Sundar Krishnakumar

Final Project Report – Synchronome Time Lapse

Board used: Jetson nano

Other hardware: UVC complaint USB Camera Logitech C270

Scheduling Policy: Rate Monotonic Scheduling Policy

ECEN 5623 - Real-Time Embedded Systems (Dr. Sam Siewert)

Summer -2020

TABLE OF CONTENTS

INTRODUCTION

The project is about synchronome and time lapse of the captured images.

Synchronome: A secondary clock that tries to synchronize with a high precision primary clock.

Timelapse: In a slow motion capture large no. of frames should be captured per second. And when they are played at the normal 24fps it looks like a slow motion video to our eyes. But in Timelapse, less frames (say 1 frame per second) are captured and then the resultant images are converted to a video format with playback at 24fps. Timelapse videos are common techniques used in nature photography.

The project related these two topics in its model.

FUNCTIONAL CAPABILITY REQUIREMENTS:

General requirement: To be synchronous with a physical clock and to capture the movement of the second's hand at each distinct positions. This is for 1Hz target capture. For a 5Hz/10Hz capture a digital stopwatch that has $1/10^{th}$ of a second measurement is used and each change of its is to be captured.

At the functional level, the system does the following:

1. The system should sample the the physical analog clock (or digital clock) at very high frequency via a webcam (here C270).A high frequency capture is needed so that there are enough frames at which the clock hand or the 1/th of a second digit is stationary and not moving. Note: The C270 supports capture rate up to 30fps and an average capture rate of around 29fps under **very good lighting conditions.**

2. Next the system should analyze the captured samples and pick a good sample that should not be a repeat or a blurry one.

3. Once the sample has been selected the system must store the frame on the hard disk.

4. Finally the system sends the saved frame over a local network to another destination machine.

Expanding the requirements into **major requirements based on real-time design principles (DP) and project requirements (PR):**

1. (DP)The frame capture must occur at monotonic request times/periods and must complete before deadline. In Rate Monotonic Policy **request time = deadline**. So the real time task that captures the frames must complete before the next request arrives. In short, this task has to be real time service
2. (PR) The capture should follow Nyquist criterion of oversampling that satisfies the target capture rate.

3. (PR) The resolution of each frame should be at least VGA (640 x 480).

4. (PR)The analysis part checks the collected previous and the current image in multiple iterations (if required) and selects a good sample (no skips, repeats or blurs).

5. (PR)The real time services must be at least two real time tasks that have affinity to a single core.

6. (PR)The samples must be saved in .ppm format. So the PPM header must be generated and should prefix the pixel information in the .ppm file. Should also incorporate "uname -a" and timestamp details as comments in the PPM header.

7. (PR)The accumulated latency in the timestamp must be less than 1 second. i.e. for a capture of 1801 frames (0 to 1800) the timestamp error must be less than zero.

8. (DP)For saving the sample on the disk file I/O is required. This step blocks instruction execution and it is a bad design to integrate file I/O in a real time service. So should not belong to the core where RT tasks are assigned. A suitable I/O decoupling mechanism should be employed.

9. (PR)A total of 1801 saved frames are needed for verification. (i.e. 30minutes of frame information for 1Hz and 3 minutes for 10Hz).

The current real time Software system for the Project “Synchronome” follows a common design principle for different target frequency captures. With additional parameter configurations the different target frequencies (**1Hz, 5Hz and 10Hz**) can be achieved. The system combines real time functions (called real time tasks/services) on one core and non-real time functions on another core.

FUNCTIONAL DESIGN OVERVIEW AND DIAGRAMS

Board used: Jetson Nano

Camera: UVC-compliant USB camera Logitech C270

Capture Pixels : 640 X 480 (YUV422)

The camera is capable of capturing frames at 30fps under good lighting conditions. However there were unexpected results which are discussed at the result analysis section of this report .The camera is a UVC compliant camera. The code accesses it using the uvcvideo driver.

The driver provides **memory mapped buffer** based on configuration. This allows the programmer to manage multiple buffers, queue them and dequeue them. This technique speeds up the capture process and allows very high capture rates (e.g. 25Hz is achievable)

High Level Description:

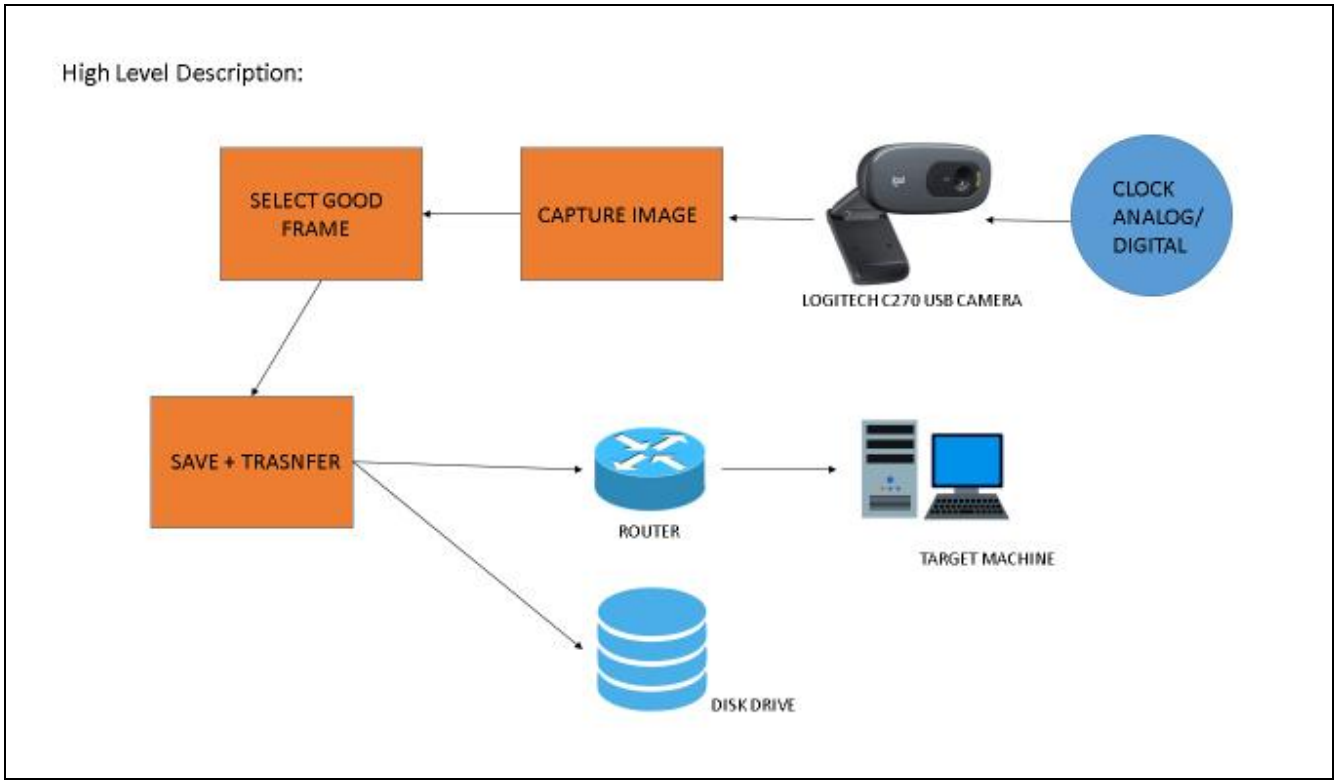

Figure2.1: High level overview

The camera in the above diagram is facing a physical analog or digital clock to capture its motion. A good number of frames are captured by the camera which results in closely spaced frames in the time domain. As a result the selection algorithm quickly finds a good frame with ample number of samples.

