ECEN 5623 – Real Time Embedded Systems

Summer 2017

EXERCISE 2

SERVICE SCHEDULING FEASIBILITY

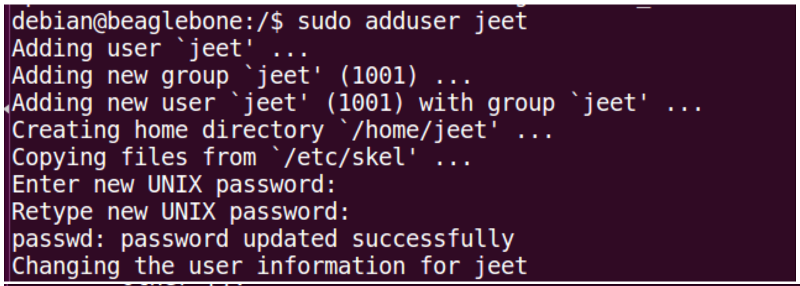
Jeet Baru

Email: jeet.baru@colorado.edu

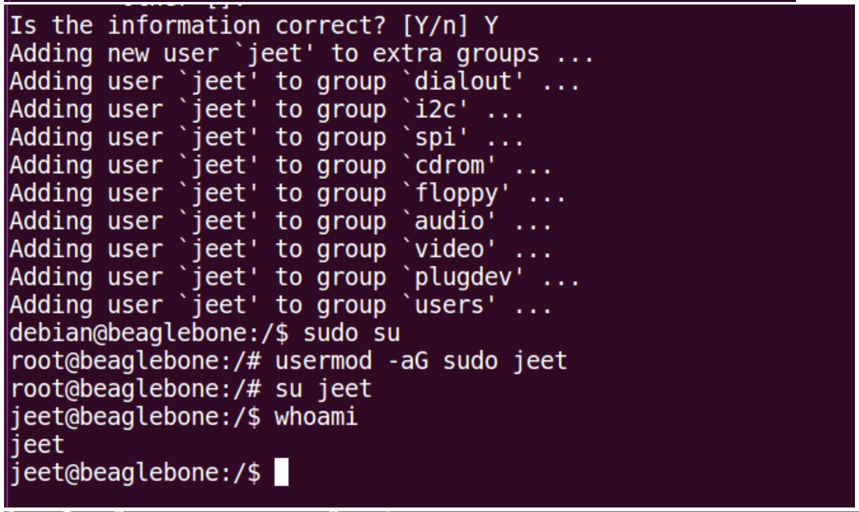
ID Number: 107189037

Question 1

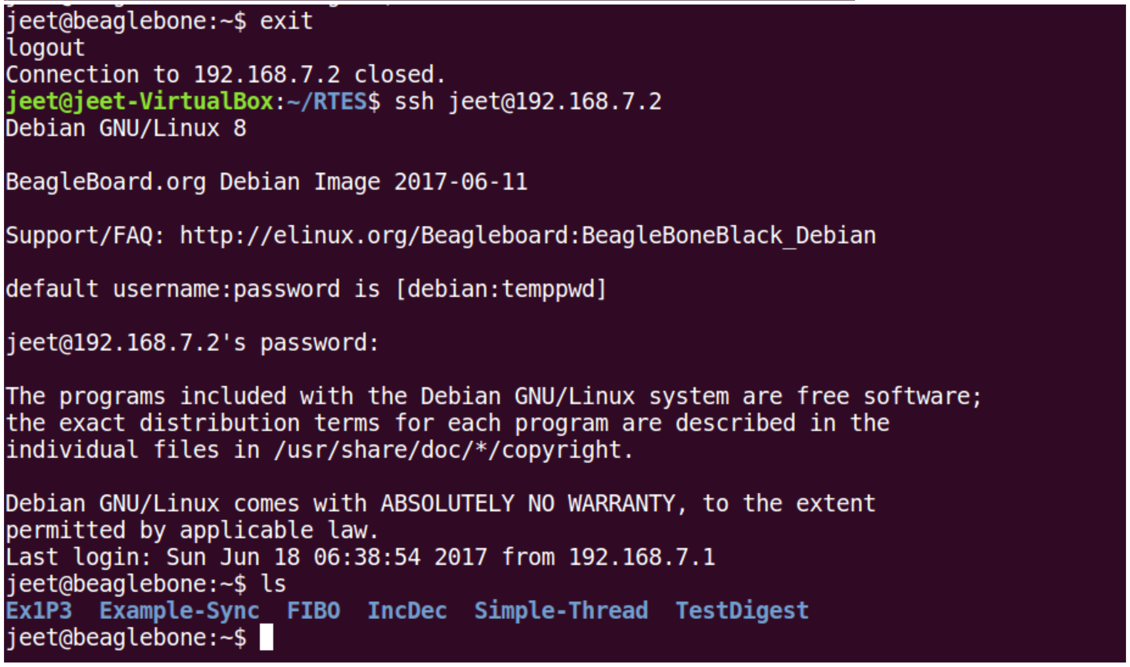
In this question we are required to create a new user in the embedded environment that we are using, in my case Ubuntu on BeagleBone Black. After adding a new user, we give it super user priveleges by adding it to 'sudo' group using flag -aG user usermod command. This can also be done by editing the sudoer file using command visudo. We then log out and log back in and test success using the whoami command.



```
debian@beaglebone:/$ sudo adduser jeet
Adding user `jeet' ...
Adding new group `jeet' (1001) ...
Adding new user `jeet' (1001) with group `jeet' ...
Creating home directory `/home/jeet' ...
Copying files from `/etc/skel' ...
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Changing the user information for jeet
```



```
Is the information correct? [Y/n] Y
Adding new user `jeet' to extra groups ...
Adding user `jeet' to group `dialout' ...
Adding user `jeet' to group `i2c' ...
Adding user `jeet' to group `spi' ...
Adding user `jeet' to group `cdrom' ...
Adding user `jeet' to group `floppy' ...
Adding user `jeet' to group `audio' ...
Adding user `jeet' to group `video' ...
Adding user `jeet' to group `plugdev' ...
Adding user `jeet' to group `users' ...
debian@beaglebone:/$ sudo su
root@beaglebone:/# usermod -aG sudo jeet
root@beaglebone:/# su jeet
jeet@beaglebone:/$ whoami
jeet
jeet@beaglebone:/$
```



```
jeet@beaglebone:~$ exit
logout
