            printf("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");
    printf("\n");

    printf("Ex-2 U=%4.2f\% (C1=1, C2=1, C3=1, C4=2; T1=2, T2=5, T3=7, T4=13; T=D): ",
           ((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, ex2_period, ex2_wcet, ex2_period) == TRUE)
        printf("FEASIBLE\n");
    else
        printf("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");
    printf("\n");

    printf("Ex-3 U=%4.2f\% (C1=1, C2=2, C3=3; T1=3, T2=5, T3=15; T=D): ",
           ((1.0 / 3.0) * 100.0 + (2.0 / 5.0) * 100.0 + (3.0 / 15.0) * 100.0));
    numServices = 3;
    if (completion_time_feasibility(numServices, ex3_period, ex3_wcet, ex3_period) == TRUE)
        printf("FEASIBLE\n");
    else
        printf("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");
    printf("\n");

    printf("Ex-4 U=%4.2f\% (C1=1, C2=1, C3=4; T1=2, T2=4, T3=16; T=D): ",
           ((1.0 / 2.0) * 100.0 + (1.0 / 4.0) * 100.0 + (4.0 / 16.0) * 100.0));
    numServices = 4;
    if (completion_time_feasibility(numServices, ex4_period, ex4_wcet, ex4_period) == TRUE)
        printf("FEASIBLE\n");
    else
        printf("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");
    printf("\n");

    printf("Ex-5 U=%4.2f%%  (C1=1, C2=2, C3=1; T1=2, T2=5, T3=10; T=D): ",
           ((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, ex5_period, ex5_wcet, ex5_period) == TRUE)
        printf("FEASIBLE\n");
    else
        printf("INFEASIBLE\n");

    if (rate_monotonic_least_upper_bound(numServices, ex5_period, ex5_wcet, ex5_period) == TRUE)
        printf("RM LUB FEASIBLE\n");
    else
```

```
        printf("RM LUB INFEASIBLE\n");
    printf("\n");

    printf("Ex-6 U=%4.2f%%  (C1=1, C2=1, C3=1; C4=2 T1=2, T2=5, T3=7,T4=13; T=D): ",
           ((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, ex6_period, ex6_wcet, ex6_deadline) == TRUE)
        printf("FEASIBLE\n");
    else
        printf("INFEASIBLE\n");

    if (rate_monotonic_least_upper_bound(numServices, ex6_period, ex6_wcet, ex6_period) == TRUE)
        printf("RM LUB FEASIBLE\n");
    else
        printf("RM LUB INFEASIBLE\n");
    printf("\n");

    printf("Ex-7 U=%4.2f%%  (C1=1, C2=2, C3=4; T1=3, T2=5, T3=15; T=D): ",
           ((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, ex7_period, ex7_wcet, ex7_period) == TRUE)
        printf("FEASIBLE\n");
    else
        printf("INFEASIBLE\n");

    if (rate_monotonic_least_upper_bound(numServices, ex7_period, ex7_wcet, ex7_period) == TRUE)
        printf("RM LUB FEASIBLE\n");
    else
        printf("RM LUB INFEASIBLE\n");
    printf("\n");
*/
printf("Ex-8 U=%4.2f%%  (C1=1, C2=1, C3=1; C4=2 T1=2, T2=5, T3=7,T4=13; T=D): ",
       ((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, ex8_period, ex8_wcet, ex8_period) == TRUE)
    printf("FEASIBLE\n");
else
    printf("INFEASIBLE\n");

if (rate_monotonic_least_upper_bound(numServices, ex8_period, ex8_wcet, ex8_period) == TRUE)
    printf("RM LUB FEASIBLE\n");
else
    printf("RM LUB INFEASIBLE\n");
printf("\n");

    printf("Ex-9 U=%4.2f%%  (C1=1, C2=2, C3=4; C4=6 T1=6, T2=8, T3=12,T4=24; T=D): ",
           ((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, ex9_period, ex9_wcet, ex9_period) == TRUE)
        printf("FEASIBLE\n");
    else
        printf("INFEASIBLE\n");

    if (rate_monotonic_least_upper_bound(numServices, ex9_period, ex9_wcet, ex9_period) == TRUE)
        printf("RM LUB FEASIBLE\n");
    else
        printf("RM LUB INFEASIBLE\n");
    printf("\n");

    printf("\n\n");
```

```
printf("******** Scheduling Point Feasibility Example\n");