If sample availability is less : Based on a difference threshold value last known good sample is chosen and saved on the disk.

Why there is a need for frame selection? The transition of the seconds hand or the $1/10^{th}$ of a second digit is not quick. It is stationary for a small time and then start transitioning into the next state. A good selection engine compares and selects those stationary frame information.

The target machine will be another laptop that runs Linux on a VirtualBox.

Architecture diagram:

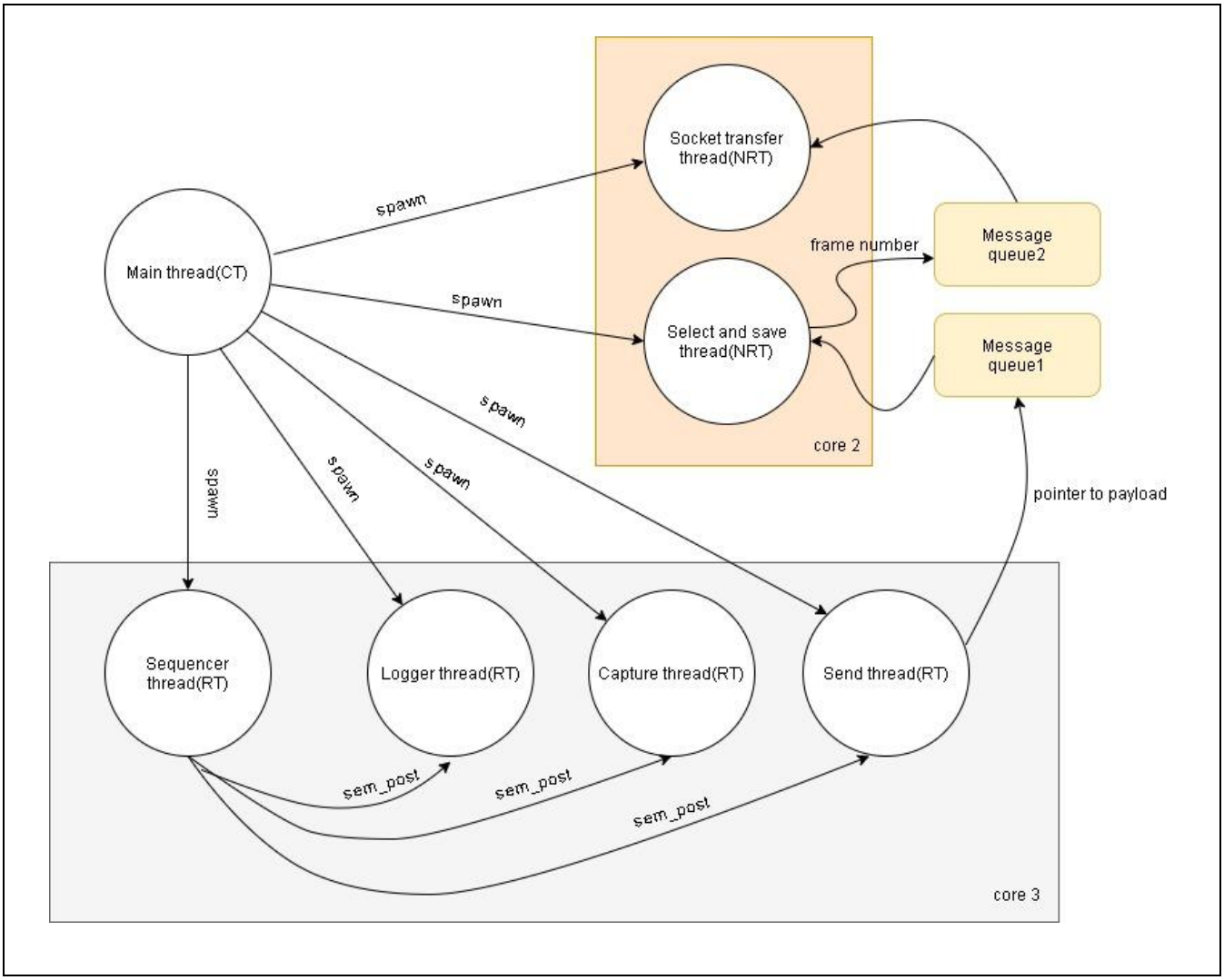

Figure2.2: Architecture block diagram

The systems involves both Real time (RT) and non-real time (NRT) tasks working on separate cores. The I/O part which saves images to the disk is decoupled from the Real time tasks and moved to another core(Core 2). While the RT tasks have their affinity set to Core 3.

The system architecture (as shown in Figure2.2) involves a main thread which acts as container thread(CT) that spawns all other tasks. The model is similar to high frequency cyclic executive. It has a high frequency sequencer that unblocks other services at the required request times by releasing a semaphore key. See sequence diagram(Figure 2.3) for the entire flow.

The **real time services** are as follows:

Frame capture task: It captures the raw YUV frames and stores in a struct type array. The captures done during the first 1sec are ignored. Once capture is done, it processes the raw image by converting **to both grayscale and RGB pixel format** in the case of 1Hz target capture and only to **grayscale** format for 5Hz and 10Hz target capture frequencies.

Timestamp (to be added in the ppm/pgm header) is taken in this task.

Grayscale: Target verification image: .pgm

RGB Colour: Target verification image: .ppm

Send frame task: It takes the processed frame and consolidates all the header information included the commented timestamp, uname details and puts it in the struct type array. Now the location at the array will have both the frame pixel information and header information. The content's address location is now sent to select and save BE service via a POSIX message queue(message queue1).

Logger task: It has the lowest priority of all real time tasks. It is used for collecting timestamp data, finding time elapsed information, jitter analysis and saving them to a .csv file in comma separated format. It is turned on only when needed.

Note: For 1Hz target frame rate, 25Hz capture service and 25Hz send service is feasible(based on Nyquist theorem) and gives enough samples to compare and select.

But for 10Hz target frame rate, 25Hz capture service and 25Hz send service, is theoretically feasible(based on Nyquist theorem), but gives only 2 samples for every $1/10^{th}$ second frame (DIGITAL CLOCK).So capture , send service should work greater than 25Hz for good number of samples to compare and select.

Also will observe and capture a physical phenomenon like melting of a cube of ice alongside the clock.

The **non- real time Best effort services** are as follows:

Select and save service: It unblocks when the message queue 1 is non-empty, does a difference and binary threshold on the previous and the current frame in the queue. Discussed more later in this section

Socket transfer service: Waits by listening to message queue2.On message arrival from select and save service, it transfers the selected and saved frames in an ordered manner to a target machine (here laptop running Linux on VirtualBox).This task is an extra feature. It can be turned ON/OFF.