Connection to 192.168.7.2 closed.
jeet@jeet-VirtualBox:~/RTES$ ssh jeet@192.168.7.2
Debian GNU/Linux 8

BeagleBoard.org Debian Image 2017-06-11

Support/FAQ: http://elinux.org/Beagleboard:BeagleBoneBlack_Debian

default username:password is [debian:temppwd]

jeet@192.168.7.2's password:

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

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sun Jun 18 06:38:54 2017 from 192.168.7.1
jeet@beaglebone:~$ ls
Ex1P3  Example-Sync  FIBO  IncDec  Simple-Thread  TestDigest
jeet@beaglebone:~$
```

Question 2

In this question we are required to read a paper on "Architecture of the Space Shuttle Primary Avionics Software System", by Gene Carlow and provide an explanation and critique of the frequency executive architecture.

In this paper the author describes the architecture of the flight control software system to be used in NASA's space shuttle. The software system had various layers of development that gave in for the final product. These layers grew from the base of the operating system up till the layer of user interface providing abstraction a practice that is followed even today. Because it was a mission critical system many redundancies were built into the system and the paper provides for reasons for various design decisions made during the software systems design. Due to the constraint on main memory the whole code could not be loaded onto it. Hence the software system had been segmented into 8 different parts called OPS based on its function and requirement in time. Each OPS would then be loaded onto the main memory when required. OPS code is otherwise stored in memory outside of main memory.