    printf("Ex-0 U=%4.2f\% (C1=1, C2=1, C3=2; T1=2, T2=10, T3=15; T=D): ",
           ((1.0 / 2.0) * 100.0 + (1.0 / 10.0) * 100.0 + (2.0 / 15.0) * 100.0));
    numServices = 3;
    if (scheduling_point_feasibility(numServices, ex0_period, ex0_wcet, ex0_period) == TRUE)
        printf("FEASIBLE\n");
    else
        printf("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");

    printf("Ex-1 U=%4.2f\% (C1=1, C2=1, C3=2; T1=2, T2=5, T3=7; T=D): ",
           ((1.0 / 2.0) * 100.0 + (1.0 / 5.0) * 100.0 + (2.0 / 7.0) * 100.0));
    numServices = 3;
    if (scheduling_point_feasibility(numServices, ex1_period, ex1_wcet, ex1_period) == TRUE)
        printf("FEASIBLE\n");
    else
        printf("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");
    printf("\n");

    printf("Ex-2 U=%4.2f\% (C1=1, C2=1, C3=1, C4=2; T1=2, T2=5, T3=7, T4=13; T=D): ",
           ((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 (scheduling_point_feasibility(numServices, ex2_period, ex2_wcet, ex2_period) == TRUE)
        printf("FEASIBLE\n");
    else
        printf("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");
    printf("\n");

    printf("Ex-3 U=%4.2f\% (C1=1, C2=2, C3=3; T1=3, T2=5, T3=15; T=D): ",
           ((1.0 / 3.0) * 100.0 + (2.0 / 5.0) * 100.0 + (3.0 / 15.0) * 100.0));
    numServices = 3;
    if (scheduling_point_feasibility(numServices, ex3_period, ex3_wcet, ex3_period) == TRUE)
        printf("FEASIBLE\n");
    else
        printf("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");
    printf("\n");

    printf("Ex-4 U=%4.2f\% (C1=1, C2=1, C3=4; T1=2, T2=4, T3=16; T=D): ",
```

```
            ((1.0 / 2.0) * 100.0 + (1.0 / 4.0) * 100.0 + (4.0 / 16.0) * 100.0));
    numServices = 3;
    if (scheduling_point_feasibility(numServices, ex4_period, ex4_wcet, ex4_period) == TRUE)
        printf("FEASIBLE\n");
    else
        printf("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");
    printf("\n");

  printf("******** Scheduling Point Feasibility Example\n");
    printf("Ex-5 U=%4.2f%%  (C1=1, C2=2, C3=1; T1=2, T2=5, T3=10; T=D): ",
           ((1.0 / 2.0) * 100.0 + (2.0 / 5.0) * 100.0 + (1.0 / 10.0) * 100.0));
    numServices = 3;
    if (scheduling_point_feasibility(numServices, ex5_period, ex5_wcet, ex5_period) == TRUE)
        printf("FEASIBLE\n");
    else
        printf("INFEASIBLE\n");

    if (rate_monotonic_least_upper_bound(numServices, ex5_period, ex5_wcet, ex5_period) == TRUE)
        printf("RM LUB FEASIBLE\n");
    else
        printf("RM LUB INFEASIBLE\n");
    printf("\n");

 printf("******** Scheduling Point Feasibility Example\n");
    printf("Ex-6 U=%4.2f%%  (C1=1, C2=1, C3=1; C4=2 T1=2, T2=5, T3=7,T4=13; T=D): ",
           ((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 (scheduling_point_feasibility(numServices, ex6_period, ex6_wcet, ex6_period) == TRUE)
        printf("FEASIBLE\n");
    else
        printf("INFEASIBLE\n");

    if (rate_monotonic_least_upper_bound(numServices, ex6_period, ex6_wcet, ex6_period) == TRUE)
        printf("RM LUB FEASIBLE\n");
    else
        printf("RM LUB INFEASIBLE\n");
    printf("\n");

printf("******** Scheduling Point Feasibility Example\n");
    printf("Ex-7 U=%4.2f%%  (C1=1, C2=2, C3=4; T1=3, T2=5, T3=15; T=D): ",
           ((1.0 / 3.0) * 100.0 + (2.0 / 5.0) * 100.0 + (4.0 / 15.0) * 100.0));
    numServices = 3;
    if (scheduling_point_feasibility(numServices, ex7_period, ex7_wcet, ex7_period) == TRUE)
        printf("FEASIBLE\n");
    else
        printf("INFEASIBLE\n");

    if (rate_monotonic_least_upper_bound(numServices, ex7_period, ex7_wcet, ex7_period) == TRUE)
        printf("RM LUB FEASIBLE\n");
    else
        printf("RM LUB INFEASIBLE\n");
    printf("\n");
*/
printf("Ex-8 U=%4.2f%%  (C1=1, C2=1, C3=1; C4=2 T1=2, T2=5, T3=7,T4=13; T=D): ",
       ((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 (scheduling_point_feasibility(numServices, ex8_period, ex8_wcet, ex8_period) == TRUE)
        printf("FEASIBLE\n");
    else
        printf("INFEASIBLE\n");

