Karros Huang
Sarthak Jain

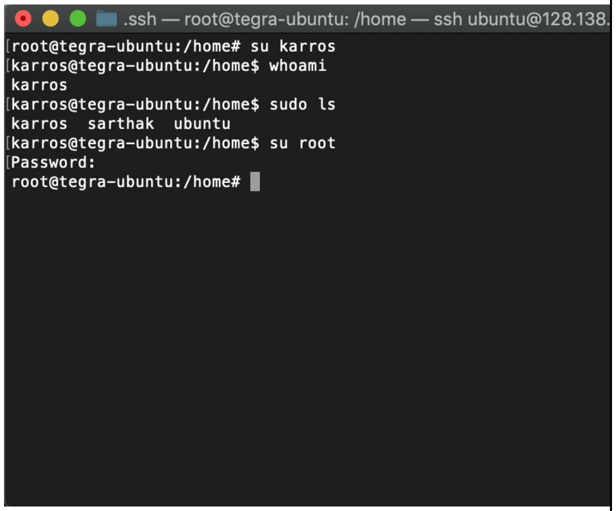
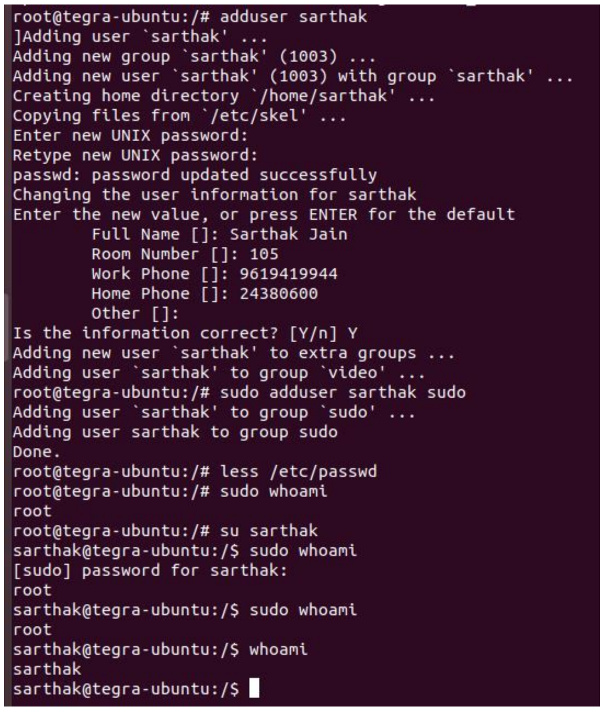
ECEN 5623, Real-Time Embedded Systems: Exercise #2 – Service Scheduling Feasibility

Programs executed on Jetson TKI by NVIDIA

12/21/19

Exercise #2:

1) *[5 points] make yourself an account on your Dev Kit. To do this, use the reset button if the system is locked, use your password to login, and then use "sudo adduser", enter a password, and enter user information as you see fit. Add your new user account as a "sudoer" using "visudo" right below root with the same privileges (if you need help with "vi", here's a quick reference or reference card– use arrows to position cursor, below root hit Esc, "i" for insert, type username and privileges as above, and when done, Esc, ":", "wq"). The old unix vi editor was one of the first full-screen visual editors – it still has the advantage of being found on virtually any Unix system in existence, but is otherwise cryptic – along with Emacs it is still widely used in IT, by developers and systems engineers, so it's good to know the basics. If you really don't like vi or Emacs, your next best bet is "nano" for Unix systems. Do a quick "sudo whoami" to demonstrate success. Logout of Linux and test your login, then logout. Use Alt+Print-Screen to capture your desktop and save as proof you set up your account. Note that you can always get a terminal with Ctrl+Alt+t key combination. If you don't like the desktop, you can try "GNOME Flashback" and please play around with customizing your account as you wish.*