There are three different modes in the system software's operational sequence structure to communicate with the computers to monitor the avionic system. These include a major mode which divides the primary processing within OPS into major steps, a specialist function (SPEC) which is initiated upon keyboard entry and executes concurrently with other processing to perform background functions, and a display function to monitor processing results of SPEC or major mode functions. Flight software systems are usually designed in a synchronous manner wherein each application process is dispatched at the correct instant at which it was meant to be executed. For this purpose, an executive is sufficient rather than using a full-blown Real-time operating system, which gives one the benefit of repeatability. However, this comes at a cost of limited flexibility for changes and since shuttle computers have to perform many different applications and also allow for changes in the requirements, PASS adopted a non-synchronous approach along with the decision of isolating application processes from external I/O.

There is also a user interface provided for communications between systems and applications software. The Flight Control OS primarily performs process management to control allocation of internal computer resources and uses a multitasking priority queue to schedule and allocate CPU resources. This queue structure ensures that the highest priority process with work to perform is given CPU resources to accomplish the task. Apart from this, FCOS also performs input/output management to control allocation of input/output processor resources. The on-board software has been developed to perform three important functions of GN&C, vehicle systems management and vehicle checkout. It is a cyclic closed loop application which performs various functions with tight timing and phasing. Different applications are performed at different frequencies depending on their priorities and need to be serviced. There is a cyclic process (executive) that controls the dispatch of the functions using a table-driven dispatching technique.

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

Overall, the paper does give us a sneak-peek into a complex software architecture. Getting the timing correct maintaining all functionality was important due to mission-criticalness and to achieve that a large number of good software constructs and practices including cyclic executives had to be implemented.

Advantages of frequency executives over threads/tasks for RTOS: -

- The frequency executive design is simpler compared to RTOS and all the scheduling algorithms that go with them. This is because it can be considered to be a form of cooperative multitasking that has only a single task and hence dispatching of the task is efficient and straightforward.
- There is little context switching involved in cyclic executives as it is a form of non-pre-emptive tasking. Context switching is a major contributor to latencies involved in the system that hamper the ability to service a request on time and cyclic executive are spared from that overhead.
- Cyclic executives are highly deterministic. The schedule only has to be stored in a table which can be read by the executive. Hence, you have a highly predictable system that you can validate and test.
- Frequency executive is repeatable and hence stable with less bugs and hence preventing malfunctioning.
- Having a table for frequencies of each task avoids the overhead that may be present in a dynamic priority scheduler which computes next task that must be executed.
- Lesser chance of corruption of data as a sequence of and time taken for execution of tasks is fixed