    if (rate_monotonic_least_upper_bound(numServices, ex8_period, ex8_wcet, ex8_period) == TRUE)
        printf("RM LUB FEASIBLE\n");
    else
        printf("RM LUB INFEASIBLE\n");
    printf("\n");

        printf("Ex-9 U=%4.2f%%  (C1=1, C2=2, C3=4; C4=6 T1=6, T2=8, T3=12,T4=24; T=D): ",
               ((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 (scheduling_point_feasibility(numServices, ex9_period, ex9_wcet, ex9_period) == TRUE)
            printf("FEASIBLE\n");
        else
            printf("INFEASIBLE\n");

        if (rate_monotonic_least_upper_bound(numServices, ex9_period, ex9_wcet, ex9_period) == TRUE)
            printf("RM LUB FEASIBLE\n");
        else
            printf("RM LUB INFEASIBLE\n");
        printf("\n");

    printf("\n\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("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 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;
}
```

File- llf_feasibility.c

```
/*
 * File: llf_feasibility.c
 * Description: Tests llf feasibility of examples 5-9 using llf feasibility check
 * Author: [Nalin Saxena]
 * Date: [2025-02-26]
 */
#include <math.h>
#include <stdio.h>
#include <limits.h>

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

// example 5
U32_T ex5_period[] = {2, 5, 10};
U32_T ex5_wcet[] = {1, 2, 1};

// example 6 ommit example 6 as per assignment instrucitons
/*
U32_T ex6_period[] = {2,5,7,13};
U32_T ex6_wcet[] = {1,1,1,2};
*/
// example 7
U32_T ex7_period[] = {3, 5, 15};
U32_T ex7_wcet[] = {1, 2, 4};

// example 8
U32_T ex8_period[] = {2, 5, 7, 13};
U32_T ex8_wcet[] = {1, 1, 1, 2};

// example 9
U32_T ex9_period[] = {6, 8, 12, 24};
U32_T ex9_wcet[] = {1, 2, 4, 6};
```

```
int edf_feasibility_test(U32_T numServices, U32_T period[], U32_T wcet[], U32_T deadline[]);

int main()
{
    int i;
    U32_T numServices;
    printf("******** LLF feasibility test running********\n");

    printf("Ex-5 U=%4.2f%%  (C1=1, C2=2, C3=1; T1=2, T2=5, T3=10; T=D): ",
           ((1.0 / 2.0) * 100.0 + (2.0 / 5.0) * 100.0 + (1.0 / 10.0) * 100.0));
    numServices = 3;
    if (edf_feasibility_test(numServices, ex5_period, ex5_wcet, ex5_period) == TRUE)
        printf("LLF Feasibility test passed ! \n");
    else
        printf("LLF Feasibility test failed ! \n");

    printf("\n \n");

    printf("Ex-7 U=%4.2f%%  (C1=1, C2=2, C3=4; T1=3, T2=5, T3=15; T=D): ",
           ((1.0 / 3.0) * 100.0 + (2.0 / 5.0) * 100.0 + (4.0 / 15.0) * 100.0));
    numServices = 3;
    if (edf_feasibility_test(numServices, ex7_period, ex7_wcet, ex7_period) == TRUE)
        printf("LLF Feasibility test passed ! \n");
    else
        printf("LLF Feasibility test failed ! \n");

    printf("\n \n");

    printf("Ex-8 U=%4.2f%%  (C1=1, C2=1, C3=1; C4=2 T1=2, T2=5, T3=7,T4=13; T=D): ",
           ((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 (edf_feasibility_test(numServices, ex8_period, ex8_wcet, ex8_period) == TRUE)
        printf("LLF Feasibility test passed ! \n");
    else
        printf("LLF Feasibility test failed ! \n");

    printf("\n \n");

    printf("Ex-9 U=%4.2f%%  (C1=1, C2=2, C3=4; C4=6 T1=6, T2=8, T3=12,T4=24; T=D): ",
           ((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 (edf_feasibility_test(numServices, ex9_period, ex9_wcet, ex9_period) == TRUE)
        printf("LLF Feasibility test passed ! \n");
    else
        printf("LLF Feasibility test failed ! \n");
}