User1: Karros	User2: Sarthak
	

```
.ssh — root@tegra-ubuntu: /home — ssh ubuntu@128.138.
[root@tegra-ubuntu:/home# su karros
[karros@tegra-ubuntu:/home$ whoami
karros
[karros@tegra-ubuntu:/home$ sudo ls
karros  sarthak  ubuntu
[karros@tegra-ubuntu:/home$ su root
[Password:
root@tegra-ubuntu:/home#
```

```
root@tegra-ubuntu:/# adduser sarthak
]Adding user `sarthak' ...
Adding new group `sarthak' (1003) ...
Adding new user `sarthak' (1003) with group `sarthak' ...
Creating home directory `/home/sarthak' ...
Copying files from `/etc/skel' ...
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Changing the user information for sarthak
Enter the new value, or press ENTER for the default
        Full Name []: Sarthak Jain
        Room Number []: 105
        Work Phone []: 9619419944
        Home Phone []: 24380600
        Other []:
Is the information correct? [Y/n] Y
Adding new user `sarthak' to extra groups ...
Adding user `sarthak' to group `video' ...
root@tegra-ubuntu:/# sudo adduser sarthak sudo
Adding user `sarthak' to group `sudo' ...
Adding user sarthak to group sudo
Done.
root@tegra-ubuntu:/# less /etc/passwd
root@tegra-ubuntu:/# sudo whoami
root
root@tegra-ubuntu:/# su sarthak
sarthak@tegra-ubuntu:/$ sudo whoami
[sudo] password for sarthak:
root
sarthak@tegra-ubuntu:/$ sudo whoami
root
sarthak@tegra-ubuntu:/$ whoami
sarthak
sarthak@tegra-ubuntu:/$
```

Karros Huang
Sarthak Jain

2) [10 points] Read the paper "Architecture of the Space Shuttle Primary Avionics Software System" [available on D2L], by Gene Carlow and provide an explanation and critique of the frequency executive architecture. 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.

Summary

The NASA's Space Shuttle Orbiter Primary Avionics Software System employs several different application process and redundancy management systems on very limited memory. In fact the entire code base is too large to be stored on the system's main memory. All software configuration is stored onto a mass memory unit that's divided into 8 key sections known as operational sequences (OPS) for all 8 stages of flight. Each section is loaded onto the main memory based on user input. Each application process within each OPS is handled by a operating system which handles process management, redundancy management, and I/O management. Scheduling of each one of these application process is synchronous and is run on a frequency executive architecture. Therefore the schedule is deterministic, redundant, repeatable, and reliable. These are all qualities placed on very high importance where there is no tolerance for any risk or failures for a system like a spacecraft.

Advantages

- The frequency executive architecture is cyclic, such that the executions of applications always happen at a set time after the period begins and is synchronous. This enforces repeatability and ensures that the program doesn't get stalled up on the process of an application experiencing runtime error.
- The cyclic architecture allows the employment of a redundancy management system, ensuring that synchronization is kept between all application processes.
- The fixed priority interrupt scheduling algorithm is preferable over others like time slice preemption since even in failure, the former degrades gracefully, allowing higher priority tasks to still execute and neglecting the lower priority tasks. Any failure in the time slicing algorithm would lead to breakdown in execution of all the tasks.

Disadvantages

- Coordination of I/0 operations and intercommunications between the GPC is asynchronous, therefore leading to potential data race problems during the start of new application processes and therefore cause overruns at both the process and system levels
- There's usually a predetermined fixed amount of time for a cycle loop based on the mission requirements, such as the "Orbit Insertion Burn Stage" lasting only five minutes. This means that all application processes must run to completion within this five minute window. However, sometimes you will have a set of processes that will be difficult to break down into blocks that all fit within the window. Therefore, creating a feasible frequency executive scheduler can be very difficult given certain constraints.

Karros Huang
Sarthak Jain

- There is limited flexibility to add changes to the frequency executive architecture without affecting the entire application process system because the schedule of each application process is entirely dependent on one another. Changing the deadlines/period of one process will the deadline of every other process. Therefore it’s difficult to update your schedule as the scope of your program changes over time.

Karros Huang
Sarthak Jain

3) [5 points] Read the paper "Building Safety-Critical Real-Time Systems with Reusable Cyclic Executives", available from http://dx.doi.org/10.1016/S0967-0661(97)00088-9. In other embedded systems classes you built ISR (Interrupt Service Routine) processing software and polling/control loops to control for example stepper motors – describe the concept of the Cyclic Executive and how this compares to the Linux POSIX RT threading and RTOS approaches we have discussed.

Summary

The cyclic executive is a synchronous scheduler comprised of major cycle periods, minor cycle periods, and frames. A major cycle period denotes the entire schedule, and the operation repeats at the end of each period until the system is changed. Within major cycles are minor cycles which describe the process sequence that executes during each minor cycle. At the end of every minor cycle, a minor clock ticks which synchronizes all the minor cycles together. Frames are embedded in the minor cycles and each frame contains only one process. Durations of the periods are set and fixed to the programs requirements, guaranteeing processing and predictable deadlines. If a process is doesn't complete within its period, then a overrun exceptions can be made to abort the process and begin recovery.

