University of Colorado
Boulder

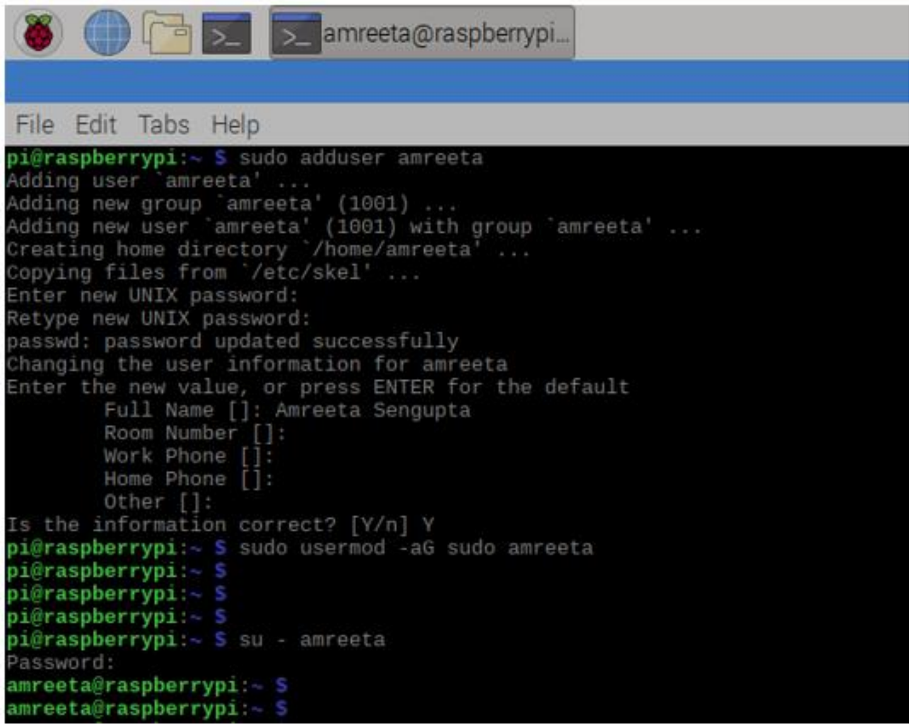
EXERCISE 2

BY
Amreeta Sengupta and Mohit Rane

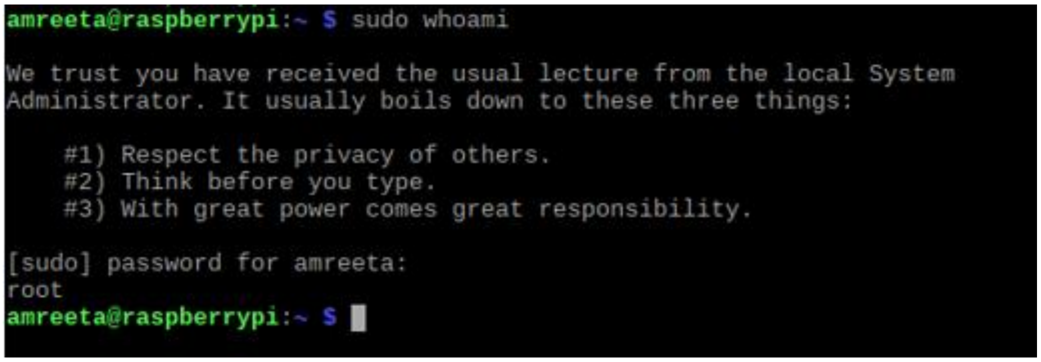
02/21/2019

Question 1 performed on Raspberry Pi and Question 4 performed on Virtual Box and Scheduling Analysis performed on Cheddar 3.1

QUESTION 1



```
pi@raspberrypi:~ $ sudo adduser amreeta
Adding user `amreeta' ...
Adding new group `amreeta' (1001) ...
Adding new user `amreeta' (1001) with group `amreeta' ...
Creating home directory `/home/amreeta' ...
Copying files from `/etc/skel' ...
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Changing the user information for amreeta
Enter the new value, or press ENTER for the default
        Full Name []: Amreeta Sengupta
        Room Number []:
        Work Phone []:
        Home Phone []:
        Other []:
Is the information correct? [Y/n] Y
pi@raspberrypi:~ $ sudo usermod -aG sudo amreeta
pi@raspberrypi:~ $
pi@raspberrypi:~ $
pi@raspberrypi:~ $
pi@raspberrypi:~ $ su - amreeta
Password:
amreeta@raspberrypi:~ $
amreeta@raspberrypi:~ $
```



```
amreeta@raspberrypi:~ $ sudo whoami

We trust you have received the usual lecture from the local System
Administrator. It usually boils down to these three things:

    #1) Respect the privacy of others.
    #2) Think before you type.
    #3) With great power comes great responsibility.

[sudo] password for amreeta:
root
amreeta@raspberrypi:~ $
```