//compute utilization
float utilization_calc(U32_T numServices, U32_T period[], U32_T wcet[])
{
    float utilization = 0.0;
    for (int i = 0; i < numServices; i++)
    {
        utilization += (float)wcet[i] / period[i];
    }
    return utilization;
}

int edf_feasibility_test(U32_T numServices, U32_T period[], U32_T wcet[], U32_T deadline[])
```

```
{
    float utilization = utilization_calc(numServices, period, wcet);
    if (utilization > 1.0)
    {
        printf("schedule is not feasible \n");
        return FALSE;
    }
    return TRUE;
}
```

File- edf_feasibility

```
/*
 * File: edf_feasibility.c
 * Description: Tests edf feasibility of examples 5-9 using edf feasibility check
 * There are two test which are used one is utilization based and second is scheduling based test
 * Author: [Nalin Saxena]
 * Date: [2025-02-26]
 */
#include <math.h>
#include <stdio.h>
#include <limits.h>

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

// example 5
U32_T ex5_period[] = {2, 5, 10};
U32_T ex5_wcet[] = {1, 2, 1};

// example 6 ommit example 6 as per assignment instrucitons
/*
U32_T ex6_period[] = {2,5,7,13};
U32_T ex6_wcet[] = {1,1,1,2};
*/
// example 7
U32_T ex7_period[] = {3, 5, 15};
U32_T ex7_wcet[] = {1, 2, 4};

// example 8
U32_T ex8_period[] = {2, 5, 7, 13};
U32_T ex8_wcet[] = {1, 1, 1, 2};

// example 9
U32_T ex9_period[] = {6, 8, 12, 24};
U32_T ex9_wcet[] = {1, 2, 4, 6};

int edf_feasibility_test(U32_T numServices, U32_T period[], U32_T wcet[], U32_T deadline[]);

int main()
{
    int i;
    U32_T numServices;
    printf("******** EDF feasibility test running********\n");

    printf("Ex-5 U=%4.2f%%  (C1=1, C2=2, C3=1; T1=2, T2=5, T3=10; T=D): ",
           ((1.0 / 2.0) * 100.0 + (2.0 / 5.0) * 100.0 + (1.0 / 10.0) * 100.0));
    numServices = 3;
    if (edf_feasibility_test(numServices, ex5_period, ex5_wcet, ex5_period) == TRUE)
```

```
        printf("EDF Feasibility test passed ! \n");
    else
        printf("EDF Feasibility test failed ! \n");

    printf("\n \n");

    printf("Ex-7 U=%4.2f%%  (C1=1, C2=2, C3=4; T1=3, T2=5, T3=15; T=D): ",
           ((1.0 / 3.0) * 100.0 + (2.0 / 5.0) * 100.0 + (4.0 / 15.0) * 100.0));
    numServices = 3;
    if (edf_feasibility_test(numServices, ex7_period, ex7_wcet, ex7_period) == TRUE)
        printf("EDF Feasibility test passed ! \n");
    else
        printf("EDF Feasibility test failed ! \n");

    printf("\n \n");

    printf("Ex-8 U=%4.2f%%  (C1=1, C2=1, C3=1; C4=2 T1=2, T2=5, T3=7,T4=13; T=D): ",
           ((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 (edf_feasibility_test(numServices, ex8_period, ex8_wcet, ex8_period) == TRUE)
        printf("EDF Feasibility test passed ! \n");
    else
        printf("EDF Feasibility test failed ! \n");

    printf("\n \n");

    printf("Ex-9 U=%4.2f%%  (C1=1, C2=2, C3=4; C4=6 T1=6, T2=8, T3=12,T4=24; T=D): ",
           ((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 (edf_feasibility_test(numServices, ex9_period, ex9_wcet, ex9_period) == TRUE)
        printf("EDF Feasibility test passed ! \n");
    else
        printf("EDF Feasibility test failed ! \n");
}
// calculate gcd
int gcd(int a, int b)
{
    while (b != 0)
    {
        int temp = b;
        b = a % b;
        a = temp;
    }
    return a;
}

int lcm(int a, int b)
{
    return (a * b) / gcd(a, b);
    // common trick to calcualte lcm  https://www.geeksforgeeks.org/program-to-find-lcm-of-two-numbers/
}

//itertaively calculate lcm of all time periods
int calc_sim_period(U32_T numServices, U32_T period[])
{
    int final_period = period[0];
    for (int i = 1; i < numServices; i++)
    {
        final_period = lcm(period[i], final_period);
    }
```

