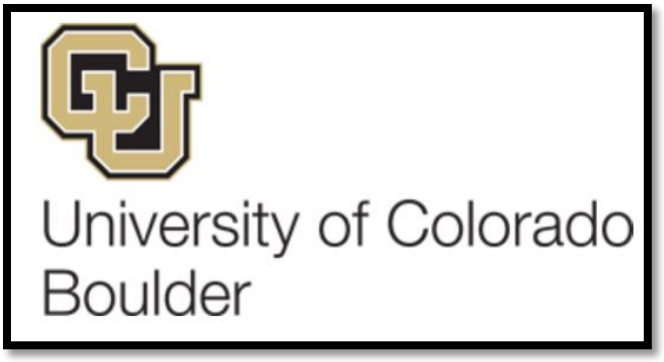

EXERCISE #6 – REAL-TIME SOFTWARE SYSTEMS

FINAL PROJECT

REAL TIME EMBEDDED SYSTEMS (ECEN 5623)

SUMMER 2019

BOARD USED:

NVIDIA's JETSON NANO DEVELOPMENT BOARD

PROJECT &REPORT BY:

OM RAHEJA

PROFESSOR:

DR. SAM SIEWERT

[NOTE: The answers for questions asked in Exercise #6 have been combined with the Final Project Report. All answers have been covered along with project report. This is done to avoid any repeatability of content in the document.

INTRODUCTION

BASIC UNDERSTANDING OF TIME LAPSE

The term *Time Lapse* refers to a technique where the frequency at which the images are captured is different from the frequency at which the images are viewed by the human eye. For example, the images would be captured from the camera at a frequency of 1 hertz, i.e. 1 frame per second and viewed at a frequency of 10 Hz, i.e. 10 frames per second. This will create an illusion of fast motion. The following image from Wikipedia explains how images are captured and displayed to portray fast motion of the images.

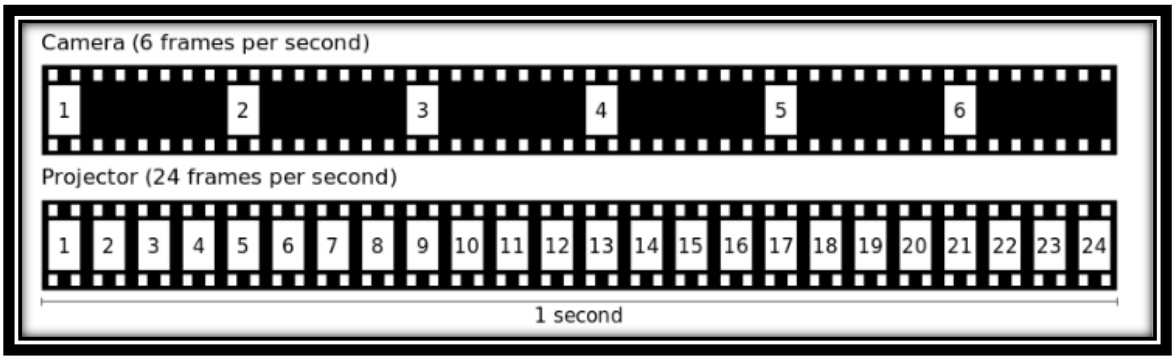

Fig 1: Time Lapse pictorial demonstration (Image Source: Wikipedia)

Let's break down the image shown above and see fundamentally how time lapse occurs. Let's say that the camera is capturing images at a rate of 6 frames per second. The first part of the image shows that in a time span of one second, 6 complete frames are captured. The projector on the other hand displays the images at a rate of 24 frames per second. This means that for human eyes the images will be viewed 4 times faster than normal rate, thus creating an illusion of things and objects moving fast in time. This is the basic concept behind time lapse. The change in the speed of the images can be given by the following formula,

$$\text{perceived speed} = \frac{\text{projection frame rate}}{\text{camera frame rate}} \times \text{actual speed}$$

.

In this report, I will be explaining the system design, flowchart and the general flow of the program for Time Lapse Image Acquisition. The system makes use of a Jetson Nano development board along with a Logitech C615 camera for image capturing. The basic overview of the project is to capture images at a certain frequency and convert the captured frames into mp4 video such that the video creates an illusion of time lapse, i.e. time moving at a faster rate than normal. The following pages will walk through a detailed description of the entire project's development, testing and debugging.

FUNCTIONAL REQUIREMENTS

Functional (capability) requirements for system design:

The entire project on Time Lapse Image Acquisition at two different frequencies has numerous functional capabilities. The functional capabilities are divided into two parts, namely, real time functional capabilities and non-real time functional capabilities.

List of real-time functional capabilities:

a. **Sequencer:**
The sequencer service schedules all the real-time threads in the system. It runs at either 1 hertz or 10 hertz depending upon the user input. All the real-time threads are created before the sequencer thread and programmed to wait on a semaphore unless the sequencer releases them. The sequencer releases the semaphores to all the real-time threads after a cycle of 1000ms (1hz) or 100ms (10hz). This timing is controlled by using a clock_nanosleep() C API.

b. **Image Capture:**
Purpose of this service: Image capturing is one of the vital and the most basic task of the project. This service runs at a frequency specified by the user at run-time. This frequency can be either 1 hertz or 10 hertz. The frames are acquired from the camera accordingly.
As described in the introduction of the final project report (attached below), the ppm file has two sections. One being the header section, consisting of the PPM identifier 'P6' along with the resolution, maximum pixel intensity value, host machine information and timestamps. The other part is the data, which consists the information captured by the camera (which happens in this service). The sequencer is programmed to run at 1hz or 10hz (depending upon the user input). The sequencer is responsible to schedule the real-time services. It does that by posting semaphores at every 1000 milliseconds (1 hertz) or 100 milliseconds (10 hertz), depending upon the frequency entered by the user. A clock_nanosleep(), which is a C API, is used to achieve the timings.
Verification: To check if the frames were being captured at the required frequency, I timestamped them. On examination of the timestamps, it was observed that each frame was captured after 1 second in case of 1 hertz and 100 milliseconds in case of 10 hertz.
The start time and end time of this service was also recorded to check the execution time. This was done for each iteration until all the frames were captured. By examining the execution times of each iteration, the worst-case execution time was calculated.

c. **Image Store:**
In this project of the time lapse image acquisition, a standard and simple system architecture would be to take the image captured by the camera and directly dump it into a ppm file. This ppm file can then be transferred to the host machine via ethernet. This design seems simple and might work for one hertz but would cause a problem when we run the program on ten hertz. The issue occurs with dumping the image into ppm format. It so happens that the ppm dump takes a little more time than usual when the memory sector of the flash is erased. This time taken might not cause a problem when we run the program on one hertz as the margin is a lot more when compared to code when ran at ten hertz. This might lead to missing deadlines. In order to rectify

this issue, we make use of a buffer so that it would provide elasticity in the system and the raw pixel data can be dumped into it and processed by a best effort service. This would solve the issue as now, the time taken to dump the pixel data into a ppm file would not result in missing deadlines as it's a best effort service. This service takes the data captured by the image capture service and stores it into a buffer, which can then be accessed by the best effort service to create a ppm file.

List of non-real time functional capabilities:

Starting from the minimum requirements and then moving towards target goals and stretch goals, I have listed all the functional capabilities incorporated in my project and the reason for selecting those.

a. **The resolution of the images that were captured is 640x480.**
   Reason: The amount of space required to store an image with a resolution of 640x480 is less when compared to that of an image having a resolution of 720x480 or 1280x960. As embedded systems have a limited amount of memory, resources should be used efficiently. An image with a resolution of 640x480 is sufficient and clear enough to observe the change in seconds hand of the clock at different frequencies (1Hz and 10Hz in this case).

b. **Timestamping on each image.**
   In order to accurately verify if an image is captured at the right instant, very precise timestamping needs to be done. I made use of the clock_gettime() API to get the start time and end time of each service. This operation is performed for all iterations until the requested number of frames are analyzed.

c. **Embedding Time stamps in the image.**
   The timestamps calculated at the start of each frame was successfully embedded in the header field of the ppm file. These timestamps can then be used to verify the correctness of the frame timing with timestamps embedded in another frames' header field. This ensures accuracy and correctness and helps in debugging. With the help of these timestamps, I also calculated the start time jitter in the next frame.

d. **Testing the code with external clock and a physical process.**
   The code was successfully tested for 2000 frames (1hz) and 6000 frames (10hz) with an external clock in the frame along with Ice melting. Images for 2000 frames being captured at 1hz and 6000 frames being captured at 10hz along with the mp4 videos for both cases have been attached in the submission folder. The ffmpeg command was used to convert the ppm images into a single mp4 format video to observe the time lapse.

d. **Image Dump.**
   This is a best effort service that reads the data from the circular buffer and converts it into a ppm file. The image store service posts a semaphore for this service as soon as it finishes completion.
   I made use of the v4l2 library functions to do all the image acquisition and image processing required in the project. Once the images are created in ppm format, this service releases a semaphore for the service that sends these images over the ethernet to a host machine.

e. Send Images via Socket.