Sequence diagram:

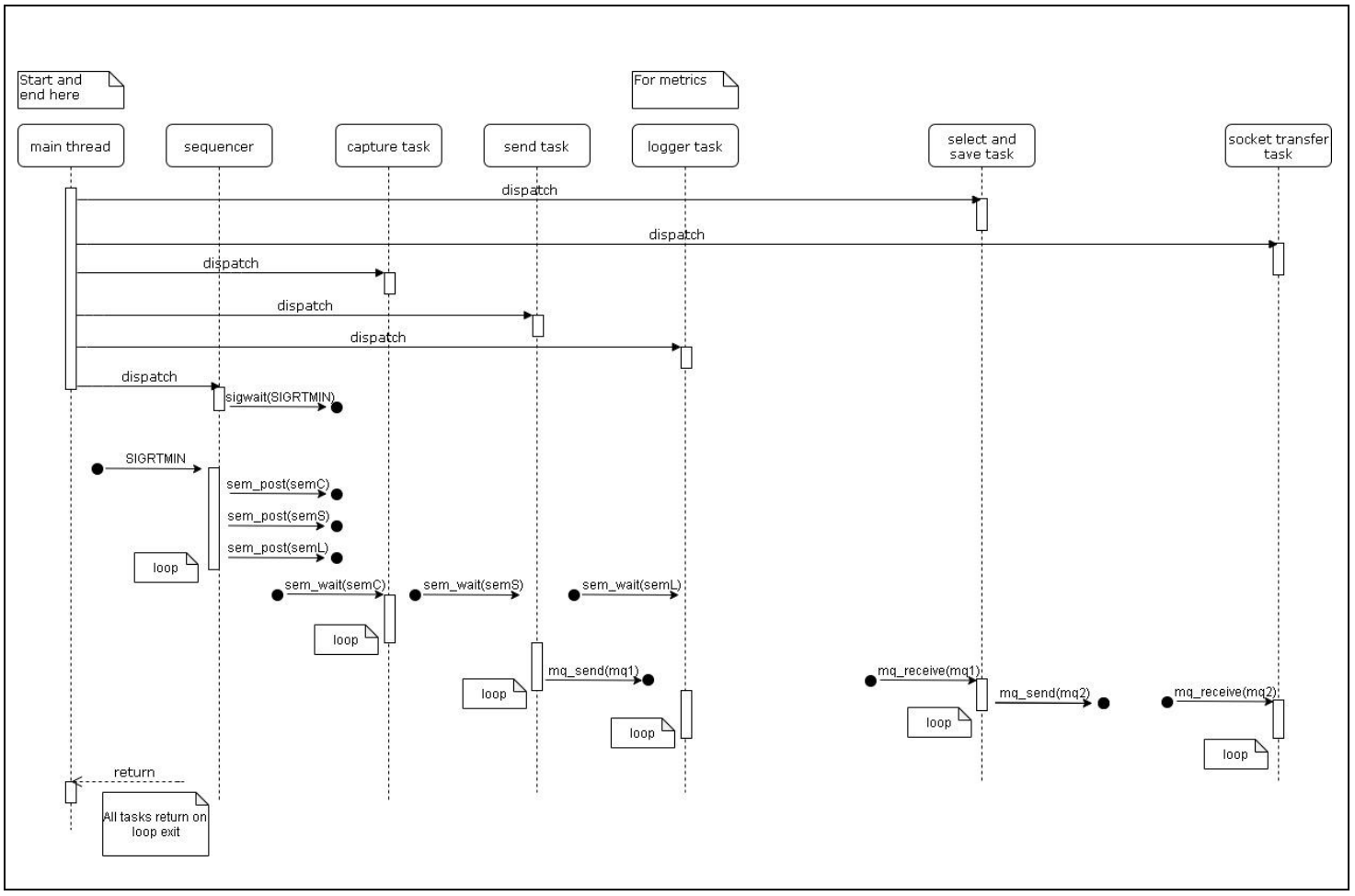

Figure2.3: Multithreaded sequence diagram model

Flowchart:

Sequencer:

The terms used as part of the sequencer: SEQ_FREQ, CAP_FREQ_MOD are discussed part of real time service requirements. FRAME_COUNT is the required number of .ppm or .pgm frames that should be saved before program completion

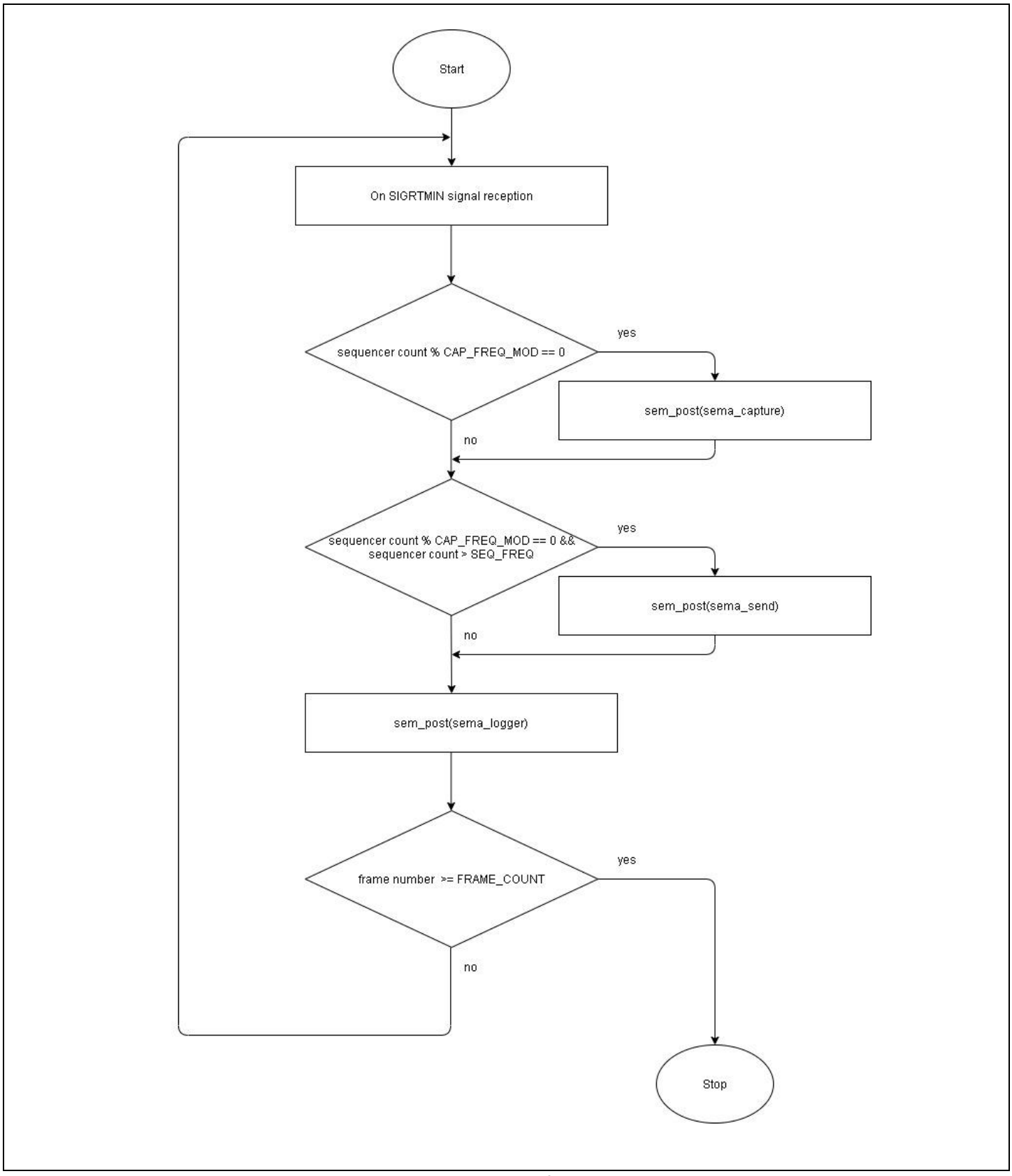

Figure2.4: sequencer flow chart

Note: Only Sequencer flow is discussed here. Other tasks have straightforward and simple execution flow as shown in the sequence diagram (See Figure2.3). But the YUV422 to RGB conversion algorithm used inside "capture task", the image selection algorithm as part of the "select and save non-real time task" and extra feature – "Frame transfer over socket" used in socket transfer task are discussed below:

YUV422 to RGB conversion:

The Logitech C270 web camera supports YUYV422 raw frame capture. This should be subjected to RGB conversion before saving as a .ppm file

The order in the byte stream for YUV422 is Y0 U0 Y1 V0.