ADDING ROOT USER WITH THE NAME "amreeta"

QUESTION 2

Overall understanding of paper and key point articulation

This paper tells us about the evolution, development and structure used in the architecture of Space Shuttle Primary Avionics Software System (PASS), which is one of most intricate flight computer programs ever developed. PASS has become essential in majority of the functions involved in the Space Shuttle Orbiter and it employs a multitude of computer redundancy management and timing critical concepts which include the Data Processing System (DPS)/IBM AP-101 General Purpose Computer (GPC) design and memory/CPU constraints; the multicomputer redundancy management and synchronization; the operational sequencing/mode control and man/machine interface requirements; the applications functional and performance requirements; and the requirement for design modularity and modification flexibility necessitated by the R&D nature of the Shuttle Program.

Along with the main memory, which has a capacity if AP-101 (106K 32-bit words), mass memory is used to accommodate the software requirements. To aid the reliability considerations that demand that the software needed to support a single critical mission phase such as ascent must be present in each redundant GPC main memory throughout that phase, PASS has been segmented into eight different phase/function combinations, each of which is discerned by a unique operational sequence (OPS). In most cases, the period in which the OPS is executed does not include any transfer between the mass memory and the GPC main memory. The main memory is loaded with the software that is required for the OPS at its initialization, from the mass memory and this load has three parts:

- The resident or systems software, which contains code and data common to all OPS loads.
- The major function base, which contains code and data common to a major applications function used in more than one OPS load.
- The OPS overlay, which contains the applications code and data unique to an OPS load.

Loading of these parts of the main memory for the new OPS is decided by the contents of the current OPS in progress. At best, the loading occurs for the OPS overlay.

The PASS man/machine interface is designed to accommodate a knowledgeable user, with minimum time and effort required to communicate with the computers to monitor and control a very complex Orbiter avionic system. This is implemented in the form of an OPS substructure and it includes:

- Major modes for segmentation of primary processing within the OPS into major steps or sequences and can be further substructured into blocks that are linked to CRT display pages in order to establish an orderly sequence for the crew to communicate and maintain control of the software. Each OPS has one or more modes and sequencing from one mode/block to another and the processing performed within are initiated either by keyboard entry from the crew or, sometimes, automatically as the result of a specific event or condition that is detected by the software. At least one major mode will be entered any time an OPS is initiated.
- Specialist function (SPEC) which is initiated within an OPS by keyboard entry from the crew and is used for secondary or background functions. It also has a substructure of blocks, which are linked to CRT displays and establish the valid keyboard entry options
- Display function (DISP) which is used for monitoring the processing results of a SPEC or major mode function. It does not initialize any processing function.

Control segments consisting of standardized logic blocks are used to basically establish the structure of an OPS, major modes, SPEC or DISP. Control segment grammar, which is basically a library of macros, has been developed to standardize and modularize the design and implementation of control segments.

The flight software system (which have a small size, i.e. 15K to 30K) employ a synchronous design approach with dispatch of each application process at a specific point, relative to the start of an overall system cycle, which gives repeatability but limited flexibility to accommodate change. The evolution of the Shuttle program lead to adoption of a nonsynchronous approach with the ever-changing requirements. It also included isolation of the application processes from the external I/O and computer redundancy management. Sequencing and control of applications were achieved through interfaces with major elements of the system software.

The Flight Computer Operating System has the following functions:

- Process management: Controls allocation of internal CPU resources in response to requests from other systems and/or application processes through a standard set of service interfaces (SVCs).
- I/O Management: Controls the allocation of the input/output processor (IOP) resources. The IOP of each GPC contains a master sequence controller (MSC) and 24 bus control elements (BCEs). Each GPC component (CPU, MSC, and 24 BCEs) is an asynchronous processor communicating with the next higher and/or lower processor during I/O operations.
- Data Processing System (DPS) - Loading of the GPC memories and sequencing and control of the GPC and IOP operating states, including hardware initialization and status checks is done by the DPS. Performs synchronization of multiple computers in a redundant configuration at the rate of 350 times per second.