Disadvantages of frequency executives compared to threads/tasks for RTOS: -

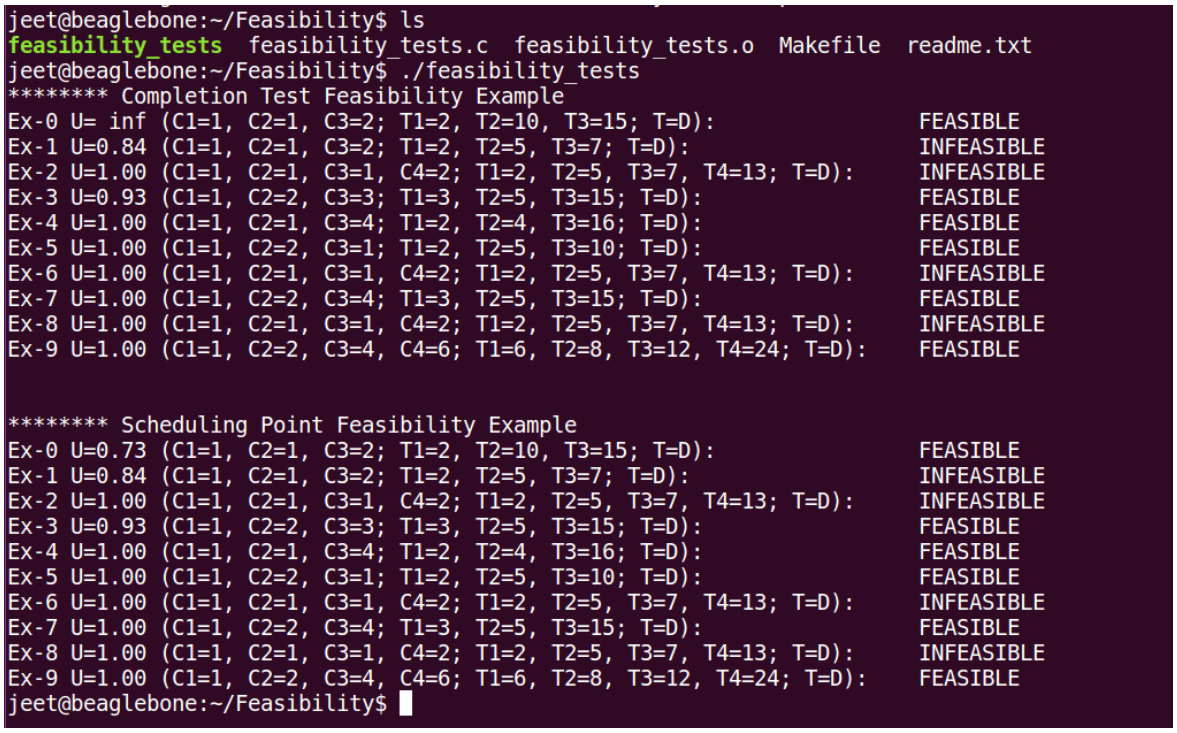
- One of the major disadvantages of cyclic schedules is the limited flexibility that you have for changes – cannot deal with run-time changes. Also any new service requires a new entry in the table to be computed that can be cumbersome and inefficient.
- Since it is a non-pre-emptive style of tasking, any long but low priority task may prevent a higher priority task from running for a long time – this is a dangerous situation for critical missions.
- Splitting big tasks into subtasks and determining the scheduling blocks for each tasks is a time-consuming process. It is also difficult and there is a possibility of introducing errors into the process. This is in contrast to most modern-day OS’s using pre-emptive scheduling and splitting a big problem into a number of threads or tasks.
- Unable to handle asynchronous requests might lead to delayed response to any interrupt, ultimately may or may not lead to missing deadlines in the worst case.

Question 3

In this question we are required to modify and run feasibility_test.c code provided by Professor Siewert. This code runs the completion time test and scheduling point test for Rate monotonic policy for given examples. The code has functions for completion time and scheduling point tests and arrays of WCETs (worst case execution times), Time periods and Deadlines of tasks is passed as parameters to return whether the given example is feasible or not.