Y – Luminance U, V – Chrominance

So every 4 byte of information contains two pixel data and on conversion to RGB pixel space will give two RGB pixel data (i.e. RGBRGB) or two RGB24 type pixel information where each channel(R,G,B) takes 1byte to store values.

Following is method to convert YUV pixel space to RGB pixel space :

```
// Code source :
http://ecee.colorado.edu/~ecen5623/ecen/ex/Linux/computer-
vision/simple-capture/capture.c

void yuv2rgb(int y, int u, int v, unsigned char *r, unsigned char *g,
unsigned char *b)
{
      int r1, g1, b1;

      // replaces floating point coefficients
      int c = y-16, d = u - 128, e = v - 128;

      // Conversion that avoids floating point
      r1 = (298 * c           + 409 * e + 128) >> 8;
      g1 = (298 * c - 100 * d - 208 * e + 128) >> 8;
      b1 = (298 * c + 516 * d           + 128) >> 8;

      // Computed values may need clipping.
      if (r1 > 255) r1 = 255;
      if (g1 > 255) g1 = 255;
      if (b1 > 255) b1 = 255;

      if (r1 < 0) r1 = 0;
      if (g1 < 0) g1 = 0;
      if (b1 < 0) b1 = 0;

      *r = r1 ;
      *g = g1 ;
      *b = b1 ;
}
```

Note: Extracting the Y – Luminance values alone will give a grayscale image.

Image selection model in select and save NRT task:

The “real time send task” populates a ring buffer that stores a struct type data structure called “payload” which has the colour image pixel information, grayscale image pixel information and relevant header information (i.e. PPM header for RGB colour image or PGM header in the case of grayscale image).

The “select and save task” receives this information via a message queue. The task applies image difference and threshold algorithm on the previous and the current grayscale image i.e. the resultant difference image is subjected to “Binary thresholding” based black and white conversion scheme.

Now it calculates the sum of difference pixels in the binary image (i.e number of white pixels in the black background).And next the difference percentage is calculated based on the following formula:

Percentage difference = Sum of difference / maximum possible difference

Maximum possible difference is: 640 x 480 (here)

Based on a selected value of percentage difference threshold value, the good frame is selected.

E.g For a a target 1Hz capture, the 20 images captured in 1 second are analyzed as mentioned above and a good frame is selected. This procedure helps avoid frame blur, frame skip and repeats.

Code reference: http://ecee.colorado.edu/~ecen5623/ecen/ex/Linux/computer-vision/diff-interactive/capture.cpp

Also, the same task saves the good frame in either colour .ppm format or grayscale .pgm format for verification purposes.

Figure2.5a: Gray image 1 (previous image)

Figure2.5b: Gray image 2 (current image)

Figure2.5c: Binary threshold result on the difference of gray image 1 and 2

An **ideal good frame** will show a complete black image when the difference image is subjected to thresholding.

Frame transfer over socket:

The host machine (Jetson Nano Board) acts as the client which sends the images. Another target machine running Linux on a virtual box(laptop) acts as the server that receives and saves the image on the disk, as the images are transferred.

AF_INET internet domain sockets are used and the packets are sent reliably based on TCP protocol. Below Figure shows the connectivity flow between the server and the client.

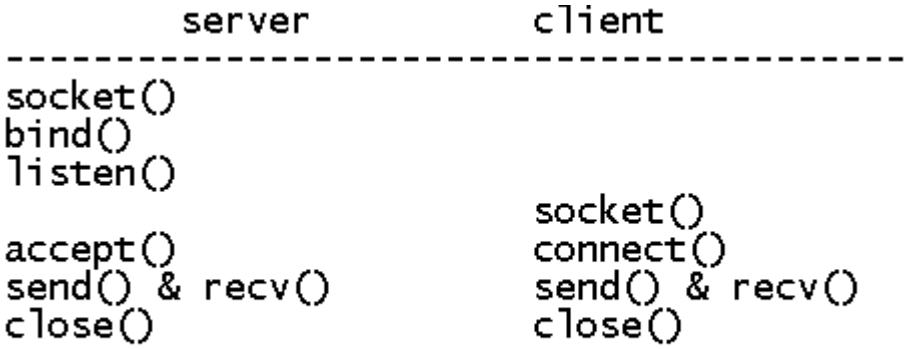

Figure2.5c: Client connection to the socket - flow

Code logic: After client makes a successful connect() call both parties can bidirectionally send and receive data.

Note: SOCK_STREAM type is used in the socket() call.SOCK_STREAM provides reliable TCP two way communication.

First the client sends the size of file before actual file transfer happens. Next the server acknowledges this transfer by sending a dummy data to the client. Next the client sends the image. Server receives the image over the network and saves it in the relevant format (.ppm for colour RGB images and .pgm for gray images).Since this is a high speed capture and transfer over a socket, we must define our own way of telling that an image transfer is over. The acknowledgement from the server side solves this purpose.

REAL-TIME REQUIREMENTS

The real time service requirements are based on Rate Monotonic Scheduling Algorithm. So task periods are same as deadline. The code implementation follows RT POSIX design methods by setting priorities to tasks and also uses SCHED_FIFO real time scheduling algorithm to implement the RM policy.The real time requirements apply for tasks (are mentioned as RT tasks in the report context) :

1. Sequencer service
2. Capture image task
3. Send image task

As mentioned earlier, the system follows a common design for all target frequencies (1Hz, 5Hz, 10Hz).Only a few additional configurable parameters are needed to be changed for each type. Those parameters only related to service requirements :

SEQ_FREQ – Frequency of the sequencer

CAP_FREQ_MOD – Frequency of the capture image and send image tasks. i.e. no. of frames to be captured and sent to “select and save image service” per second.

FPS – Target capture frequency

FPS defines the set count increment. There is a set number variable part of the struct type payload data structure which is assigned the value of another variable called “set count”. This increments for every (SEQ_FREQ / (CAP_FREQ_MOD * FPS))th iteration of the capture count(Number of captures received from the V4L2 camera buffer)

By this method of labelling the images in the payload, the select and save image service can analyze the images of a single set number and pick a good frame from that.

For e.g. For a 10Hz target captures running at 30fps capture rate with 120Hz sequencer frequency, there can be possibly three images that belong to every 1/10th of second digit movement. So by setting FPS as 10, every 120 / (4 * 10) = 3 images have the same set number i.e. these captures should belong to one particular $1/10^{th}$ of a second digit.

And this is how the common design system is applied for different target frequencies.

All units: In milliseconds

1Hz:

Sequencer service:

S0: Zero indicates the highest priority. Lowest index has the highest priority for real time tasks (RT Tasks)

T0: Inverse of SEQ_FREQ. Here SEQ_FREQ = 100

D0: Same as T0

C0: WCET analysis

Capture image service:

S1: Zero indicates the highest priority. Lowest index has the highest priority for real time tasks (RT Tasks)

T1: SEQ_FREQ divided by CAP_FREQ_MOD tells us what capture frequency the system is currently set for. Here CAP_FREQ_MOD = 4. So (100/4) gives 25 frames per second. Inverse of this is the period.

D1: Same as T0

C1: WCET analysis

Send image service:

S2: Zero indicates the highest priority. Lowest index has the highest priority for real time tasks (RT Tasks)

T2: SEQ_FREQ divided by CAP_FREQ_MOD tells us what capture frequency the system is currently set for. Here CAP_FREQ_MOD = 4. So (100/4) gives 25 frames per second. Inverse of this is the period.