After images have been dumped by the 'Image dump' service, it acquires a semaphore from the 'Image dump' service. This service, after acquiring the semaphore, reads the file created, from the flash and sends it over the ethernet. Once images are transferred, they can be verified for accuracy (for any data loss) by viewing the images on the host machine.

FUNCTIONAL DESIGN OVERVIEW AND DIAGRAMS

HIGH LEVEL SYSTEM AND SOFTWARE DESIGN

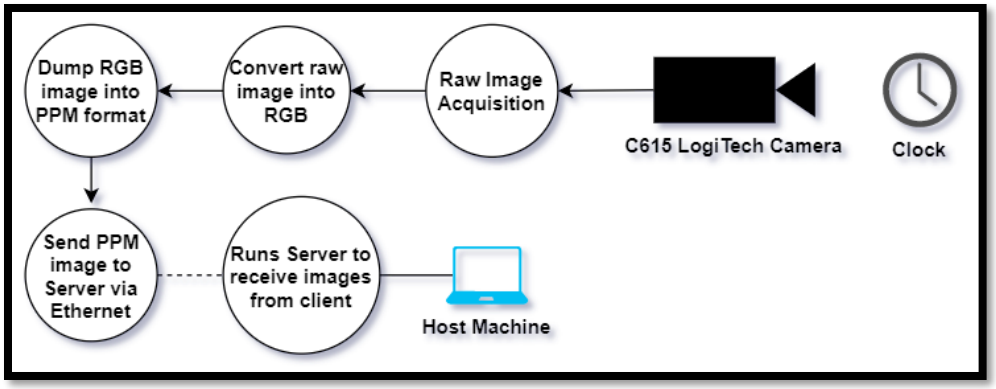

Fig 2: System Overview Diagram (Hardware)

The above figure, Fig 2, gives an idea about the high-level system design for the Time Lapse Image Acquisition project. The basic idea of the project is to capture images from a camera (Logitech C615 was used for the project) at a frequency of 1 hertz or 10 hertz. There are 5 parameters depending upon which the behavior of the program can change. These arguments are device path, number of frames to be analyzed, horizontal and vertical resolution of the image and the frequency at which the program should capture images. All the above-mentioned arguments are entered by the user in command line (more detail about this in the following sections). There is also an option of enabling or disabling a feature. This feature is sending an image over the ethernet to a different host machine. The user gets an option to enable or disable this feature at run-time.

There are a few key components that are used for the project. The two most important components of the project are the Logitech C615 camera and the Jetson Nano development board running Linux. In order to verify and test our program, a small table clock is required.

The basic functioning of the time lapse image acquisition project is as follows:

- The camera starts capturing the images at the specified frequency by the user. This can be either one hertz or ten hertz.
- The captured data is stored in a buffer which holds the data until further processing.
- The data that is stored in the buffer is then converted into RGB format and stored back into the buffer.
- The RGB image data is then copied into a circular buffer by another real-time service to decouple the I/O from the system.
- This data is later retrieved from the circular buffer and dumped into a ppm file to get the final image.
- This process is repeated until the specified number of frames are captured.
- Once all of the frames are captured, the analysis can be done by verifying timestamps in the /var/log/syslog directory as well as in the header section of the ppm file. The entire analyzed data is dumped into a csv file making the analysis easier than scrolling through the log file and the print statements.

- As soon as the images are converted into a ppm file, they are sent over from the target machine to the host machine via sockets. The target machine in this case acts as a client and the host machine acts as a server.
- The images can be viewed on the host machine after all the images have been sent over the ethernet to it.

SYSTEM ARCHITECTURE

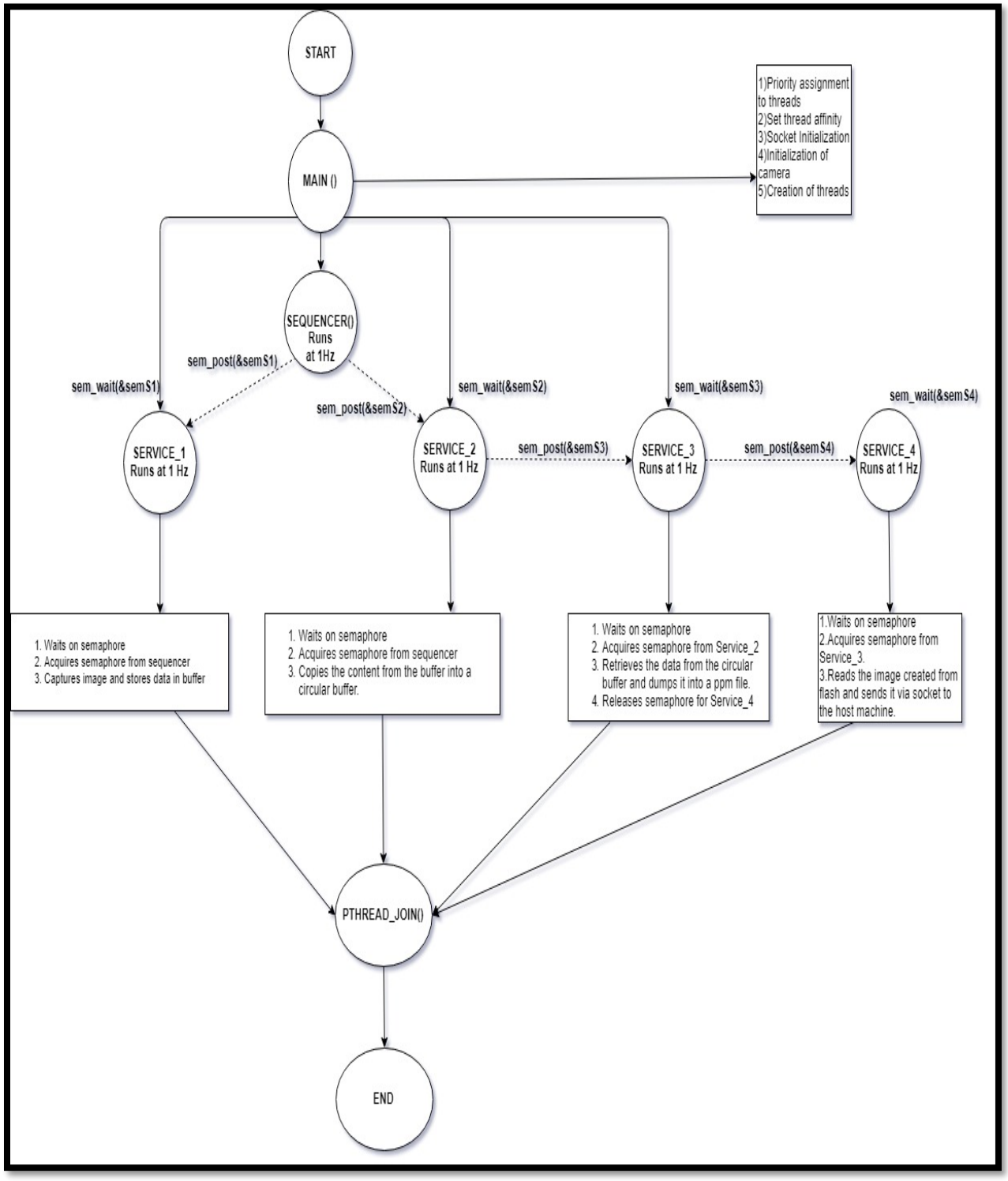

Fig 3: System Architecture (Software)

FLOWCHART

The flowchart has been divided into multiple sections for the convenience of explanation. The entire flowchart is broken down into four parts, each explaining the flow of data in that particular thread.