```
    return final_period;
}

//compute utilization
float utilization_calc(U32_T numServices, U32_T period[], U32_T wcet[])
{
    float utilization = 0.0;
    for (int i = 0; i < numServices; i++)
    {
        utilization += (float)wcet[i] / period[i];
    }
    return utilization;
}

int edf_feasibility_test(U32_T numServices, U32_T period[], U32_T wcet[], U32_T deadline[])
{
    float utilization = utilization_calc(numServices, period, wcet);
    if (utilization > 1.0)
    {
        printf("schedule is not feasible \n");
        return FALSE;
    }
    int time = 0;
    int rem_wcets[numServices];
    int rem_deadline_times[numServices];
    int lcm_period = calc_sim_period(numServices, period);
    printf("\nsimulation period is %d \n", lcm_period);

    // initalize remaining time and capacity
//below code is not neccessary since utilization_calc test is sufficient to check edf feasibility
    for (int i = 0; i < numServices; i++)
    {
        rem_wcets[i] = wcet[i];
        rem_deadline_times[i] = period[i];
    }
    while (time < lcm_period)
    {
        int current_task_index = -1;
        int curr_earliest_deadline = INT_MAX;

        // find task with earliest deadline
        for (int i = 0; i < numServices; i++)
        {
            if (curr_earliest_deadline > rem_deadline_times[i] && rem_wcets[i] > 0)
            {
                curr_earliest_deadline = rem_deadline_times[i];
                current_task_index = i;
            }
        }
        if (current_task_index == -1)
        {
            time++;
            continue;
        }

        if (rem_wcets[current_task_index] > 0 && time > rem_deadline_times[current_task_index])
        {
            printf("Deadline missed! Task %d missed its deadline at time %d. Deadline was %d.\n",
                   current_task_index, time, rem_deadline_times[current_task_index]);
            return FALSE;
```

```
            // infeasible
        }
        //simulate task execution
        rem_wcets[current_task_index] -= 1;
        time++;

        if (rem_wcets[current_task_index] == 0)
        {
            // restore period and wcet
            rem_wcets[current_task_index] = wcet[current_task_index];
            rem_deadline_times[current_task_index] += period[current_task_index];
        }
    }
    return TRUE; //all done no deadlines missed return true
}
```

File - Deadline monotonic test

```
/*
 * File: DM_feasibility.c
 * Description: Tests DM feasibility for example 6 ( as per section 3.6 of text book its a simple utilization
 * utilization would be evaulated as U=∑Ci/Di<=1 (where C is compuation and Di is deadline should be less than 1)
 * The book also suggest to perform scheduling point tests and completition test
 */
#include <math.h>
#include <stdio.h>
#include <limits.h>

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

U32_T ex6_period[] = {2, 5, 7, 13};
U32_T ex6_wcet[] = {1, 1, 1, 2};
U32_T ex6_deadline[] = {2, 3, 7, 15};

int DM_feasibility_test(U32_T numServices, U32_T period[], U32_T wcet[], U32_T deadline[]);
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 main()
{
    int i;
    U32_T numServices;
    printf("******** DM feasibility test running********\n");

    printf("Ex-6 Scenario (C1=1, C2=1, C3=1 ,C4=2; D1=2, D2=3, D3=7 , D4=15; )");
    numServices = 4;
    if (DM_feasibility_test(numServices, ex6_period, ex6_wcet, ex6_deadline) == TRUE)
        printf("DM Feasibility test passed ! \n");
    else
        printf("DM Feasibility test failed ! \n");

    printf("********Completition  test running********\n");
    if (completion_time_feasibility(numServices, ex6_deadline, ex6_wcet, ex6_deadline) == TRUE)
        printf("CT test FEASIBLE\n");
    else
        printf("CT test INFEASIBLE\n");
    printf("********Scheduling point test running********\n");
```

```
    if (scheduling_point_feasibility(numServices, ex6_deadline, ex6_wcet, ex6_deadline) == TRUE)
        printf("FEASIBLE\n");
    else
        printf("INFEASIBLE\n");
}

float compute_interference(int i, U32_T period[], U32_T wcet[], U32_T deadline[])
{
    U32_T interference=0;
    for (U32_T j = 0; j < i; j++) //j<<i
    {
        interference += (U32_T)(ceil((float)deadline[i] / period[j])) * wcet[j];
    }
    return interference;
}

int DM_feasibility_test(U32_T numServices, U32_T period[], U32_T wcet[], U32_T deadline[])
{
    float utilization;
    for (U32_T i = 0; i < numServices; i++)
    {
        U32_T interference = compute_interference(i, period, wcet, deadline);
        utilization = (float)wcet[i] / deadline[i] + (float)interference / deadline[i];
    }
    if (utilization > 1.0)
    {
        printf("test failed task not schedulable \n");
        return FALSE;
    }
    return TRUE;
}

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;
}
```