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

We use the modified code to generate test results for all examples and obtain results as follows:



```
jeet@beaglebone:~/Feasibility$ ls
feasibility_tests  feasibility_tests.c  feasibility_tests.o  Makefile  readme.txt
jeet@beaglebone:~/Feasibility$ ./feasibility_tests
******** Completion Test Feasibility Example
Ex-0 U= inf (C1=1, C2=1, C3=2; T1=2, T2=10, T3=15; T=D):               FEASIBLE
Ex-1 U=0.84 (C1=1, C2=1, C3=2; T1=2, T2=5, T3=7; T=D):                 INFEASIBLE
Ex-2 U=1.00 (C1=1, C2=1, C3=1, C4=2; T1=2, T2=5, T3=7, T4=13; T=D):    INFEASIBLE
Ex-3 U=0.93 (C1=1, C2=2, C3=3; T1=3, T2=5, T3=15; T=D):                FEASIBLE
Ex-4 U=1.00 (C1=1, C2=1, C3=4; T1=2, T2=4, T3=16; T=D):                FEASIBLE
Ex-5 U=1.00 (C1=1, C2=2, C3=1; T1=2, T2=5, T3=10; T=D):                FEASIBLE
Ex-6 U=1.00 (C1=1, C2=1, C3=1, C4=2; T1=2, T2=5, T3=7, T4=13; T=D):    INFEASIBLE
Ex-7 U=1.00 (C1=1, C2=2, C3=4; T1=3, T2=5, T3=15; T=D):                FEASIBLE
Ex-8 U=1.00 (C1=1, C2=1, C3=1, C4=2; T1=2, T2=5, T3=7, T4=13; T=D):    INFEASIBLE
Ex-9 U=1.00 (C1=1, C2=2, C3=4, C4=6; T1=6, T2=8, T3=12, T4=24; T=D):   FEASIBLE


******** Scheduling Point Feasibility Example
Ex-0 U=0.73 (C1=1, C2=1, C3=2; T1=2, T2=10, T3=15; T=D):               FEASIBLE
Ex-1 U=0.84 (C1=1, C2=1, C3=2; T1=2, T2=5, T3=7; T=D):                 INFEASIBLE
Ex-2 U=1.00 (C1=1, C2=1, C3=1, C4=2; T1=2, T2=5, T3=7, T4=13; T=D):    INFEASIBLE
Ex-3 U=0.93 (C1=1, C2=2, C3=3; T1=3, T2=5, T3=15; T=D):                FEASIBLE
Ex-4 U=1.00 (C1=1, C2=1, C3=4; T1=2, T2=4, T3=16; T=D):                FEASIBLE
Ex-5 U=1.00 (C1=1, C2=2, C3=1; T1=2, T2=5, T3=10; T=D):                FEASIBLE
Ex-6 U=1.00 (C1=1, C2=1, C3=1, C4=2; T1=2, T2=5, T3=7, T4=13; T=D):    INFEASIBLE
Ex-7 U=1.00 (C1=1, C2=2, C3=4; T1=3, T2=5, T3=15; T=D):                FEASIBLE
Ex-8 U=1.00 (C1=1, C2=1, C3=1, C4=2; T1=2, T2=5, T3=7, T4=13; T=D):    INFEASIBLE
Ex-9 U=1.00 (C1=1, C2=2, C3=4, C4=6; T1=6, T2=8, T3=12, T4=24; T=D):   FEASIBLE
jeet@beaglebone:~/Feasibility$
```

Source code to perform these tests has been attached in the zip file.

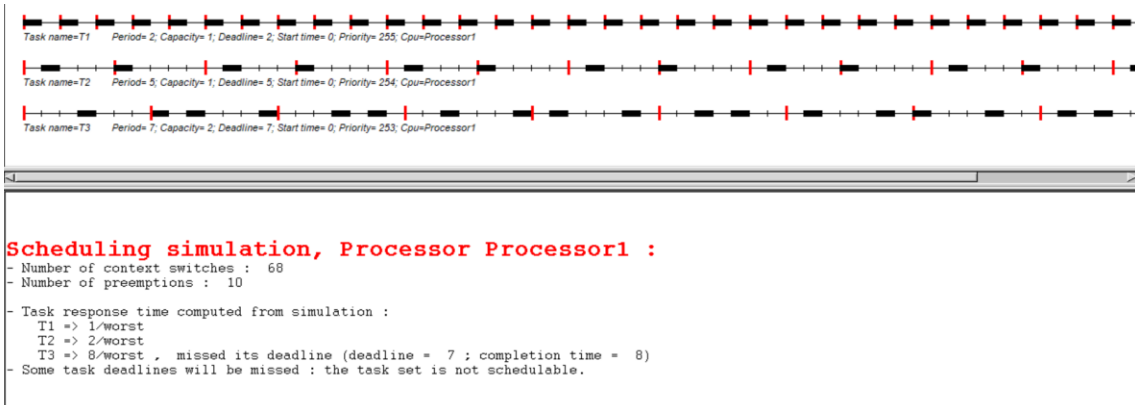
We also use Cheddar, a GPL real time scheduling tool to compare and verify our results from running the code. Cheddar allows us to model software architectures of real-time systems to check its schedulability or other performance criteria. It does this by scheduling simulations and/or feasibility test.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

Screenshots of results obtained from cheddar have been attached in the zip file.