MAIN THREAD:

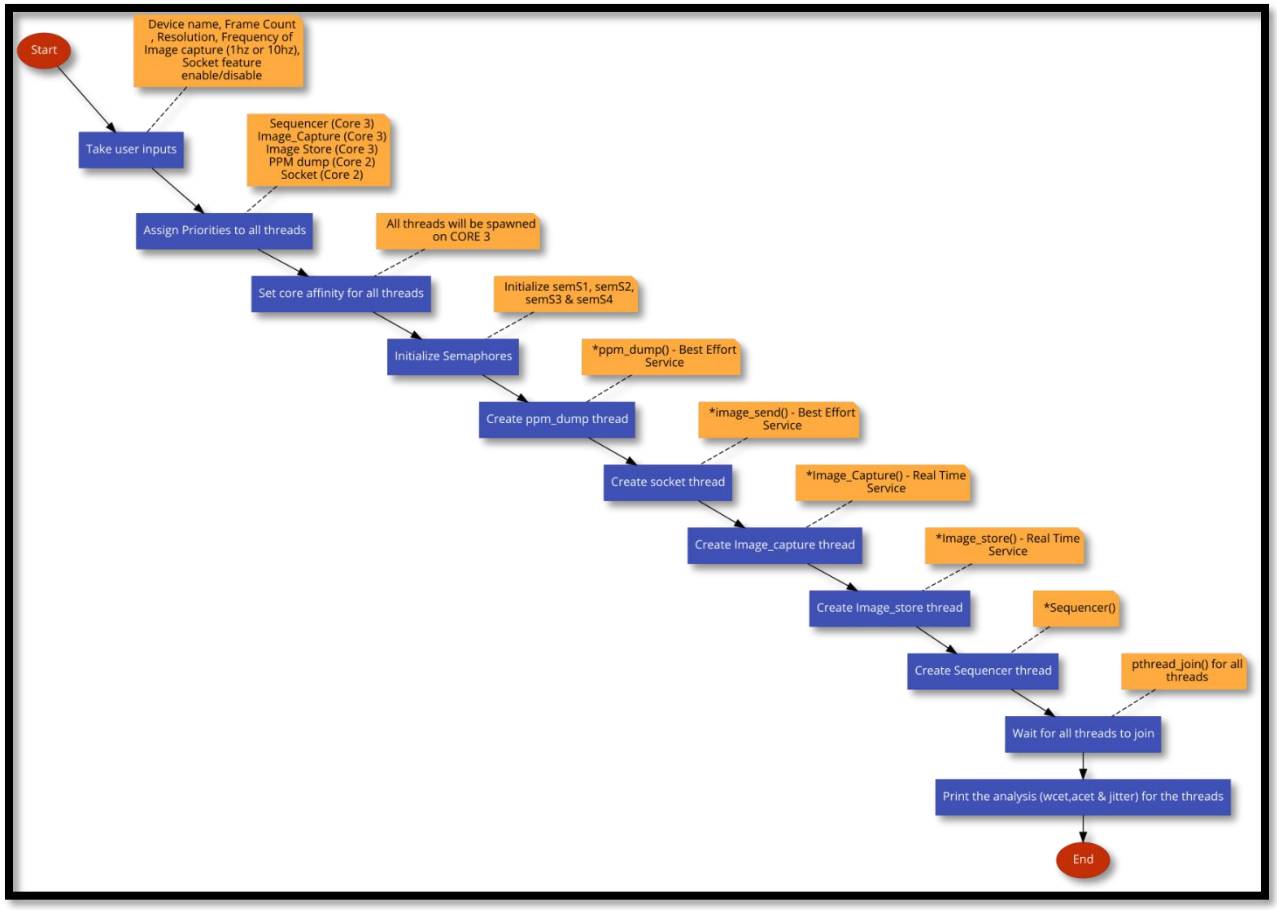

Fig 4: Flowchart (main.c)

The main thread carries out a number of tasks to get the program foundation up and running. The device path, frame count, resolution and frequency of capturing frame are taken as user inputs. The user also has the privilege to enable or disable a socket feature which transfers the images via sockets to a host machine. It is made sure that all the real-time threads run on the same core. For this, CPU affinity is set such that all the threads are bound to the same core. Semaphores are used as synchronization mechanisms so that the program behavior is deterministic. After assigning the priorities and setting the CPU affinity for all threads, the threads are created. First, the best effort threads, ppm dump and the socket services are created which are made to wait on a semaphore. Once the best effort threads are created, they are followed by creation of real-time threads. There are two real-time threads apart from the sequencer, namely, image capture thread and the image storage thread. The sequencer thread, running at the same frequency as the remaining real-time threads, i.e. 1 hertz or 10 hertz, is then spawned, which releases the semaphores for all the real-time threads. The threads waiting on the semaphore, acquire it and run to completion according to FIFO scheme. The sequencer, along with the other real time

threads runs until the requested number of frames are captured. The main thread waits on pthread_join() for all the threads to complete their execution.

IMAGE CAPTURE THREAD [SERVICE 1]:

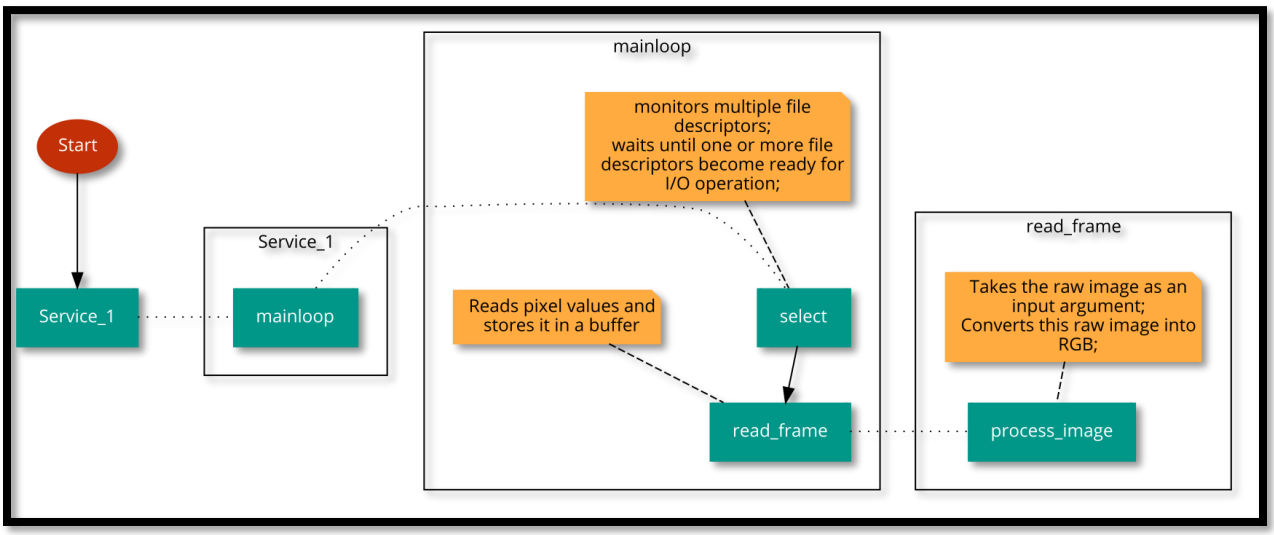

Fig 5: Flowchart (image capture service)

The image capture thread, as the name suggests, captures the surrounding environment and processes them in RGB format. This thread waits on the semaphore unless the sequencer releases it at every 1 hertz or 10 hertz. Thus, after every one second or hundred milliseconds, this thread will be given the semaphore and will eventually capture the image and store the data in a buffer. Time stamps at the start and end of each cycle are logged in order to ensure proper and appropriate functioning of the thread.

IMAGE STORE [SERVICE 2]:

Fig 6: Flowchart (image store service)

The image store service is responsible for retrieving the data from a buffer and dumping in into a circular/ring buffer. This is a real-time service which releases the semaphore to a best effort service running on a different core. This is done to get rid off the excess time taken during the flash erase. If this section is excluded from the system, a lot of frames can be missed due to the time taken by image dump thread during flash erase. In order to ensure that none of the frames are missed, a circular buffer is implemented to provide elasticity in the system.