System Control is used for the initialization and configuration control of the DPS and associated avionic data network. Its functions include establishment of an interface with other GPCs. The GPC, in response to crew commands, initiate SPECs which can be used for resetting the MTU, initiating a memory dump, examining or changing a specific core location in memory etc.

The user interface is used for external control of systems or applications processing defined in the control segment and its functions include:

- Command input processing.
- Operations control.
- Output message processing.

Support of the crew interface, keyboard entry, and display generation is accomplished with both on-board software and off-line processors.

The on-board software has been developed for three major applications which are:

- Guidance, navigation and control (GN&C): It is used to determine the vehicle position, velocity and attitude, performs sensor redundancy management; provides the crew with displays and data entry capabilities to monitor and control the avionics subsystems; and issues the engine and/or affector commands for a mission from lift-off through touchdown and rollout. 200 of the GN&C functions called principal functions, are included in six different OPSs, three of which must execute in a redundant computer configuration. There are from one to ten major modes included in the six OPSs, and ten SPECs, each of which is available in one or more of the OPSs.
- Vehicle systems management (SM): SM monitors the performance and configuration of Orbiter and payload subsystems to detect abnormal conditions and alert the flight and ground crews via CRT displays and/or lights and alarms so they can take corrective action. SM functions are included in a single OPS initiated during the orbit coast phase of a Shuttle mission.
- Vehicle checkout (VCO): The vehicle checkout (VCO) is designed to support avionics system initialization and checkout under control of ground and/or flight crews. Its primary functions and associated I/O interfaces are configured into three ground checkout OPSs and one in-flight checkout OPS. A unique feature

of the VCO design is the test control supervisor (TCS). Initiated through a SPEC, it provides to the user a variety of command/response processes, which make it possible to develop test sequences external to the software for execution in a vehicle checkout OPS.

The structural aspects of the software architecture and HAL/S language together with the standardization of interfaces through the FCOS SVCs and control segment grammar have been major factors in the successful implementation of PASS software. Use of comprehensive architectural design strategies and effective configuration controls makes it possible to more readily absorb and manage the instability of a changing environment. Thus, PASS architecture lays a strong foundation to support a dynamic environment with respect to an Orbiter avionic system development.

Advantages of frequency executive

- The absence of unexpected context switches results in reduced overhead as not even interrupts demand attention due to predetermined execution line.
- Jitter is reduced to a great extent due to predictable periods of execution.
- The predictable nature results in reduced chances of unfamiliar errors.
- It is favorable to be used in hard real time systems which perform deterministic functions.

Disadvantages of frequency executive

- Reduces the flexibility of the system and its use is mostly limited to deterministic systems.
- A mismatch in the frequency of the tasks can result in hindrance to the system performance as the scheduler will be required to derive a way to schedule the tasks such that the frequency requirements are met which will increase the CPU utilization.
- It is inconvenient to use in sporadic tasks and tasks with dependencies.
- It is unable to deal with runtime changes.

QUESTION 3

Overall understanding of paper and key point articulation

This paper tells us about the development of safety critical systems using synchronous architecture based on cyclic scheduling implemented in Ada 95 as the strict safety requirements limits the use of concurrent threads and dynamic data objects. This architecture uses object-oriented design and includes the use of New Ada characteristics such as exceptions and generics. It also describes the use of reusable components for development of real time systems.

Synchronous architectures represent predictable behavior due to absence of sudden interleaving of operations which makes it reliable and makes testing easy. Its disadvantage is the complexity of building cyclic structures and the low abstraction level of the resulting software which increases its difficulty and cost of maintenance and development. One way of increasing the system flexibility as well as suitability for all types of real time systems is the use of reusable software components.

The cyclic schedule consists of a full execution pattern for a major cycle which in turn consists of a fixed number of minor cycles executed periodically. The minor cycle consists of the frames of variable duration which basically executes a process or a subprocess. Portability is introduced with the use of real time clocks and timer software.

Fault tolerance is incorporated using the concept of recovery groups in which if one of the process in a set of related processes fails, then the entire group is aborted. Another failure, that needs to be taken care of, is deadline overrun.