D2: Same as T0

C2: WCET analysis

5Hz:

Sequencer service:

S0: Zero indicates the highest priority. Lowest index has the highest priority for real time tasks (RT Tasks)

T0: Inverse of SEQ_FREQ. Here SEQ_FREQ = 100

D0: Same as T0

C0: WCET analysis

Capture image service:

S1: Zero indicates the highest priority. Lowest index has the highest priority for real time tasks (RT Tasks)

T1: SEQ_FREQ divided by CAP_FREQ_MOD tells us what capture frequency the system is currently set for. Here CAP_FREQ_MOD = 4. So (100/4) gives 25 frames per second. Inverse of this is the period.

D1: Same as T0

C1: WCET analysis

Send image service:

S2: Zero indicates the highest priority. Lowest index has the highest priority for real time tasks (RT Tasks)

T2: SEQ_FREQ divided by CAP_FREQ_MOD tells us what capture frequency the system is currently set for. Here CAP_FREQ_MOD = 4. So (100/4) gives 25 frames per second. Inverse of this is the period.

D2: Same as T0

C2: WCET analysis

10Hz:

Sequencer service:

S0: Zero indicates the highest priority. Lowest index has the highest priority for real time tasks (RT Tasks)

T0: Inverse of SEQ_FREQ. Here SEQ_FREQ = 120

D0: Same as T0

C0: WCET analysis

Capture image service:

S1: Zero indicates the highest priority. Lowest index has the highest priority for real time tasks (RT Tasks)

T1: SEQ_FREQ divided by CAP_FREQ_MOD tells us what capture frequency the system is currently set for. Here CAP_FREQ_MOD = 4. So (120/4) gives 30 frames per second. Inverse of this is the period.

D1: Same as T0

C1: WCET analysis

Send image service:

S2: Zero indicates the highest priority. Lowest index has the highest priority for real time tasks (RT Tasks)

T2: SEQ_FREQ divided by CAP_FREQ_MOD tells us what capture frequency the system is currently set for. Here CAP_FREQ_MOD = 4. So (120/4) gives 30 frames per second. Inverse of this is the period.

D2: Same as T0

C2: WCET analysis

How request frequency or deadline was determined? Nyquist sampling criterion was used.

For 1Hz target frame rate, 25Hz capture service and 25Hz send service is feasible(also complies with Nyquist theorem) and gives enough samples to compare and select.

But for 10Hz target frame rate, 25Hz capture service and 25Hz send service, is theoretically feasible(based on Nyquist theorem), but gives only 2 samples for every $1/10^{th}$ second frame (DIGITAL CLOCK).So capture , send service should work greater than 25Hz for good number of samples to compare and select. Near to acceptable performance was seen only at 30fps. (More explanation in result analysis section)

WCET Analysis:

Maximum of all execution time is taken as the Worst Case Execution Time (WCET).

1Hz RGB image or target .ppm model:

	wcet_capture	wcet_send	wcet_sequencer
	0.01686s	0.000213s	0.000204s
Approx ->	16.9ms	0.21ms	0.204ms

Table3.1a: WCET analysis 1Hz

5Hz Grayscale image or target .pgm model:

	wcet_capture	wcet_send	wcet_sequencer
	0.003074s	0.00028s	0.000275s
Approx ->	3.1ms	0.28ms	0.3ms

Table3.1b: WCET analysis 5Hz

10Hz Grayscale image or target .pgm model:

	wcet_capture	wcet_send	wcet_sequencer
	0.02906s	0.000217s	0.000258s
Approx ->	29.1ms	0.22ms	0.3ms

Table3.1c: WCET analysis 10Hz

Note: wcet_capture average of 10Hz is 0.00338s or 3.38ms

FPS defines when the set count increment. There is a set number variable part of the struct type payload data structure which is assigned the value of another variable called “set count”. This increments for every (SEQ_FREQ / (CAP_FREQ_MOD * FPS))th iteration of the capture_cnt variable(Number of captures received from the V4L2 camera buffer)

By this method of labelling the images in the payload, the select and save image service can analyze the images of a single set number and pick a good frame from that.

For eg. For a 10Hz target captures running at 30fps capture rate with 120Hz sequencer frequency, there can be possibly three images that belong to every 1/10th of second digit movement. So by setting FPS as 10, every 120 / (4 * 10) = 3 images have the same set number i.e. these captures should belong to one particular $1/10^{th}$ of a second digit.

And this is how the common design system is applied for different target frequencies.

(The WCET timings for capture image task are not proportionate. Because configurable parameter setting called BUFFER_CNT – V4L2 memory mapped image buffer, is differently set for each target capture.)

1Hz:

Cheddar Analysis:

	wcet_capture	wcet_send	wcet_sequencer
	0.01686s	0.000213s	0.000204s
Approx ->	16.9ms	0.21ms	0.204ms
For Cheddar analysis ->	17ms	1ms	1ms
Deadline/Period ->	40ms	40ms	10ms

Table4.1a : task Schedule set

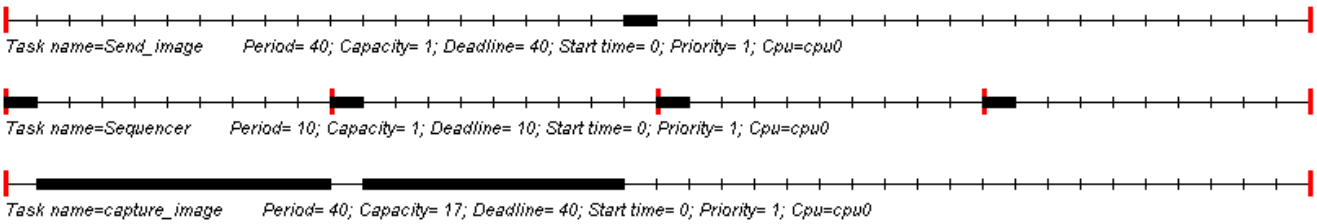

Figure4.1a: Cheddar timing diagram

```
Scheduling simulation, Processor cpu0 :
- Number of context switches :  5
- Number of preemptions :  1

- Task response time computed from simulation :
    Send_image => 20/worst
    Sequencer => 1/worst
    capture_image => 19/worst
- No deadline missed in the computed scheduling : the task set seems to be schedulable.
```

Figure4.1b: Cheddar scheduling simulation report

```
Scheduling feasibility, Processor cpu0 :
1) Feasibility test based on the processor utilization factor :

- The base period is 40 (see [18], page 5).
- 18 units of time are unused in the base period.
- Processor utilization factor with deadline is 0.55000 (see [1], page 6).
- Processor utilization factor with period is 0.55000 (see [1], page 6).
- In the preemptive case, with RM, the task set is schedulable because the processor utilization factor 0.55000 is equal or less than 1.00000 (see
[19], page 13).


2) Feasibility  test based on worst case task response time :

- Bound on task response time :  (see [2], page 3, equation 4).
    Send_image => 20
    capture_image => 19
    Sequencer => 1
- All task deadlines will be met : the task set is schedulable.
```

Figure4.1c: Cheddar scheduling feasibility report

5Hz:

Cheddar Analysis:

	wcet_capture	wcet_send	wcet_sequencer
	0.003074s	0.00028s	0.000275s
Approx ->	3.1ms	0.28ms	0.3ms
For Cheddar analysis ->	4ms	1ms	1ms
Deadline/Period ->	40ms	40ms	10ms