PPM DUMP [SERVICE 3]:

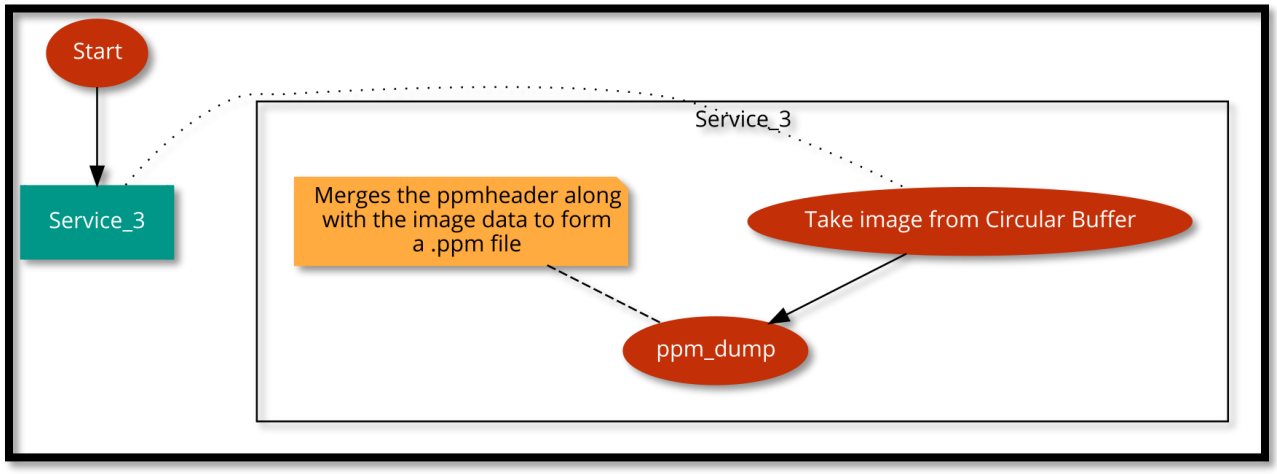

Fig 6: Flowchart (image dump service)

The ppm dump service acquires the data from the circular buffer that was stored by the image store, and then converts it into ppm file. The ppm file format consists of two parts, one being the ppm header and the other being the data. PPM header of an image that was captured during the testing of code is shown below.

The first line of the ppm header signifies the PPM identifier. This PPM identifier can be either P3 or P6. The second line consists of a timestamp indicating at what instance in time, that image was taken. The next line displays the resolution of the image. This image has a resolution of 640x480. The last line indicates the maximum value of the color components for the pixels.

```
root@om:/home/om/Desktop/version3_modular/1Hz_Images_2000_frames# head -n 4 Test00000036.ppm
P6
#1565283546 sec 0000000436msec
640 480
255
```

Fig 7: Header field of the ppm file

SEND IMAGE OVER ETHERNET [SERVICE 4]:

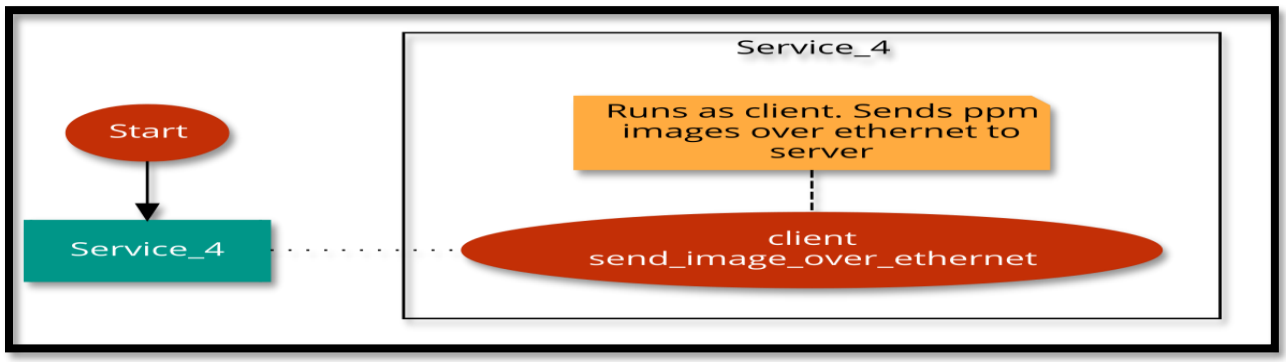

Fig 8: Flowchart (image sending over ethernet)

This service reads all the ppm files created by service 3 and sends it over the ethernet to a remote server. The service acts as a client and a server runs on a separate host machine. The client, running on the target machine sends all the images to the server. This is the usual case in embedded systems that have a finite amount of resources like memory. Rather than storing the entire data on the target machine, the data is sent over to a host machine via a suitable protocol. In this case, I am making use of the ethernet to transfer these images to the host machine.

REAL-TIME ANALYSIS AND DESIGN WITH TIMING DIAGRAMS

The system design consists of three real-time services. The sequencer is the highest priority service that acts as a scheduler for the entire system. It runs at 1hz/10hz and releases the semaphore at the same frequency at which it is running. After the sequencer releases the semaphore, it sleeps for the remaining time. This remaining time will vary depending upon the frequency at which it is running (1 hertz or 10 hertz). This cycle repeats until all the frames are captured. After semaphores are released by the sequencer, the other threads waiting in the queue gets executed according to the Rate monotonic policy, i.e. highest frequency thread runs first. In this case, as the frequency of all the services is same, the thread with the highest priority will be given preference over the others in the queue.

The service having a priority one less than that of the sequencer is the image capture service. This service waits on a semaphore that is released by the sequencer. Once this service grabs the semaphore, it captures the data using the camera and converts it into RGB format. This is then stored in a buffer for service 3 (image store service) to transfer it into a circular buffer. The last thread, i.e. the service which sends the ppm images via ethernet to the host machine has the least priority among all the services stated above.

This section will give a little numerical statistic regarding the timing of each service and whether or not these services can be used to build a feasible system.

REAL TIME SERVICES AND REQUIREMENTS

Sequencer:

As per the requirements mentioned for the project, the images need to be captured at 1 hertz and 10 hertz. In order to accomplish this, we would need to decouple the input and output with the system. This is done in order to get rid of the delay caused by the system because of sector erase. This delay may not cause a deadline miss in case of 1 hertz as the slack time is sufficient enough to hide this flaw. But, when it comes to 10 hertz, the slack time reduces to $1/10^{th}$ of a second, i.e. 100 milliseconds. Thus, I/O decoupling is an essential feature to add in the system for getting a functionally correct and a deterministic output. I have used a circular buffer to store the images captured for the best effort service to access it and process it in slack time.

According to the Rate monotonic policy, the deadline should be equal to the time period. Therefore, the deadline for 1 hertz will be 1000 milliseconds and 100 milliseconds for 10 hertz.

The worst-case execution time for the service was determined by running the program for the entire duration and calculating the maximum execution time taken by the service.

For 1 Hertz:

Parameter	Value
Execution Time (C_i)	0.120 msec (wcet)
Time Period (T_i)	1000 msec
Deadline (D_i)	1000 msec

For 10 Hertz:

Parameter	Value
Execution Time (C_i)	0.165 msec (wcet)
Time Period (T_i)	100 msec
Deadline (D_i)	100 msec

Image Capture:

The image capturing service runs at 1 hertz frequency for the same reason as mentioned for the sequencer. According to the Rate monotonic policy, the deadline should be equal to the time period. Therefore, the deadline for 1 hertz will be 1000 milliseconds and 100 milliseconds for 10 hertz.

The worst-case execution time for the service was determined by running the program for the entire duration and calculating the maximum execution time taken by the service.

The system is run using the worst-case execution time to make sure that if the system is feasible after running with the worst-case execution time, then it will be feasible for other execution time lesser than worst case execution time. Thus, the execution time for cheddar analysis was taken to be the worst-case execution time for respective services.