The main component of synchronous architecture is the cyclic scheduler which is basically an iterative procedure that allows multiplexing of periodic process set in a processor. Basically, a major schedule is executed repeatedly during a major cycle until the change in the operating mode of the system. The major schedule is divided into one or more minor schedules and the minor cycles occur one after another and are synchronized by the ticks of the minor clock wherein each tick indicates the end of a minor cycle. If the process sequence corresponding to the minor cycle has not finished its execution, minor cycle overrun occurs. Each minor schedule is further divided into frames and if the execution of the frame is not completed, it results in frame overrun.

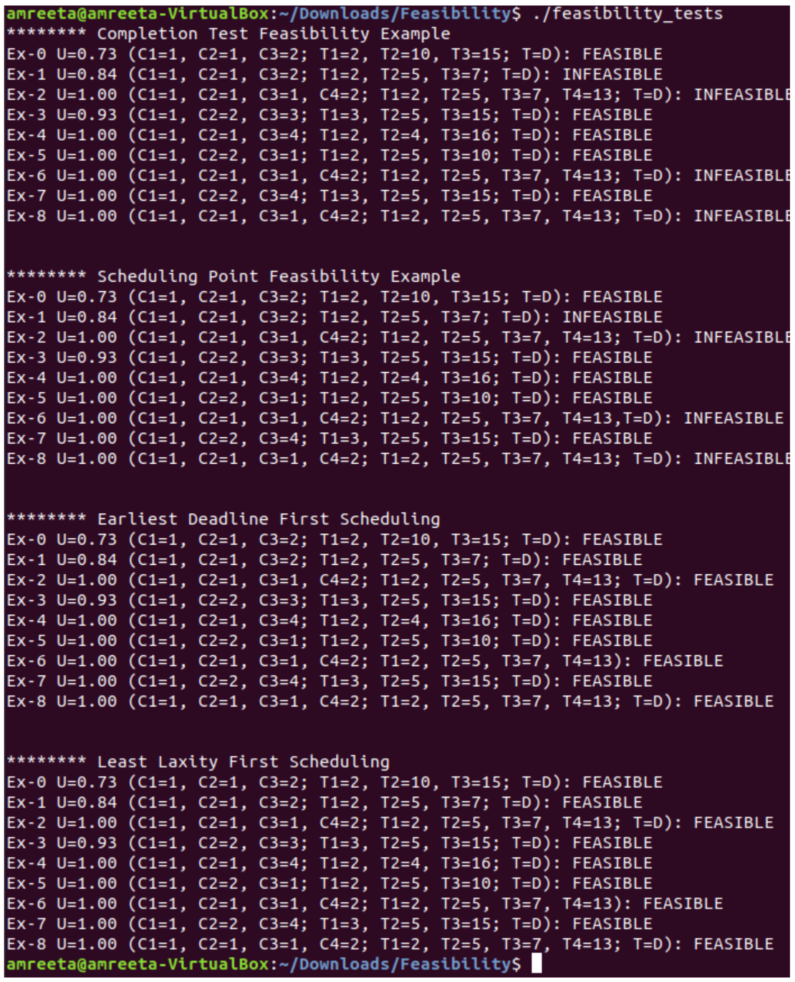
The error control that is implemented here is through the use of exceptions, which leads to temporary suspension of program and an exception handler takes control of the execution. There are several predefined exceptions or new exceptions can be explicitly activated through the statement raise. Exception mechanism can be used in the case of appearance of overruns which will lead to activation of the corresponding exception. Using this mechanism, the cyclic scheduler's main loop structure can be coded in a simple and clean manner.

A generic architecture is used for the design of the synchronous real time systems which include components like structures for the definition of the basic data structures in a portable manner, executives which is a subsystem consisting of components like compiler, scheduler, two interval timers etc., and shells which are templates for building the application processes. The entire system structure is represented as a global component which makes it easier to build. This architecture promotes reusability and easy adaptability. When implemented in Ada, it provides abstraction through the hierarchical public and private packages and also has standard interfaces which facilitates the concept of reusability. Overall, this architecture encourages a safe and simple development of real time systems.

Comparison of Cyclic executive and RTOS approaches

- The absence of unexpected context switches in cyclic executives results in reduced overhead as not even interrupts demand attention due to predetermined execution line while RTOS and Linux approaches have to deal with preemption of tasks and dynamic priority scheduling designs which lead to greater overheads being associated with them. This also results in lower jitter in case of cyclic executives.
- The cyclic executive has a lower abstraction layer than the RTOS or Linux systems which makes addition of new features and maintenance more difficult. A real time operating system on the other hand has greater flexibility and portability.
- The executable code can be kept small in cyclic executives due to the lack of requirement of operating system or run time support system.
- Cyclic executives can be used in safety critical applications due to its predictable nature while RTOS systems can be expected to have more arbitrary errors.