A summary of all tests has been described in a tabular format as shown below:

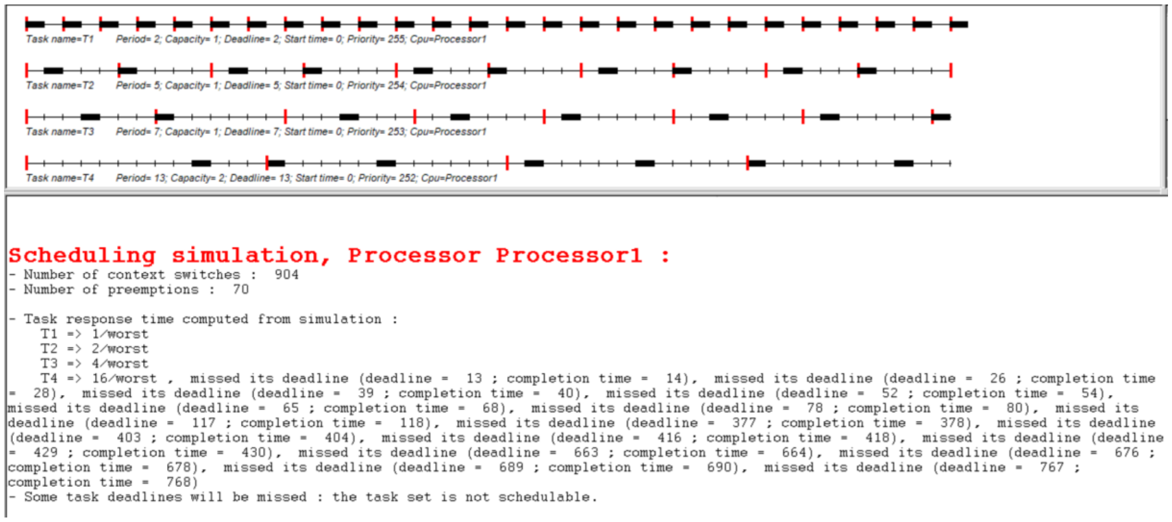
	Ex 0	Ex 1	Ex 2	Ex 3	Ex 4	Ex 5	Ex 6	Ex 7	Ex 8	Ex 9
RM Cheddar	Feasible	Not Feasible	Not Feasible	Feasible	Feasible	Feasible	Not Feasible	Feasible	Not Feasible	Feasible
EDF Cheddar	Feasible	Feasible	Feasible	Feasible	Feasible	Feasible	Feasible	Feasible	Feasible	Feasible
LLF Cheddar	Feasible	Feasible	Feasible	Feasible	Feasible	Feasible	Feasible	Feasible	Feasible	Feasible
DM Cheddar	X	X	X	X	X	X	Feasible	X	X	X
RM Completion Time Test	Feasible	Not Feasible	Not Feasible	Feasible	Feasible	Feasible	Not Feasible	Feasible	Not Feasible	Feasible
RM Scheduling Point Test	Feasible	Not Feasible	Not Feasible	Feasible	Feasible	Feasible	Not Feasible	Feasible	Not Feasible	Feasible

We observe that Examples 1, 2, 6 and 8 fail if scheduled using the RM policy while are feasible if dynamic policies like EDF and LLF are applied. Screenshot of the timing diagram for example 1 and 2 (same for examples 6 and 8) are as shown below:


Task name=T1 Period= 2; Capacity= 1; Deadline= 2; Start time= 0; Priority= 255; Cpu=Processor1
Task name=T2 Period= 5; Capacity= 1; Deadline= 5; Start time= 0; Priority= 254; Cpu=Processor1
Task name=T3 Period= 7; Capacity= 2; Deadline= 7; Start time= 0; Priority= 253; Cpu=Processor1

```
Scheduling simulation, Processor Processor1 :
- Number of context switches :  68
- Number of preemptions :  10

- Task response time computed from simulation :
    T1 => 1/worst
    T2 => 2/worst
    T3 => 8/worst ,  missed its deadline (deadline =  7 ; completion time =  8)
- Some task deadlines will be missed : the task set is not schedulable.
```