For 1 Hertz:

Parameter	Value
Execution Time (C_i)	16 msec (wcet)
Time Period (T_i)	1000 msec
Deadline (D_i)	1000 msec

For 10 Hertz:

Parameter	Value
Execution Time (C_i)	16 msec (wcet)
Time Period (T_i)	100 msec
Deadline (D_i)	100 msec

Image Store:

The image store service runs at 1 hertz frequency for the same reason as mentioned for the sequencer. According to the Rate monotonic policy, the deadline should be equal to the time period. Therefore, the deadline for 1 hertz will be 1000 milliseconds and 100 milliseconds for 10 hertz.

The worst-case execution time for the service was determined by running the program for the entire duration and calculating the maximum execution time taken by the service.

The execution time for the image dump service was taken as the worst-case execution time in order to ensure feasibility in worst-case scenario.

For 1 Hertz:

Parameter	Value
Execution Time (C_i)	18 msec (wcet)
Time Period (T_i)	1000 msec
Deadline (D_i)	1000 msec

For 10 Hertz:

Parameter	Value
Execution Time (C_i)	32 msec (wcet)
Time Period (T_i)	100 msec
Deadline (D_i)	100 msec

Image Dump (Best Effort Service):

The image dump service runs at 1 hertz frequency for the same reason as mentioned for the sequencer. According to the Rate monotonic policy, the deadline should be equal to the time period. Therefore, the deadline for 1 hertz will be 1000 milliseconds and 100 milliseconds for 10 hertz.

The worst-case execution time for the service was determined by running the program for the entire duration and calculating the maximum execution time taken by the service.

The execution time for the image dump service was taken as the worst-case execution time in order to ensure feasibility in worst-case scenario.

For 1 Hertz:

Parameter	Value
Execution Time (C_i)	240 msec (wcet)
Time Period (T_i)	1000 msec
Deadline (D_i)	1000 msec

For 10 Hertz:

Parameter	Value
Execution Time (C_i)	3656 msec (wcet)
Time Period (T_i)	100 msec
Deadline (D_i)	100 msec

CHEDDAR ANALYSIS

Cheddar analysis for 1 hertz:

For cheddar analysis, it was observed that if we keep the time periods and deadlines same for all the services, then the cheddar tool does not give the expected timing diagram. Thus, in order to get deterministic timing diagram from the cheddar tool, the time periods of sequencer and image capture service altered to fulfil the rate monotonic policy (highest priority for highest frequency service). By altering this, the cheddar tool seems to give correct response.

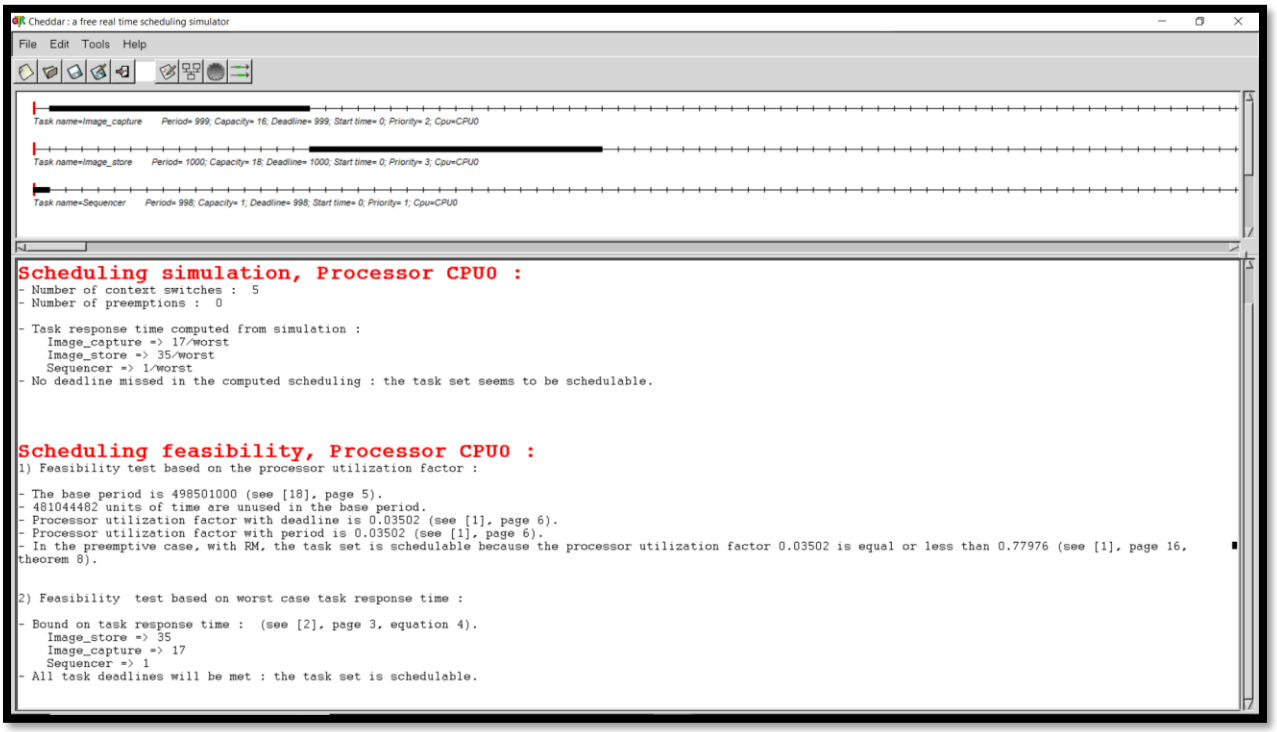

Fig 9: Cheddar Analysis for one hertz

Scheduling Point and Completion Test analysis [for 1 Hertz]:

On checking for the scheduling point and completion test, it was observed that for one hertz, the services pass both the tests. Therefore, it is guaranteed to never miss a deadline.

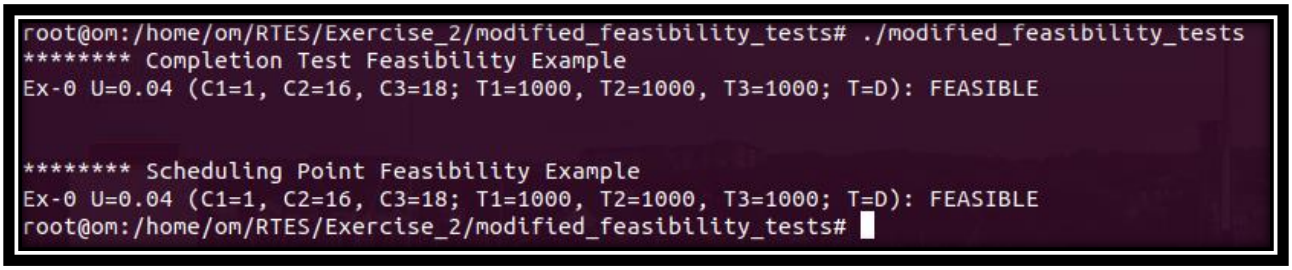

Fig 10: Scheduling point and Completion time test

Verification of Service Feasibility [1 Hertz]:

The timing diagram has been scaled for convenience. The scale may not be accurate to demonstrate the feasibility, but it is good enough to check for one. We can see that only 3.41% CPU utilization takes place. According to the RMA, for three services, the CPU utilization comes out to be 77.9%. Thus, as the CPU utilization in our case is 3.41%, we have a safety margin of approximately 96%. The CPU utilization calculated for this case was using the worst-case execution time for each service. Thus, this service set will be feasible and safe for cases where C_i varies. It was observed that the value of execution time varied between 15 msec to 15.9 msec when tested for 1800 frames.