QUESTION 4



```
amreeta@amreeta-VirtualBox:~/Downloads/Feasibility$ ./feasibility_tests
******** Completion Test Feasibility Example
Ex-0 U=0.73 (C1=1, C2=1, C3=2; T1=2, T2=10, T3=15; T=D): FEASIBLE
Ex-1 U=0.84 (C1=1, C2=1, C3=2; T1=2, T2=5, T3=7; T=D): INFEASIBLE
Ex-2 U=1.00 (C1=1, C2=1, C3=1, C4=2; T1=2, T2=5, T3=7, T4=13; T=D): INFEASIBLE
Ex-3 U=0.93 (C1=1, C2=2, C3=3; T1=3, T2=5, T3=15; T=D): FEASIBLE
Ex-4 U=1.00 (C1=1, C2=1, C3=4; T1=2, T2=4, T3=16; T=D): FEASIBLE
Ex-5 U=1.00 (C1=1, C2=2, C3=1; T1=2, T2=5, T3=10; T=D): FEASIBLE
Ex-6 U=1.00 (C1=1, C2=1, C3=1, C4=2; T1=2, T2=5, T3=7, T4=13; T=D): INFEASIBLE
Ex-7 U=1.00 (C1=1, C2=2, C3=4; T1=3, T2=5, T3=15; T=D): FEASIBLE
Ex-8 U=1.00 (C1=1, C2=1, C3=1, C4=2; T1=2, T2=5, T3=7, T4=13; T=D): INFEASIBLE


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


******** Earliest Deadline First Scheduling
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): FEASIBLE
Ex-2 U=1.00 (C1=1, C2=1, C3=1, C4=2; T1=2, T2=5, T3=7, T4=13; T=D): FEASIBLE
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): FEASIBLE
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): FEASIBLE


******** Least Laxity First Scheduling
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): FEASIBLE
Ex-2 U=1.00 (C1=1, C2=1, C3=1, C4=2; T1=2, T2=5, T3=7, T4=13; T=D): FEASIBLE
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): FEASIBLE
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): FEASIBLE
amreeta@amreeta-VirtualBox:~/Downloads/Feasibility$
```

THE SCHEDULING POLICIES AS IMPLEMENTED IN THE MODIFIED CODE

	Feasibility Test (Code)			
	RM Completion Test	**RM Scheduling Point**	**Earliest Deadline First**	**Least Laxity First**
Ex_0	PASS	PASS	PASS	PASS
Ex_1	**FAIL**	**FAIL**	PASS	PASS
Ex_2	**FAIL**	**FAIL**	PASS	PASS
Ex_3	PASS	PASS	PASS	PASS
Ex_4	PASS	PASS	PASS	PASS
Ex_5	PASS	PASS	PASS	PASS
Ex_6	**FAIL**	**FAIL**	PASS	PASS
Ex_7	PASS	PASS	PASS	PASS
Ex_8	**FAIL**	**FAIL**	PASS	PASS

FEASIBILITY TEST (CODE) RESULTS

	Cheddar 3.1 (Windows)							
	Rate Monotonic		**Earliest Deadline First**		**Least Laxity First**		**Deadline Monotonic**	
	Simulation	Feasibility	Simulation	Feasibility	Simulation	Feasibility	Simulation	Feasibility
Ex_0	PASS	PASS	PASS	PASS	PASS	PASS		
Ex_1	**FAIL**	**FAIL**	PASS	PASS	**FAIL**	PASS		
Ex_2	**FAIL**	**FAIL**	PASS	PASS	**FAIL**	PASS		
Ex_3	PASS	PASS	PASS	PASS	PASS	PASS		
Ex_4	PASS	PASS	PASS	PASS	PASS	PASS		
Ex_5	PASS	PASS	PASS	PASS	PASS	PASS		
Ex_6	**FAIL**	**FAIL**	PASS	PASS	**FAIL**	PASS	**FAIL**	**FAIL**
Ex_7	PASS	PASS	PASS	PASS	PASS	PASS		
Ex_8	**FAIL**	**FAIL**	PASS	PASS	**FAIL**	PASS		