Comparison

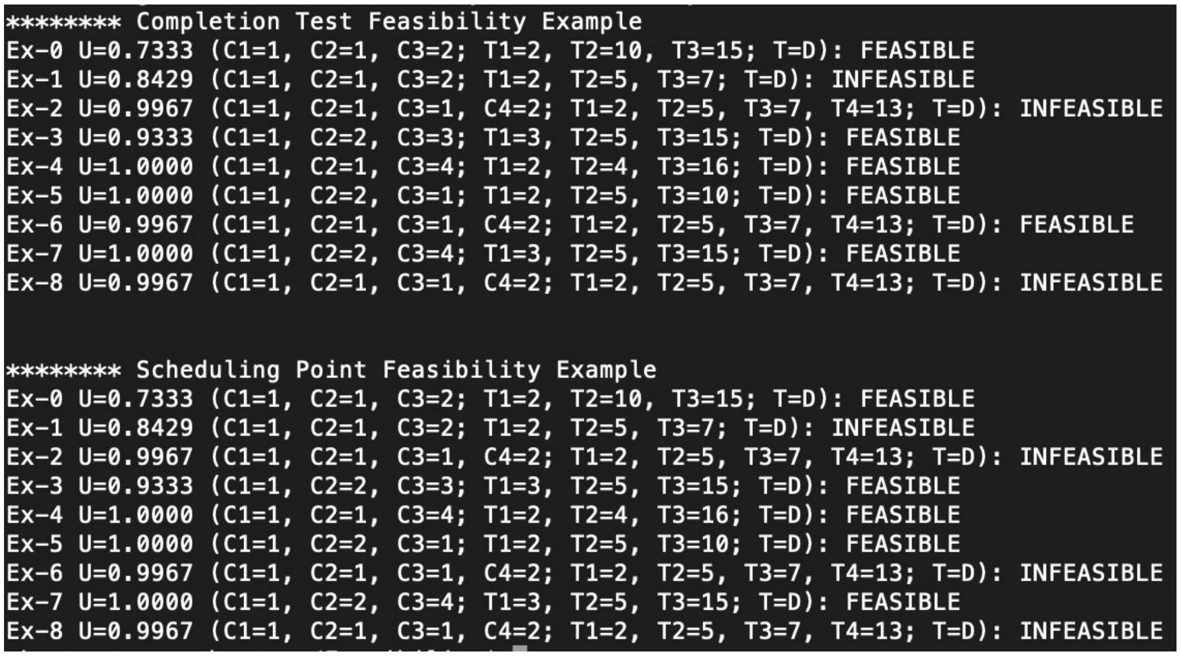
We have implemented programs that run a ISR processing software and polling/control loops to control things like a stepping motor. If we were to reimplement that with a cyclic executive, it would comprise of creating several frames for each processes (i.e. polling ADC data, arithmetic computation and conversion, generating PWM signal from DAC, etc,) and allocating a set deadline for each one of of those processes. So instead of having our program halted on polling ADC data, it would poll for ADC data for a deterministic amount of time, and then after begin arithmetic conversions for a deterministic amount of time, and then generate PWM signal, etc. The cyclic executive shares a lot of similarities with POSIX RT & RTOS threads, such that application processes are broken down into frames, or threads, and then can be scheduled. The difference is that POSIX RT & RTOS support dynamic priority scheduling and therefore schedule designs such as earliest deadline first or least laxity first are possible. However, all the before mentioned is possible because processes can be preempted at arbitrary points. The executive would require much more complex timing mechanisms to add preemption and context switching functionality which would add too much execution overhead for a real time system.

Karros Huang
Sarthak Jain

4) [50 points] Download Feasibility example code and build it on a Jetson, DE1-SoC or TIVA or Virtual Box and execute the code. Compare the tests provided to analysis using Cheddar for the first 4 examples. Now, implement the remaining examples [5 more] that we reviewed in class (found here). Complete analysis for all three policies using Cheddar (RM, EDF, LLF). 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". Does your modified Feasibility code agree with Cheddar analysis in all 5 additional cases? Why or why not?

To test and validate the feasibility of the 9 different schedules (EX-0 - EX-8). We compiled and ran the following files on the Nvidia Jetson TK1:
feasibility_tests.c
llf.c
edf.c

Originally *feasibility_tests.c* only contained examples EX-0 - EX-4, but we modified it so that we also included EX-5 - EX-8. *llf.c* & *edf.c* was implemented separately and includes feasibility tests for EX-0 - EX-8. The screenshots below show the output of the 3 programs.