Table4.2: task Schedule set

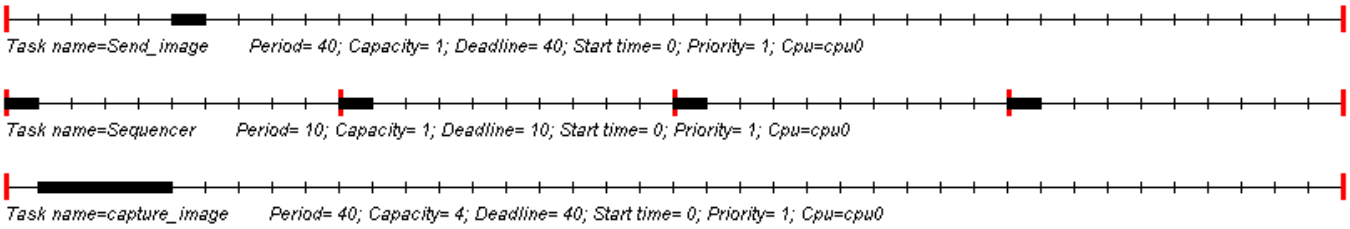

Figure4.2a: Cheddar timing diagram

```
Scheduling simulation, Processor cpu0 :
- Number of context switches :  3
- Number of preemptions :  0

- Task response time computed from simulation :
    Send_image => 6/worst
    Sequencer => 1/worst
    capture_image => 5/worst
- No deadline missed in the computed scheduling : the task set seems to be schedulable.
```

Figure4.2b: Cheddar scheduling simulation report

```
Scheduling feasibility, Processor cpu0 :
1) Feasibility test based on the processor utilization factor :

- The base period is 40 (see [18], page 5).
- 31 units of time are unused in the base period.
- Processor utilization factor with deadline is 0.22500 (see [1], page 6).
- Processor utilization factor with period is 0.22500 (see [1], page 6).
- In the preemptive case, with RM, the task set is schedulable because the processor utilization factor 0.22500 is equal or less than 1.00000 (see
[19], page 13).


2) Feasibility  test based on worst case task response time :

- Bound on task response time :  (see [2], page 3, equation 4).
    Send_image => 6
    capture_image => 5
    Sequencer => 1
- All task deadlines will be met : the task set is schedulable.
```

Figure4.2c: Cheddar scheduling feasibility report

10Hz:

Cheddar Analysis:

	wcet_capture	wcet_send	wcet_sequencer
	0.02906s	0.000217s	0.000258s
Approx ->	29.1ms	0.22ms	0.3ms
For Cheddar analysis ->	30ms	1ms	1ms
Deadline/Period ->	33.33ms	33.33ms	8.33ms
For Cheddar analysis ->	33ms	33ms	8ms

Table4.3: task Schedule set

Cheddar analysis fails for a wcet_capture time of 33ms. But wcet_capture average of 10Hz program is 0.00338s or 3.38ms. For cheddar analysis : 4ms

This will be used in Cheddar and other timing analysis. And reason for the unreasonably high WCET value will be explained in the result analysis section.

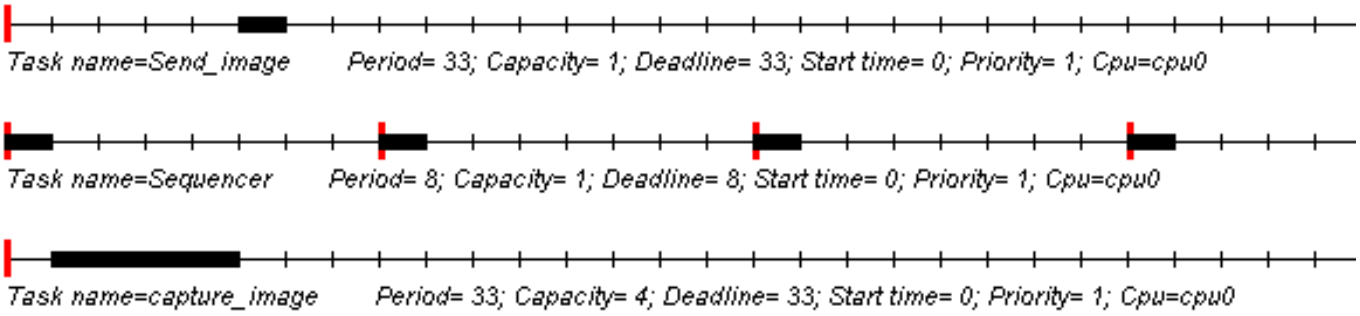

Figure4.3a: Cheddar timing diagram

```
Scheduling simulation, Processor cpu0 :
- Number of context switches :  31
- Number of preemptions :  3

- Task response time computed from simulation :
    Send_image => 6/worst
    Sequencer => 1/worst
    capture_image => 5/worst
- No deadline missed in the computed scheduling : the task set seems to be schedulable.
```

Figure4.3b: Cheddar scheduling simulation report

```
Scheduling feasibility, Processor cpu0 :
1) Feasibility test based on the processor utilization factor :

- The base period is 264 (see [18], page 5).
- 191 units of time are unused in the base period.
- Processor utilization factor with deadline is 0.27652 (see [1], page 6).
- Processor utilization factor with period is 0.27652 (see [1], page 6).
- In the preemptive case, with RM, the task set is schedulable because the processor utilization factor 0.27652 is equal or less than 0.77976
[1], page 16, theorem 8).


2) Feasibility  test based on worst case task response time :

- Bound on task response time :  (see [2], page 3, equation 4).
    Send_image => 6
    capture_image => 5
    Sequencer => 1
- All task deadlines will be met : the task set is schedulable.
```

Figure4.3c: Cheddar scheduling feasibility report

Scheduling Point and Completion Test analysis:

The test is done with the help of the code from the course website:
http://ecee.colorado.edu/~ecen5623/ecen/ex/Linux/code/Feasibility/feasibility_tests.c

The code performs scheduling point test and completion time test. These tests are exact feasibility algorithms. The scheduling point test is based upon the Lehoczky, Sha, and Ding(LSD) exact analysis

1Hz, 5Hz, 10Hz:

```
sundar@sundar-desktop:~/rtes/ex6/ns$ ./feasibility
******** Completion Test Feasibility Example
Ex-0 U=0.55 (C1=1, C2=1, C3=17; T1=10, T2=40, T3=40; T=D): FEASIBLE
Ex-1 U=0.23 (C1=1, C2=1, C3=4; T1=10, T2=40, T3=40; T=D): FEASIBLE
Ex-2 U=0.28 (C1=1, C2=1, C3=4; T1=8, T2=33, T3=33; T=D): FEASIBLE


******** Scheduling Point Feasibility Example
Ex-0 U=0.55 (C1=1, C2=1, C3=17; T1=10, T2=40, T3=40; T=D): FEASIBLE
Ex-1 U=0.23 (C1=1, C2=1, C3=4; T1=10, T2=40, T3=40; T=D): FEASIBLE
Ex-2 U=0.28 (C1=1, C2=1, C3=4; T1=8, T2=33, T3=33; T=D): FEASIBLE
```

Figure4.4: Completion time and Schedule point test – results (units milliseconds)

So all the task sets are feasible according to Figure4.4

Time-stamp tracing using syslog of other non-intrusive mechanism: Was done using a separate logger task. All the time stamps were logged in a .csv file and analyzed after program completion. The same file also contains information about jitter analysis and WCET analysis for different target frequencies.

See [git link]

Manual timing diagram test:

The LCM for 10Hz test is 264.So doing manual timing diagram for 1Hz and 5Hz alone.