TESTS ON DIFFERENT SCHEDULING POLICIES ON CHEDDAR

- Rate Monotonic Tests fail for Example 1, 2, 6 and 8 while EDF and LLF tests pass for those examples.
- Reasons why Rate Monotonic Tests fails and EDF and LLF passes:
  - Rate Monotonic is Fixed Priority Scheduling whereas Earliest Deadline First and Least Laxity First are categorized under Dynamic Priority Scheduling. In Dynamic Priority scheduling policies, whenever an additional thread is placed in the ready queue, the scheduler reevaluates all dispatch priorities. This does not happen in Rate Monotonic.
  - Dynamic policies can achieve almost one CPU utilization while fixed priority policies can't.
  - Another important reason is that the Rate Monotonic is a Sufficient feasibility test, so it will always fail a service set that can miss deadlines. However, it also fails a service set that is real-time safe occasionally. So, RM feasibility test is not precise. If a service test fails Rate Monotonic Feasibility test then in reality, it can either be schedulable or not schedulable.
  - However, Earliest Deadline First and Least Laxity First are Necessary and Sufficient tests. Completion Tests and Scheduling Point tests (based on Rate Monotonic) are also Necessary and Sufficient (N&S) tests. Unlike Sufficient tests, N&S feasibility tests are precise. If a service set fails this test, then it means that it definitely is not schedulable. And if it does pass this test, then the service set will be schedulable.
  - Service sets passing Sufficient tests can be considered as a subset of service sets passing Necessary and Sufficient tests; which in turn can be considered as a subset of all service sets.
- In example 2 and 8, schedule generated for LLF policy is incorrect and thus shows that as not schedulable, even though its feasibility test passes. So, we have also used Cheddar 2.1 for this example which generated the correct schedule.
- In example 6, Deadline Monotonic fails to complete the service before the deadline 67 so it is indicated as a failure.

BUGS IDENTIFIED IN CHEDDAR 3.1:

The scheduling simulation shown on Cheddar 3.1 for Least Laxity First is wrong for example 1, 2, 6 and 8 and so it is shown to FAIL. However, it should actually pass, if simulation is done correctly. The feasibility test for LLF is done correctly. LLF test for example 1, 2 and 6 in Cheddar 2.1 is attached in the .zip folder which shows that the scheduling simulation passes for the service test. The schedule in the modified code matches the Cheddar 3.1 schedule except in the case of 1, 2, 6, and 8 due to this bug. Cheddar 2.1 does not have this error.

QUESTION 5

Three assumptions in the Liu and Layland paper:

- All tasks requests are periodic in nature, with constant interval between requests.
- Each task must be completed before the next request for it occurs. It basically means that completion time should be less than the task period.
- The tasks requests are independent in nature and are not dependent on the initiation or the completion of requests for other tasks.

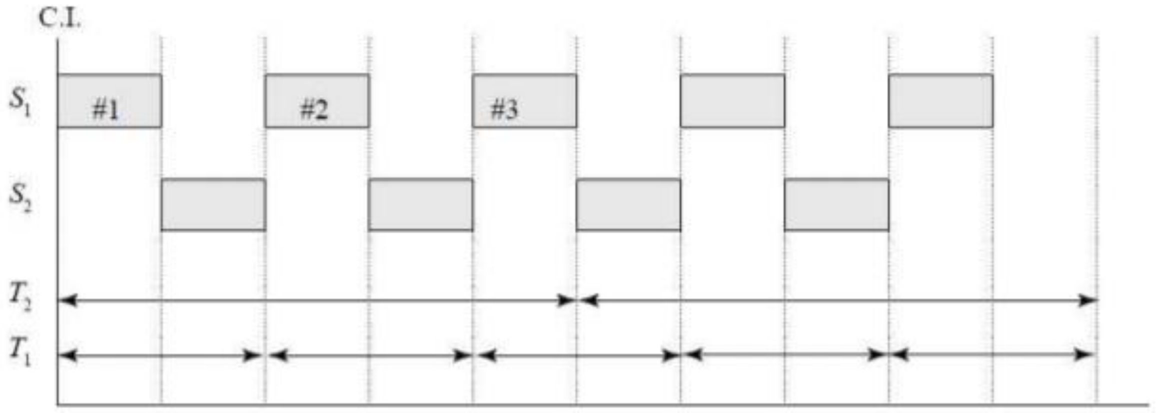
Three constraints that are made on the RM LUB derivation:

- The task deadline is equal to the task period. Factors like jitter, input latency, output latency are not taken into consideration.
- Rate Monotonic scheduling policy is restricted to fixed priority tasks and preemptive tasks. This limits the tasks with dynamic priority and non-preemptive tasks.
- The CPU utilization factor (U = m($2^{1/m}$-1)) is only applicable when the ratio between the request period of any two tasks from a give set of tasks is less than 2 i.e. $\frac{T_i}{T_j} \leq 2$, where $T_i > T_j$.