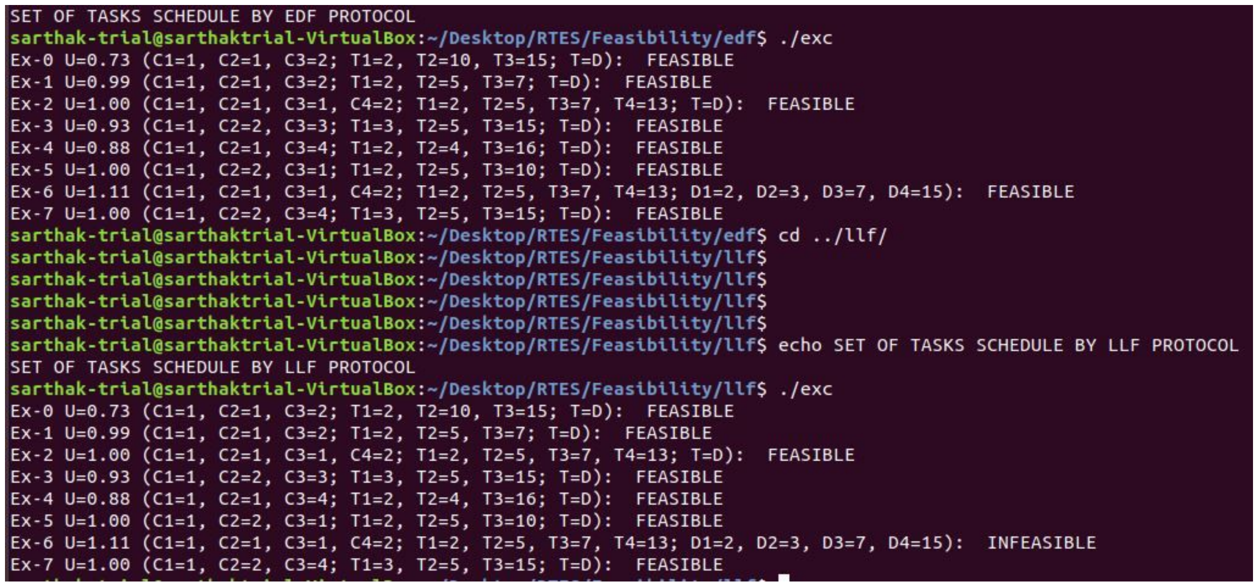


```
******** Completion Test Feasibility Example
Ex-0 U=0.7333 (C1=1, C2=1, C3=2; T1=2, T2=10, T3=15; T=D): FEASIBLE
Ex-1 U=0.8429 (C1=1, C2=1, C3=2; T1=2, T2=5, T3=7; T=D): INFEASIBLE
Ex-2 U=0.9967 (C1=1, C2=1, C3=1, C4=2; T1=2, T2=5, T3=7, T4=13; T=D): INFEASIBLE
Ex-3 U=0.9333 (C1=1, C2=2, C3=3; T1=3, T2=5, T3=15; T=D): FEASIBLE
Ex-4 U=1.0000 (C1=1, C2=1, C3=4; T1=2, T2=4, T3=16; T=D): FEASIBLE
Ex-5 U=1.0000 (C1=1, C2=2, C3=1; T1=2, T2=5, T3=10; T=D): FEASIBLE
Ex-6 U=0.9967 (C1=1, C2=1, C3=1, C4=2; T1=2, T2=5, T3=7, T4=13; T=D): FEASIBLE
Ex-7 U=1.0000 (C1=1, C2=2, C3=4; T1=3, T2=5, T3=15; T=D): FEASIBLE
Ex-8 U=0.9967 (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=0.7333 (C1=1, C2=1, C3=2; T1=2, T2=10, T3=15; T=D): FEASIBLE
Ex-1 U=0.8429 (C1=1, C2=1, C3=2; T1=2, T2=5, T3=7; T=D): INFEASIBLE
Ex-2 U=0.9967 (C1=1, C2=1, C3=1, C4=2; T1=2, T2=5, T3=7, T4=13; T=D): INFEASIBLE
Ex-3 U=0.9333 (C1=1, C2=2, C3=3; T1=3, T2=5, T3=15; T=D): FEASIBLE
Ex-4 U=1.0000 (C1=1, C2=1, C3=4; T1=2, T2=4, T3=16; T=D): FEASIBLE
Ex-5 U=1.0000 (C1=1, C2=2, C3=1; T1=2, T2=5, T3=10; T=D): FEASIBLE
Ex-6 U=0.9967 (C1=1, C2=1, C3=1, C4=2; T1=2, T2=5, T3=7, T4=13; T=D): INFEASIBLE
Ex-7 U=1.0000 (C1=1, C2=2, C3=4; T1=3, T2=5, T3=15; T=D): FEASIBLE
Ex-8 U=0.9967 (C1=1, C2=1, C3=1, C4=2; T1=2, T2=5, T3=7, T4=13; T=D): INFEASIBLE
```