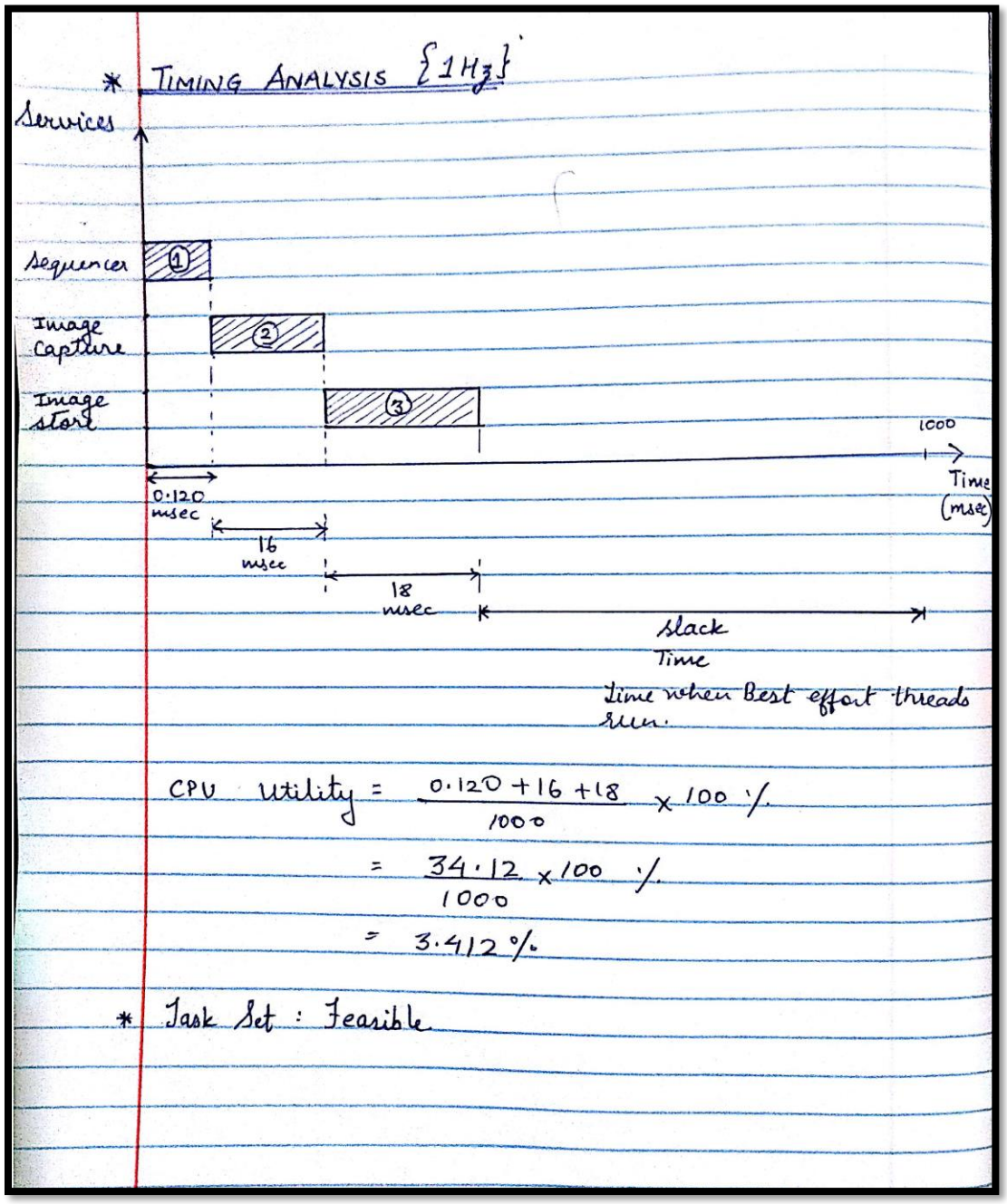

Fig 11: Hand drawn Timing diagram for one hertz

Cheddar analysis for 10 hertz:

For cheddar analysis, it was observed that if we keep the time periods and deadlines same for all the services, then the cheddar tool does not give the expected timing diagram. Thus, in order to get deterministic timing diagram from the cheddar tool, the time periods of sequencer and image capture service altered to fulfil the rate monotonic policy (highest priority for highest frequency service). By altering this, the cheddar tool seems to give correct response.

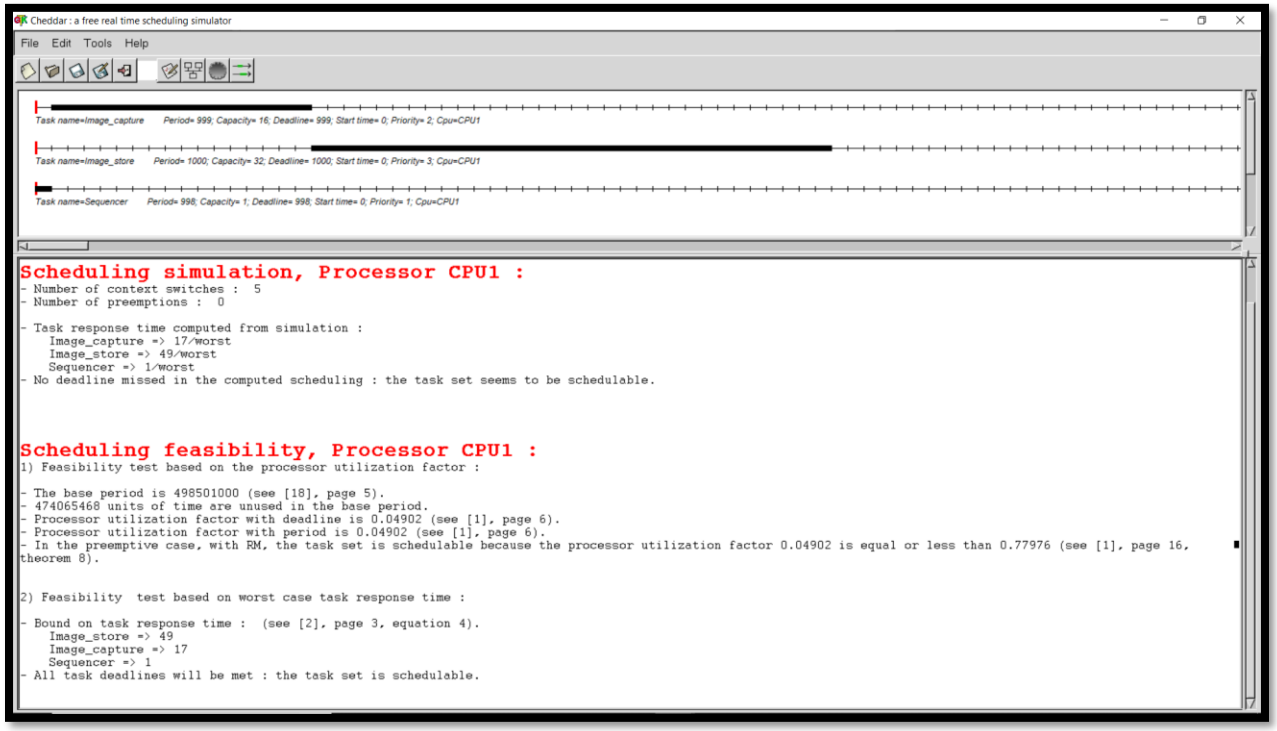

Fig 12: Cheddar Analysis for ten hertz

Scheduling Point and Completion Test analysis [for 10 Hertz]:

On checking for the scheduling point and completion test, it was observed that for ten hertz, the services pass both the tests. Therefore, it is guaranteed to never miss a deadline.

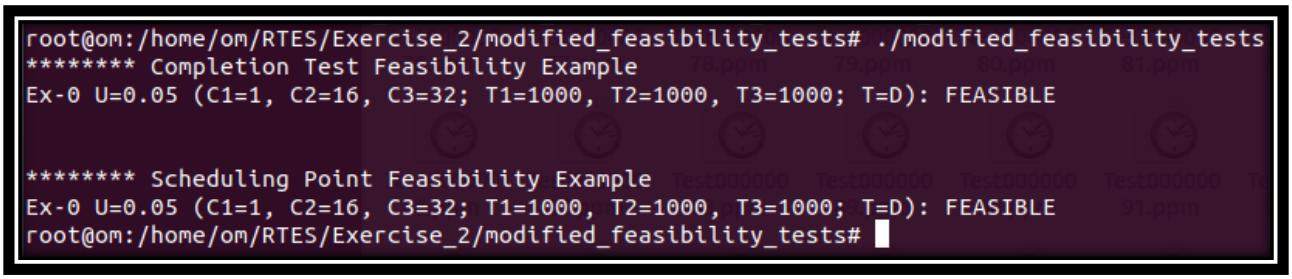

Fig 13: Scheduling point and Completion time test

Verification of Service Feasibility [10 Hertz]:

The timing diagram has been scaled for convenience. The scale may not be accurate to demonstrate the feasibility, but it is good enough to check for one. We can see that only 4.81% CPU utilization takes place. According to the RMA, for three services, the CPU utilization comes out to be 77.9%. Thus, as the CPU utilization in our case is 4.81%, we have a safety margin of 95%. The CPU utilization calculated for this case was using the worst-case execution time for each service. Thus, this service set will be feasible and safe for cases where C_i varies. It was observed that the value of execution time varied between 15-16 msec over 6000 frames.