Three key derivation steps:

1. In Rate Monotonic Least Upper Bound derivation, two cases have been taken into consideration.

   The first case is the one in which C1 is short enough to fit the three releases.

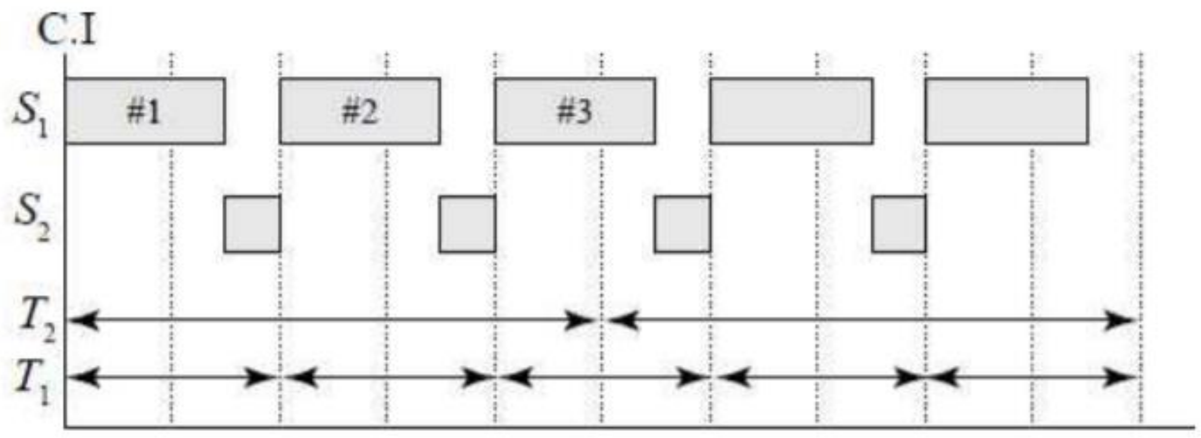


$$Eq\text{–}1 : C_1 \leq T_2 - T_1\lfloor T_2/T_1 \rfloor$$

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

$$Eq\text{–}3 : U = \frac{C_1}{T_1} + \frac{C_2}{T_2}$$

In the second case, C1 is a bit large to fit in the critical time zone.

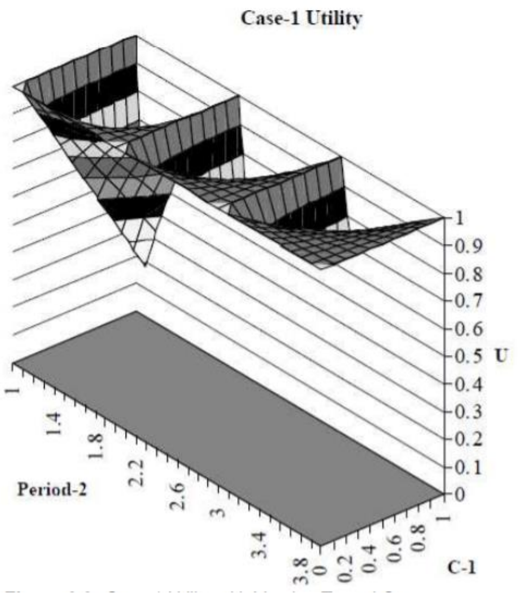


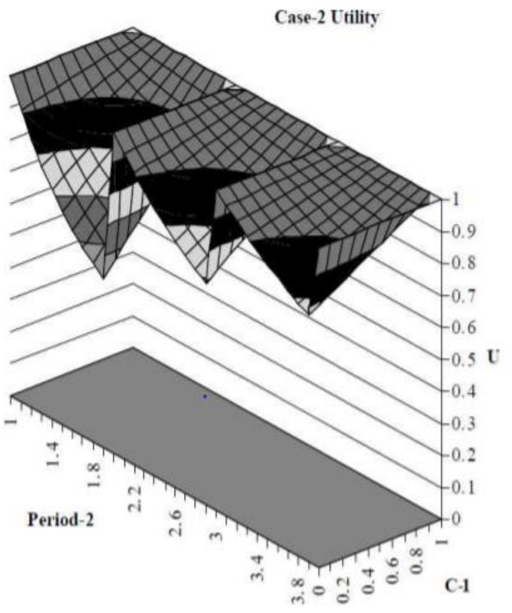
$$C_1 \geq T_2 - T_1 \lfloor T_2 / T_1 \rfloor$$