Task name=T1 Period= 2; Capacity= 1; Deadline= 2; Start time= 0; Priority= 255; Cpu=Processor1
Task name=T2 Period= 5; Capacity= 1; Deadline= 5; Start time= 0; Priority= 254; Cpu=Processor1
Task name=T3 Period= 7; Capacity= 1; Deadline= 7; Start time= 0; Priority= 253; Cpu=Processor1
Task name=T4 Period= 13; Capacity= 2; Deadline= 13; Start time= 0; Priority= 252; Cpu=Processor1

```
Scheduling simulation, Processor Processor1 :
- Number of context switches :  904
- Number of preemptions :  70

- Task response time computed from simulation :
    T1 => 1/worst
    T2 => 2/worst
    T3 => 4/worst
    T4 => 16/worst ,  missed its deadline (deadline =  13 ; completion time =  14),  missed its deadline (deadline =  26 ; completion time
=  28),  missed its deadline (deadline =  39 ; completion time =  40),  missed its deadline (deadline =  52 ; completion time =  54),
missed its deadline (deadline =  65 ; completion time =  68),  missed its deadline (deadline =  78 ; completion time =  80),  missed its
deadline (deadline =  117 ; completion time =  118),  missed its deadline (deadline =  377 ; completion time =  378),  missed its deadline
(deadline =  403 ; completion time =  404),  missed its deadline (deadline =  416 ; completion time =  418),  missed its deadline (deadline
=  429 ; completion time =  430),  missed its deadline (deadline =  663 ; completion time =  664),  missed its deadline (deadline =  676 ;
completion time =  678),  missed its deadline (deadline =  689 ; completion time =  690),  missed its deadline (deadline =  767 ;
completion time =  768)
- Some task deadlines will be missed : the task set is not schedulable.
```

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

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

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

Question 4

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

1. The deadline for a given task is equal to its period, which might not be the case always.
2. The threads have fixed priority based on the time period
3. The tasks are preemptive and run till completion

Assumptions made in the Liu and Laylands paper on Rate Monotonic policy about hard real time environment that are critical for the analysis are:

1. Tasks having hard real time deadlines have periodic requests.
2. Run time for each task is constant and does not vary with time.
3. The tasks are independent and the requests for a certain task do not depend on the initiation or the completion of requests for other tasks
4. Each task must be completed before the next request for it occurs.

Three key and tricky steps in the derivation for Least upper bound are:

1. Case 1:

   In this part the authors of the paper ignore intermediate steps and directly arrive at the equation for utilization factor from the equation for largest possible completion time for task 2. This step I felt was tricky.

   On reading the book however the idea became clearer as intermediate steps were described and explained. We understand why the floor(T_2/T_1) expression is used. Because in the example, the third occurrence of T_1 overlaps with T_2 and thus it cannot be considered in the equation as it does not occur fully. The floor expression generalizes this overlap. Further steps simplify the equation and break it down to separate terms whose effect is analyzed. We hence finally achieve the plot depicting how Utilization factor decreases monotonically as C_1 increases.

2. Case 2:

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

   Chapter 3 explains this very well and effectively explains how the utilization is derived and why the floor(T_2/T_1) expression is used. The text further goes on to analyze the terms of the utilization equation using values for T_2 and T_1. The final plot shows how the utilization factor increases as C_1 increases.

   Finally, difference between utilization factor functions for case 1 and case 2 is evaluated to obtain a general expression for

3. Final part I did not understand was the derivation in theorem 4. In this part the authors use previous results to derive an expression for utilization factor for m tasks. I did not understand how $C_1 = T_2 - T_1$ is same as C_1' and how $C_2' = C_2 + \Delta$ but others don't follow this.

   Chapter 3 of the textbook however doesn't seem to have taken this approach to derive an expression for m tasks. A simple comparison of expression for two tasks and showing that m=2 and hence coming up with generic expression although is lucid and easy to understand but not very convincing.