Tasks scheduled with RM

Karros Huang
Sarthak Jain



```
SET OF TASKS SCHEDULE BY EDF PROTOCOL
sarthak-trial@sarthaktrial-VirtualBox:~/Desktop/RTES/Feasibility/edf$ ./exc
Ex-0 U=0.73 (C1=1, C2=1, C3=2; T1=2, T2=10, T3=15; T=D):  FEASIBLE
Ex-1 U=0.99 (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=0.88 (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=1.11 (C1=1, C2=1, C3=1, C4=2; T1=2, T2=5, T3=7, T4=13; D1=2, D2=3, D3=7, D4=15):  FEASIBLE
Ex-7 U=1.00 (C1=1, C2=2, C3=4; T1=3, T2=5, T3=15; T=D):  FEASIBLE
sarthak-trial@sarthaktrial-VirtualBox:~/Desktop/RTES/Feasibility/edf$ cd ../llf/
sarthak-trial@sarthaktrial-VirtualBox:~/Desktop/RTES/Feasibility/llf$
sarthak-trial@sarthaktrial-VirtualBox:~/Desktop/RTES/Feasibility/llf$
sarthak-trial@sarthaktrial-VirtualBox:~/Desktop/RTES/Feasibility/llf$
sarthak-trial@sarthaktrial-VirtualBox:~/Desktop/RTES/Feasibility/llf$
sarthak-trial@sarthaktrial-VirtualBox:~/Desktop/RTES/Feasibility/llf$ echo SET OF TASKS SCHEDULE BY LLF PROTOCOL
SET OF TASKS SCHEDULE BY LLF PROTOCOL
sarthak-trial@sarthaktrial-VirtualBox:~/Desktop/RTES/Feasibility/llf$ ./exc
Ex-0 U=0.73 (C1=1, C2=1, C3=2; T1=2, T2=10, T3=15; T=D):  FEASIBLE
Ex-1 U=0.99 (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=0.88 (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=1.11 (C1=1, C2=1, C3=1, C4=2; T1=2, T2=5, T3=7, T4=13; D1=2, D2=3, D3=7, D4=15):  INFEASIBLE
Ex-7 U=1.00 (C1=1, C2=2, C3=4; T1=3, T2=5, T3=15; T=D):  FEASIBLE
```

Tasks scheduled with EDF and LLF

Then we downloaded Cheddar v3.0 and simulated EX-0 - EX-8 on the software. Screenshots of the schedules for Cheddar have been attached in a separate .zip file. Below is a list of the comparisons of the outputs from our program vs. Cheddar.

1. Example 0: Tests provided agree with analysis using Cheddar, for all three scheduling protocols.
2. Example 1: Rate Monotonic protocol fails this test, whereas Earliest Deadline First and Least Laxity First Protocols both pass the schedule. This holds true for both the tests provided as well as analysis using Cheddar. As to why RM protocol fails, it is a fixed priority scheduling algorithm. The advantage of using a dynamic priority scheduling algorithm is that it gives priority to the task having least deadline remaining, instead of a fixed priority like RM. We can also see from the scheduling screenshots provided where RM fails, and the difference in schedules plotted.

   A test of feasibility for EDF and LLF is that of utilization factor being less than or equal to 1. For this particular example, this holds true, and we can see from the screenshots above that U = 0.99, and is accordingly passed by both dynamic scheduling protocols. RM does not pass the same.

   Another point of note is that Cheddar v3.0 shows a mistake in the LLF scheduling simulation of this service set, by logging that some task deadlines will be missed, even though the generated scheduler did not have any missed deadlines. However, this is not in keeping with our schedules, and this fact is verified by Cheddar v2.0, the screenshot of which is provided as well.
3. Example 2: Rate Monotonic protocol fails this test as well, whereas Earliest Deadline First and Least Laxity First Protocols both pass the schedule. This holds true for both the

tests provided as well as analysis using Cheddar. As to why RM protocol fails, it is a fixed priority scheduling algorithm. The advantage of using a dynamic priority scheduling algorithm is that it gives priority to the task having least deadline remaining, instead of a fixed priority like RM. We can also see from the scheduling screenshots provided where RM fails, and the difference in schedules plotted.