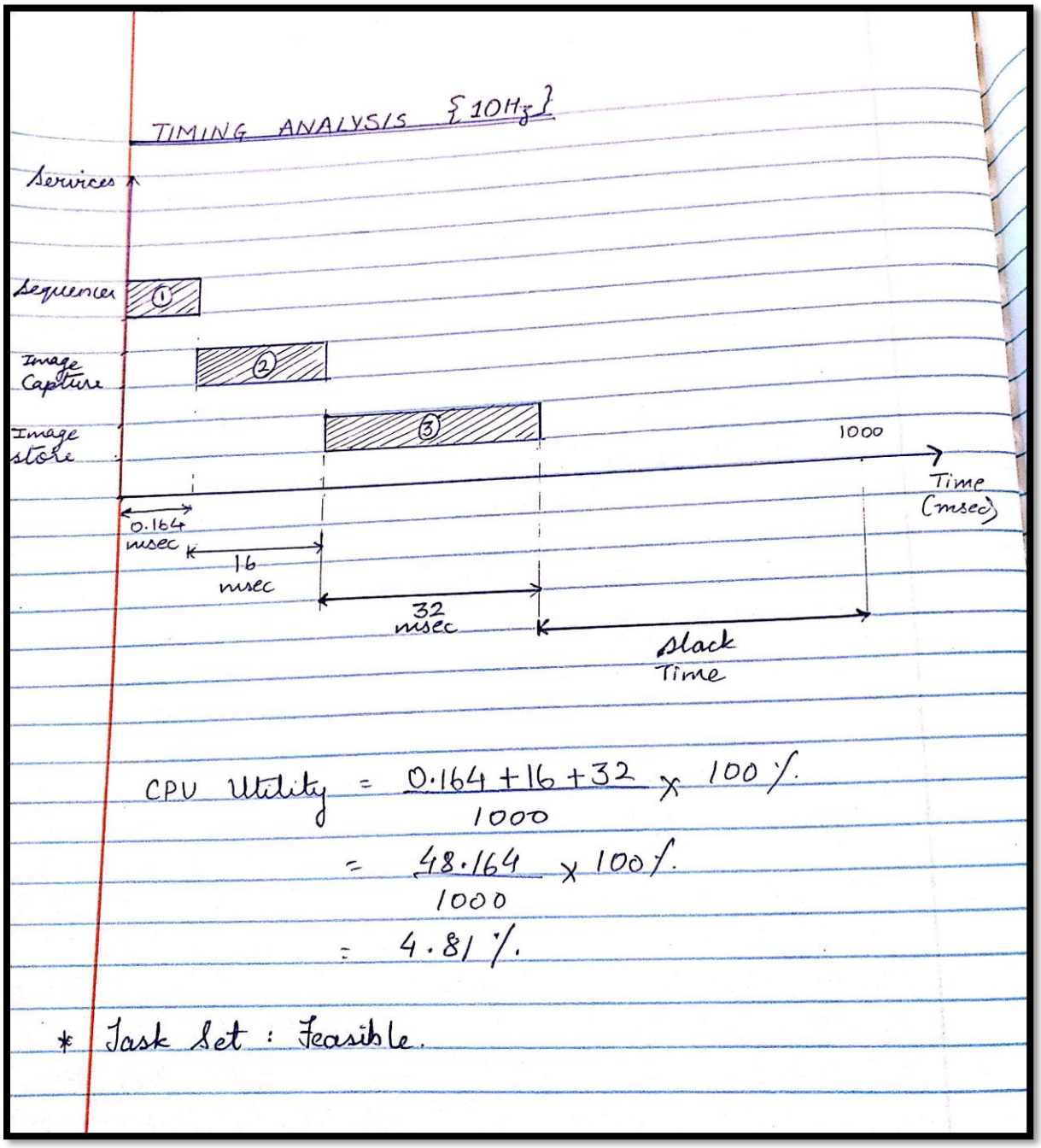

Fig 14: Hand drawn Timing diagram for one hertz

ANALYSIS OF SYSTEM

PROOF OF CONCEPTS AND TESTS COMPLETED

[NOTE: Currently, complete analysis has been completed for 1hertz only. Analysis for 10 hertz will be updated after completion]

ANALYSIS OF SYSTEM AT 1 HERTZ

To test if the system consisting of three services meet the deadline, the syslog file was checked for accurate timestamps. Timestamps on the image header can also be checked to ensure accurate and precise timings. From the below image, it can be seen that all services start and end before the other in queue starts. The print statements in the syslog can be out of order but that does not mean that they are executed out of order. It can be verified with the help of timestamps that the task in execution runs to completion and only then the other service starts.

```
Aug  9 21:58:05 om time_lapse: SEQ Count: 2#011 Sequencer start Time: 1565409485.550565 seconds
Aug  9 21:58:05 om time_lapse: SEQ Count: 2#011 Sequencer end Time: 1565409485.550628 seconds
Aug  9 21:58:05 om time_lapse: IMAGE CAPTURE START
Aug  9 21:58:05 om time_lapse: S1 Count: 2#011 Image Capture start Time: 1565409485.550662 seconds
Aug  9 21:58:05 om time_lapse: S1 Count: 2#011 Image Capture End Time: 1565409485.565387 seconds
Aug  9 21:58:05 om time_lapse: IMAGE CAPTURE END
Aug  9 21:58:05 om time_lapse: IMAGE DUMP START
Aug  9 21:58:05 om time_lapse: S2 Count: 2#011 Image Dump start Time: 1565409485.565444 seconds
Aug  9 21:58:05 om time_lapse: S2 Count: 2#011 Image Dump End Time: 1565409485.567782 seconds
Aug  9 21:58:05 om time_lapse: IMAGE DUMP END
Aug  9 21:58:05 om time_lapse: SOCKET SEND START
Aug  9 21:58:05 om time_lapse: S3 Count: 2#011 S3 Start Time: 1565409485.567867 seconds
Aug  9 21:58:05 om time_lapse: S3 Count: 2#011 S3 End Time: 1565409485.567892 seconds
Aug  9 21:58:05 om time_lapse: SOCKET SEND END
```

```
Aug  9 21:58:06 om time_lapse: SEQ Count: 3#011 Sequencer start Time: 1565409486.550563 seconds
Aug  9 21:58:06 om time_lapse: SEQ Count: 3#011 Sequencer end Time: 1565409486.550623 seconds
Aug  9 21:58:06 om time_lapse: IMAGE CAPTURE START
Aug  9 21:58:06 om time_lapse: S1 Count: 3#011 Image Capture start Time: 1565409486.550658 seconds
Aug  9 21:58:06 om time_lapse: IMAGE DUMP START
Aug  9 21:58:06 om time_lapse: S1 Count: 3#011 Image Capture End Time: 1565409486.565391 seconds
Aug  9 21:58:06 om time_lapse: IMAGE CAPTURE END
Aug  9 21:58:06 om time_lapse: S2 Count: 3#011 Image Dump start Time: 1565409486.565459 seconds
Aug  9 21:58:06 om time_lapse: S2 Count: 3#011 Image Dump End Time: 1565409486.567786 seconds
Aug  9 21:58:06 om time_lapse: IMAGE DUMP END
Aug  9 21:58:06 om time_lapse: SOCKET SEND START
Aug  9 21:58:06 om time_lapse: S3 Count: 3#011 S3 Start Time: 1565409486.567873 seconds
Aug  9 21:58:06 om time_lapse: S3 Count: 3#011 S3 End Time: 1565409486.567887 seconds
```

```
Aug  9 21:58:07 om time_lapse: SEQ Count: 4#011 Sequencer start Time: 1565409487.550549 seconds
Aug  9 21:58:07 om time_lapse: SEQ Count: 4#011 Sequencer end Time: 1565409487.550609 seconds
Aug  9 21:58:07 om time_lapse: IMAGE CAPTURE START
Aug  9 21:58:07 om time_lapse: S1 Count: 4#011 Image Capture start Time: 1565409487.550644 seconds
Aug  9 21:58:07 om time_lapse: IMAGE DUMP START
Aug  9 21:58:07 om time_lapse: S1 Count: 4#011 Image Capture End Time: 1565409487.565417 seconds
Aug  9 21:58:07 om time_lapse: IMAGE CAPTURE END
Aug  9 21:58:07 om time_lapse: S2 Count: 4#011 Image Dump start Time: 1565409487.565484 seconds
Aug  9 21:58:07 om time_lapse: S2 Count: 4#011 Image Dump End Time: 1565409487.567888 seconds
Aug  9 21:58:07 om time_lapse: IMAGE DUMP END
Aug  9 21:58:07 om time_lapse: SOCKET SEND START
Aug  9 21:58:07 om time_lapse: S3 Count: 4#011 S3 Start Time: 1565409487.567975 seconds
Aug  9 21:58:07 om time_lapse: S3 Count: 4#011 S3 End Time: 1565409487.567989 seconds
Aug  9 21:58:07 om time_lapse: SOCKET SEND END
```

```
Aug  9 21:58:08 om time_lapse: SEQ Count: 5#011 Sequencer start Time: 1565409488.550555 seconds
Aug  9 21:58:08 om time_lapse: SEQ Count: 5#011 Sequencer end Time: 1565409488.550619 seconds
Aug  9 21:58:08 om time_lapse: IMAGE CAPTURE START
Aug  9 21:58:08 om time_lapse: S1 Count: 5#011 Image Capture start Time: 1565409488.550653 seconds
Aug  9 21:58:08 om time_lapse: IMAGE DUMP START
Aug  9 21:58:08 om time_lapse: S1 Count: 5#011 Image Capture End Time: 1565409488.565384 seconds
Aug  9 21:58:08 om time_lapse: IMAGE CAPTURE END
Aug  9 21:58:08 om time_lapse: S2 Count: 5#011 Image Dump start Time: 1565409488.565456 seconds
Aug  9 21:58:08 om time_lapse: S2 Count: 5#011 Image Dump End Time: 1565409488.567766 seconds
Aug  9 21:58:08 om time_lapse: IMAGE DUMP END
Aug  9 21:58:08 om time_lapse: SOCKET SEND START
Aug  9 21:58:08 om time_lapse: S3 Count: 5#011 S3 Start Time: 1565409488.567852 seconds
Aug  9 21:58:08 om time_lapse: S3 Count: 5#011 S3 End Time: 1565409488.567867 seconds
Aug  9 21:58:08 om time_lapse: SOCKET SEND END
```