1Hz:

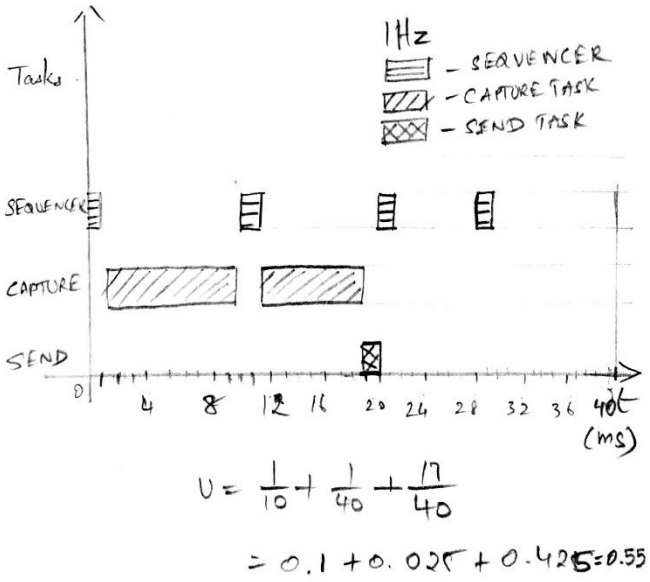

Figure4.5a: Utility is 0.55 for 1Hz

5Hz:

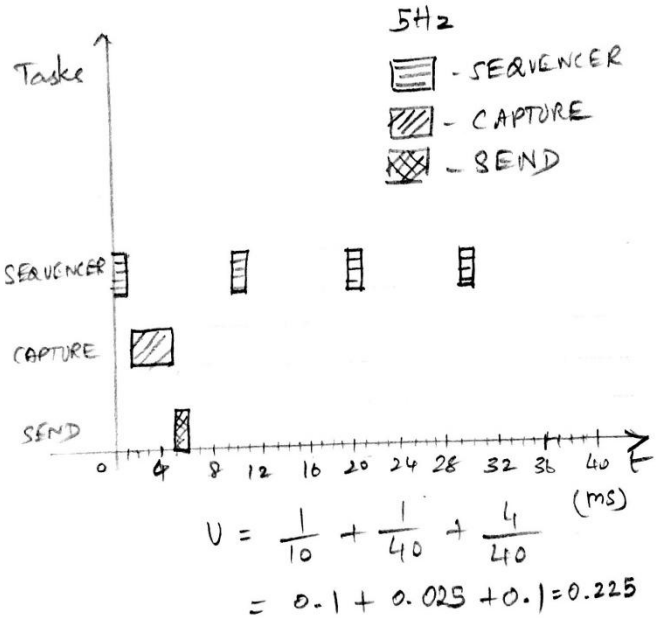

Figure4.5b: Utility is 0.225 for 5Hz

The manual hand drawn diagrams for 1Hz and 5Hz are similar to results obtained from above tests.

Regarding safety margin: Based on the number of samples required to analyze and select good frame, suitable sequencer, capture image task and send image task frequencies or service periods have been selected. In the analysis, the ceiling of the WCET values and the floor of deadline/periods are assumed. And the test results indicate a utility not more than 60% for all 3 target frequencies. So it is above the safety margin criteria of 30% (as per Liu and Layland paper).

PROOF-OF-CONCEPT WITH EXAMPLE OUTPUT AND TESTS COMPLETED

(In these tests, the extra feature – transfer of image over socket, was turned ON and program's performance was analyzed in terms of capture jitter and accumulated latency as shown below.)

1Hz:

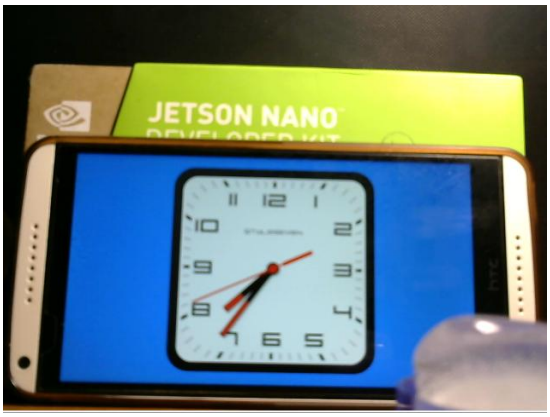

Figure5.1a: rgb_0.ppm

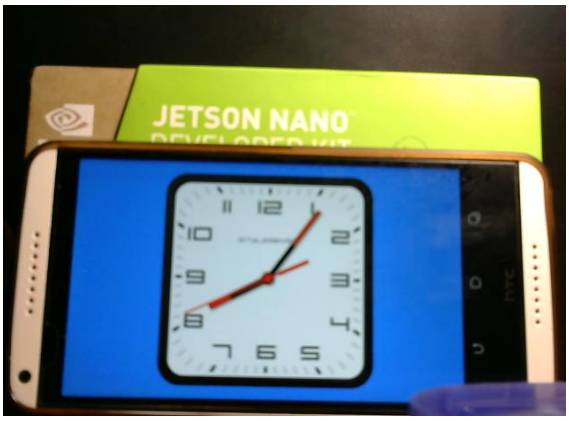

Figure5.1b: rgb_1800.ppm

Header information:

```
P6
#26608416.28.191295097
#Linux sundar-desktop 4.9.140-tegra #1 SMP PREEMPT Wed Apr 8
18:10:49 PDT 2020 aarch64
640 480
255
```

Figure5.2a: rgb_0.ppm

```
P6
#26608446.28.341328086
#Linux sundar-desktop 4.9.140-tegra #1 SMP PREEMPT Wed Apr 8
18:10:49 PDT 2020 aarch64
640 480
255
```

Figure5.2b: rgb_1800.ppm

The above images are part of 1Hz target capture **.** Figure 5.2 a-b show the PPM header information.The accumulated latency over the captures the 1801 frames (30 minutes) is **well below 1second .** Also the header contains the "uname –a" information.

5Hz:

The 5Hz capture was attempted using RGB colour image as well. But with grayscale images the program performed well. So all timing analysis done previously and the documentation are based on grayscale image capture.

Figure5.3a: gray_0.pgm

Figure5.3b: gray_1800.pgm

```
P5
#26607552.22.20802472
#Linux sundar-desktop 4.9.140-tegra #1 SMP PREEMPT Wed Apr 8
18:10:49 PDT 2020 aarch64
640 480
255
```

Figure5.4a: gray_0.pgm

```
P5
#26607558.22.20810573
#Linux sundar-desktop 4.9.140-tegra #1 SMP PREEMPT Wed Apr 8
18:10:49 PDT 2020 aarch64
640 480
255
```

Figure5.4b: gray_1800.pgm

The above images are part of 5Hz target capture **.** Figure 5.4 a-b show the PPM header information.The accumulated latency over the captures the 1801 frames (6 minutes) is **well below 1second .** Also the header contains the “uname –a” information.

10Hz:

In the real time requirements section, a note said that wcet_capture of 10Hz target frequency was abnormally high compared to the average execution time of the image capture task.

Analysis:

10Hz capture gave poor results with RGB output format (.ppm) with sequencer at 100Hz and other RT services at 25Hz (i.e. 25 frames per second capture).We can clearly see that only 2 frames are available for a unique 1/10th of a second digit information. So the selection algorithm (difference and threshold) fails to do a good work with very less samples to select from.

Maximum possible frame capture rate of 30fps is selected here (sequencer at 120Hz and other RT services at 30Hz). This gives 30 frames for one second. So again only 3 frames for every100 millisecond (i.e. those 3 frames can belong to a unique 1/10th of second digit position).In addition, the camera capture rate drops below 30 during course and increases the execution time of the capture task - it takes a longer time to receive the image from the v4l2 buffer on **dequeue ioctl call**, causing it to miss its deadline .**The capture jitter analysis plots of 10Hz frequency proves the same**. The code logic does not allow send image task to execute if there is a deadline miss. So output image generation stops around the same frame number.

So, the 10Hz target capture was successful upto 600-700 saved frames above which the system encounters deadline misses in the capture image RT task.