A test of feasibility for EDF and LLF is that of utilization factor being less than or equal to 1. For this particular example, this holds true, and we can see from the screenshots above that U = 0.9967, and is accordingly passed by both dynamic scheduling protocols. RM does not pass the same.

Another point of note is that Cheddar v3.0 shows a mistake in the LLF scheduling simulation of this service set, by saying that some task deadlines will be missed. However, this is not in keeping with our schedules, and this fact is verified by Cheddar v2.0, the screenshot of which is provided as well.

4. Example 3: Tests provided agree with analysis using Cheddar, for all three scheduling protocols. A small point of note is that Cheddar admits, RM cannot prove anything in case of this example, as U = 0.93, whereas condition for RM Least Upper Bound is $U < 0.77976$.
5. Example 4: Tests provided agree with analysis using Cheddar, for all three scheduling protocols. Another possible bug in Cheddar 3.1 is that although U = 1, RM declares the service set as feasible because it's deriving a RM LUB of 1.0 instead of ln(2), and therefore U <= RM LUB.
6. Example 5: Tests provided agree with analysis using Cheddar, for all three scheduling protocols.
7. Example 6: Example 6 properties are quite similar to those of example 2, the only difference being that a deadline different to the period is provided in this set. RM fails completely for this set, as having a deadline equal to the period is a requirement for RM. DM fails for this service set, whereas EDF passes this set. It is worth noting that the output by EDF for this set and Example(2) set are different, which makes sense, seeing as the deadline has now changed. LLF, however marks this set as infeasible, which is verified by Cheddar 3.1. Another point worth noting is that Cheddar shows the total utilization of this task set as 1.11. We assume it calculates the utilization by the formula $U = C_i/D_i$, where D signifies Deadline. Although the necessary condition for utilization is $U<=1$, Cheddar as well as the output code are able to plot the schedule.
8. Example 7: Tests provided agree with analysis using Cheddar, for all three scheduling protocols.
9. Example 8 has a similar analysis as that for example 2, as both service sets have the same tasks with periods and run-times, therefore this schedule was not included in the program because running it would've produced the same results as Example 2.

Karros Huang
Sarthak Jain

5) [30 points] 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. 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.

Assumptions

- All tasks with hard deadlines are periodic.
- Each task must run to completion before the occurrence of the next request.
- Each task is independent from one another, meaning that the start of one task does not depend on the completion of another task.

Constraints

- The task deadline must equal its period.
- The scheduler's tasks are runs to completion, preempted, and priority of the tasks are fixed.
- For each task to run to completion before the occurrence of the next request, there must be buffering hardware for each peripheral function, and that control loops within the computer must be designed to handle latent conditions (i.e. allowing an extra unit sample of delay).

Tricky Derivation Steps

- $U = 1 - f(1-f)/(I+f) \rightarrow f = (2^{1/2} - 1)$
  Liu & Layland come to this derivation from the fact that U is monotonically increasing with I, and the minimum of U occurs when I = 1. So therefore by minimizing U over f, we get the relation that $f = (2^{1/2} - 1)$. The steps to getting this key relationship is very ambiguous. When we tried to work this out, we took the original equation, U = 1 - f(1-f)/(I+f), substituted I = 1, and then took the derivative of U in respect to f, and set it equal to 0 to find the values of f that produce minimum U. The corresponding f values we got were f = -1, f = 0.382, and f = 2.618, in which neither of these are f = $(2^{1/2} - 1)$ = 0.4142. So it's unclear how to they reached that conclusion.
- $C_1 = T_2 - T_1 + \Delta \qquad \Delta > 0$
  Based on the assumptions stated above, it's clear that the computation time needs to be completed before the next period starts, however the equation says that the computation time is the duration of the previous period plus some additional time. Which in effect would violate one of the assumptions and now it's not too clear to why this statement is made. Perhaps RM LUB derivation factors in computation times exceeding its period?
- $C_1' = T_2 - T_1$
  $C_2' = C_2 + \Delta$
  C_2 is nowhere previously defined. Did the paper mean $C_2' = C_1' + \Delta$? Because then this would agree with the previous equations.