Fig 12: Proof that system meets timing requirements

It can be clearly seen that each service is triggered exactly after one second for 1 hertz. This is accurate up to 100 milliseconds for all frames numbered from 1 to 1800.

From the csv files made for each service, we can see that the requests for each service were pretty accurate up to nanoseconds range. It can be observed from the csv files (attached with raw images and video folder uploaded on canvas) that the execution time for all the services was observed to be very stable. Higher stability resulted in less amount of jitter. Lesser the jitter, more predictable the response of the services.

ANALYSIS OF SYSTEM AT 10 HERTZ

The system for 10 hertz was verified similar to the way it was tested for 1 hertz. It was observed that the images were captured at exactly a 100 millisecond. However, it was observed that in case of 10 hertz, there was a little jitter involved which resulted in the system having a few blurry frames. However, timing was achieved sufficiently to observe the system working at 10 hertz with a little amount of jitter.

```
Aug 17 15:54:35 om time_lapse: SEQ Count: 6012#011 Sequencer start Time: 1566078875.507717 seconds
Aug 17 15:54:35 om time_lapse: SEQ Count: 6012#011 Sequencer end Time: 1566078875.507789 seconds
Aug 17 15:54:35 om time_lapse: [IMAGE CAPTURE]: IMAGE CAPTURE START
Aug 17 15:54:35 om time_lapse: S1 Count: 6012#011 Image Capture start Time: 1566078875.507836 seconds
Aug 17 15:54:35 om time_lapse: ********************* Mainloop start **********************************
Aug 17 15:54:35 om time_lapse: ***************************** select start ***********************************
Aug 17 15:54:35 om time_lapse: ***************************** select end *************************************
Aug 17 15:54:35 om time_lapse: ********read frame start **************
Aug 17 15:54:35 om time_lapse: ****************Process image start*********************
Aug 17 15:54:35 om time_lapse: ****************Process image end**********************
Aug 17 15:54:35 om time_lapse: Image store start
Aug 17 15:54:35 om time_lapse: **********************read frame end *************
Aug 17 15:54:35 om time_lapse: ********************* Mainloop End   **********************************
Aug 17 15:54:35 om time_lapse: S1 Count: 6012#011 Image Capture End Time: 1566078875.522882 seconds
Aug 17 15:54:35 om time_lapse: IMAGE CAPTURE END
Aug 17 15:54:35 om time_lapse: S2 Count: 6012#011 S2 Start Time: 1566078875.522920 seconds
Aug 17 15:54:35 om time_lapse: Head = 6013
Aug 17 15:54:35 om time_lapse: S2 Count: 6012#011 Image Store End Time: 1566078875.540615 seconds
Aug 17 15:54:35 om time_lapse: Image store end
Aug 17 15:54:35 om time_lapse: IMAGE DUMP START
Aug 17 15:54:35 om time_lapse: S3 Count: 6012#011 Image Dump start Time: 1566078875.540710 seconds
Aug 17 15:54:35 om time_lapse: DUMP IMAGE.....................................................................
Aug 17 15:54:35 om time_lapse: DUMPED IMAGE!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
Aug 17 15:54:35 om time_lapse: Tail = 6013
Aug 17 15:54:35 om time_lapse: S3 Count: 6012#011 Image Dump End Time: 1566078875.543101 seconds
Aug 17 15:54:35 om time_lapse: IMAGE DUMP END
```

CHALLENGES FACED

I. According to my system design, I planned to have 4 real-time services. Sequencer being the first, followed by Image capture, Image dump and Image sending via socket having priorities in the order stated. When I was trying to send the images just after they were getting created, I observed that only 1018 frames were being transferred to the server/host machine via sockets. I tried debugging this issue by logging each and every occurring in the program but was not successful in rectifying the issue. I plan to solve this issue in order to transfer all the frames without any data loss from target machine to the host machine.

II. Secondly, I am still not able to capture glitch-free frames when running at 10 hertz. I feel the reason behind glitchy frames is lack of I/O decoupling. I plan to incorporate I/O decoupling in order to make the ppm dump service non-real time, so that images are captured at every 100 msec and stored in the circular buffer. This data can then be taken by the image dump service to convert it into ppm file. This will ensure that all the images are captured accurately and after conversion to ppm file, can be transferred to the host.

CONCLUSION

By implementing the Time Lapse Image Acquisition project, I was able to understand how services should be scheduled in real-time. I understood how even a few milliseconds or microseconds jitter can cause a system to fail or behave non-deterministically. This project helped me learn important concepts like the Rate monotonic policy and its application. By analyzing the system, it was observed that the system works perfectly fine for one hertz. Analysis for 10 hertz is yet to be completed. The updated version will be submitted once all the to-do tasks and remaining analysis are completed.

REFERENCES

[1] Time Lapse Photography: https://en.wikipedia.org/wiki/Time-lapse_photography

[2] PPM file format: http://paulbourke.net/dataformats/ppm/

[3] Scheduling Point and Completion Time Test code was referenced from Professor Sam Siewert's Course website page (ECEN 5623).

[4] References taken for codes are mentioned in the code submitted.

TOOLS USED

[1] Code2Flow application: https://code2flow.com/ for making flowcharts.

[2] Draw IO application: https://www.draw.io/ for making software/hardware diagrams.

[3] Cheddar for timing analysis.

GITHUB LINK

- https://github.com/omraheja/Real-Time-Embedded-Systems-ECEN-5623