Figure5.5a: gray_0.pgm

Figure5.5b: gray_600.pgm

```
P5
#26608387.48.437411160
#Linux sundar-desktop 4.9.140-tegra #1 SMP PREEMPT Wed Apr 8
18:10:49 PDT 2020 aarch64
640 480
255
```

Figure5.6a: gray_0.pgm

```
P5
#26608388.48.404149050
#Linux sundar-desktop 4.9.140-tegra #1 SMP PREEMPT Wed Apr 8
18:10:49 PDT 2020 aarch64
640 480
255
```

Figure5.6b: gray_600.pgm

The above images are part of 10Hz target capture **.** Figure 5.6 a-b show the PPM header information.The accumulated latency over the captures the 601 frames (60 seconds) is **well below 1second .** Also the header contains the "uname –a" information.

Jitter Analysis:

For Jitter analysis all the real time tasks were taken into acoount.

Jitter here is relative to the deadline.

Calculation:

Jitter in the frame = execution time of service – deadline

(All units in seconds)

1Hz RGB :

Capture task jitter:

AVERAGE: -0.0242265 MIN: -0.04 MAX: -0.02314

Send task jitter:

AVERAGE: -0.039932052 MIN: -0.04 MAX: -0.039787

Sequencer jitter:

AVERAGE: -0.009926881 MIN: -0.009963 MAX: -0.009796

5Hz RGB :

Capture task jitter:

AVERAGE: -0.037304433 MIN: -0.04 MAX: -0.036926

Send task jitter:

AVERAGE: -0.039921967 MIN: -0.04 MAX: -0.03972

Sequencer jitter:

AVERAGE: -0.009921523 MIN: -0.009965 MAX: -0.009725

10Hz RGB :

Capture task jitter:

AVERAGE: -0.030008392 MIN: -0.033333 MAX: -0.004274

Send task jitter:

AVERAGE: -0.033246687 MIN: -0.033333 MAX: -0.033116

Sequencer jitter:

AVERAGE: -0.00825135 MIN: -0.008295 MAX: -0.008075

In the above green highlighted text, the average, minimum and maximum jitter are shown for the different target capture frequencies.

The jitter analysis plots are here: [git link]

So why this jitter?

This jitter is the result of inaccurate clock, caching, page faults and other pipeline level optimizations for the sake of performance. These are inbuilt hardware features of a General Purpose System. Because of this there is **a variation in the execution time** for each occurrence. Thus causing difficulty in calculating or predicting the **exact execution time** for a real time thread/service.

Note: The effect of variable CPU frequency towards jitter is not included here as the experiment is carried out at maximum Core frequency.

Commands to set maximum frequency on Jetson board: [git link]

To avoid this jitter and to get a predictable execution time , the following are few solutions:

1. Locking the cache (Tightly coupled memory)
2. Disable paging using special Linux calls.
3. Using processors cores that do not have pipelining.
4. Use of dedicated/Auxiliary Clock.

(**Note:** In real time systems, determinism is the requirement and not the throughput.)

How constant the request frequency is for your system design? The initial sequencer design was based on a nanosleep(). There were clock drifts occurring on each wake up , during the long run of the program. This caused frame skips or repeats. A more accurate clock_nanosleep() call did not solve the problem. So changed the sequencer to a timer model : The Sequencer waits on sigwait call. When it receives a specific signal it wakes up and resumes its code logic. According to Real time POSIX design methods (mentioned here: https://www.eetimes.com/posix-in-real-time/) a real time signal – SIGRTMIN is used to wake up the sequencer. Clock drifts are found to occur rarely in this current model.

CONCLUSION

The above shown high frequency sequencer model works well for 1Hz target capture. The same model does a good job for 5hz capture. But does not meet the 10Hz requirement for required number of frames. If I were to redesign from the beginning, I would choose a hardware that has a high precision Hardware timer feature, a camera with higher average frame capture capacity and a mechanical clock.

FORMAL REFERENCES

My sincere thanks to Dr. Sam Siewert for helping us throughout the coursework and during the final project development.

I also want to thank the TAs for their best help and support in all the assignment works.

APPENDIX

CODE:

BUILDING AND RUNNING THE CODE:

- Take both the code and makefile from the git link : https://github.com/sundarkrish/ECEN5623-Summer2020/tree/master/exercise6/submission/code
- Run the shell script before executing the code.
- Do "**make all**" .
- Do **./1hz ./5hz or ./10hz** based on which target frequency should be tested. Use **sudo** to run. Define **LOG** at the beginning to output WCET analysis .csv file
- The server. c is compiled using the command : gcc –o server server.c -lrt –lpthread on the remote machine. For running do **./server** . First run the server then start the frame capture code for proper connection sequence.

RESULTS:

All verification image files are here: https://github.com/sundarkrish/ECEN5623-Summer2020/tree/master/exercise6/submission

REFERENCES:

Timestamp in buffer structure in v4l2:::

1. https://stackoverflow.com/questions/10266451/where-does-v4l2-buffer-timestamp-value-starts-counting

About ppm format:::

1. http://paulbourke.net/dataformats/ppm/

create a ppm file:::

1. https://www.youtube.com/watch?v=TS4Y5xw9ryQ
2. https://www.google.com/search?q=ppm+file+format&source=lnms&tbm=bks&sa=X&ved=2ahUKEwinvr-2jZLqAhWUBs0KHR15D7sQ_AUoAXoECBAQCQ&biw=1366&bih=576 – books
3. https://www.geeksforgeeks.org/c-program-to-write-an-image-in-pgm-format/ - grayscale

v4l2 code samples:::

1. http://jwhsmith.net/2014/12/capturing-a-webcam-stream-using-v4l2/ - good

rgb ycbcr pixel format explanation:::

1. http://embeddedprogrammer.blogspot.com/2012/07/hacking-ov7670-camera-module-sccb-cheat.html
2. https://gist.github.com/jayrambhia/5866483

opencv with cuda:::

1. https://medium.com/@mikkelwilson/opencv-with-cuda-acceleration-test-e9f7155e1c84

Demos:::

http://ecee.colorado.edu/~ecen5623/ecen/video/lecture/Standard-Project-Demos/

http://ecee.colorado.edu/~ecen5623/ecen/video/lecture/Creative-Project-Demos/

YUV cannot be stored:::

1. https://www.google.com/books/edition/Digital_Media_Processing/od1PLzHJbJYC?hl=en&gbpv=1&dq=why+convert+yuv422+format+to+rgb&pg=PA540&printsec=frontcover

cuda tutorial in c:::

1. https://cuda-tutorial.readthedocs.io/en/latest/tutorials/tutorial01/

v4l2 with opencv:::

1. https://gist.github.com/jayrambhia/5866483

omit the first 9 frames always:::

1. http://ecee.colorado.edu/~ecen5623/ecen/ex/Linux/simple-capture-1800/capture.c

clock_nanosleep example:::

1. https://gist.github.com/gandro/95fa51fb2263891c56926595a90190db

sigwait and sig masking:::

1. https://linux.die.net/man/3/pthread_sigmask
2. https://devarea.com/linux-handling-signals-in-a-multithreaded-application/#.XyTwZ-pKjIU

socket:::

1. https://stackoverflow.com/questions/15445207/sending-image-jpeg-through-socket-in-c-linux

vm bridge adaptor:::

1. https://stackoverflow.com/questions/48138413/how-to-connect-through-socket-to-virtual-machine
2. https://blogs.oracle.com/scoter/networking-in-virtualbox-v2

Recv receiving 0 bytes:::

1. https://stackoverflow.com/questions/3091010/recv-socket-function-returning-data-with-length-as-0