$$C_2 = T_1 \lfloor T_2 / T_1 \rfloor - C_1 \lfloor T_2 / T_1 \rfloor$$

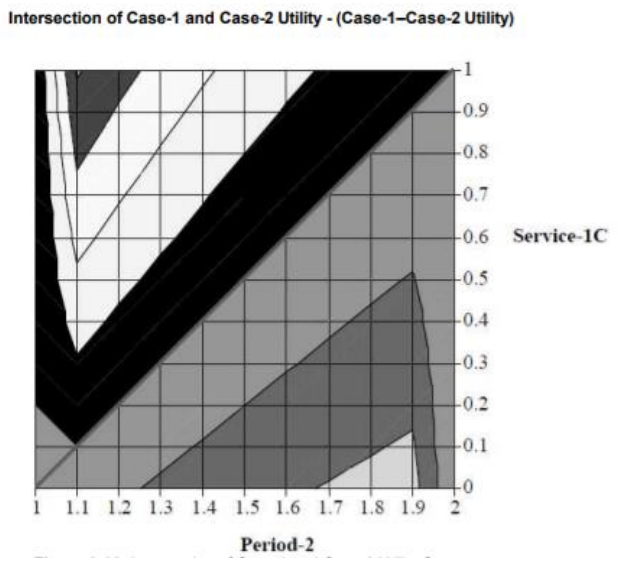
$$U = \frac{C_1}{T_1} + \frac{\left[T_1 \lfloor T_2 / T_1 \rfloor - C_1 \lfloor T_2 / T_1 \rfloor \right]}{T_2}$$

For a generalized case which satisfies both cases, we must take the intersection of Case 1 and Case 2. The curves for both the cases are given below:





The difference in the two curves where the function difference is zero, will give us the intersection of the two curves for generation of a standard equation.



The diagonal represents the area where the function difference is zero, so equation od the diagonal will give us the relation between T2 AND C1.

If we compare the two cases, C1 for both the cases are equivalent with a difference in the equality. Thus, the intersection will be:

$$C_1 = T_2 - T_1 \lfloor T_2 / T_1 \rfloor$$

The diagonal above gives us the relation between C1 and T2 which can be substituted above, to obtain the relation between T2 and T1. This relation turns out to be theorem 4 of Liu and Layland paper which is T2/T1 < 2.

Next common relation for both the cases, as per book, for C2 is:

$$C_2 = T_2 - C_1 \lceil T_2 / T_1 \rceil$$

We didn’t understand how this equation represents intersection of the two C2 equations of the respective cases.

2. To determine the utility for the generalized case, we can substitute the values of C1 and C2 in the equation below:

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

On substitution of the two values and on simplification of the equation, we get

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

Next, 'I' which represents number of interferences can be written as the floor value of T2/T1 and f which represents the fractional interference can be written as

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

On substitution of 'I' and f in the above equation, we get

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

Now to determine the least upper bound for utility, we must minimize the utility. We were unable to understand why 'I' must be minimized for minimization of utility and the reason behind the substitution of 1 for 'I'.

3. On substituting 1 in the place of 'I', we get:

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

To find the minima of utility, its derivative should be determined and equated to zero.

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

Post this, we get a value for f which is substituted back into the above equation to get the value of U which is:

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

We were unable to understand the how this differentiation and its substitution is exactly performed.

REFERENCES

- "Architecture of the Space Shuttle Primary Avionics Software System" - Gene D. Carlow.
- “Building safety-critical real-time systems with reusable cyclic executives”- J. Zamorano, A. Alonso, J.A. de la Puente.
- “Scheduling Algorithms for Multiprogramming in a Hard-Real Time Environment”- C.L Liu, James W. Layland
- “Real Time Embedded Components and Systems with Linux and RTOS”- Sam Siewert, John Pratt.