ECEN-5623 PROJECT REPORT

YASH GUPTE

Date: 9th August 2019

Introduction

In this project, frames will be captured in realtime at 1Hz and 10Hz. The images captured will be converted to gray scale and stored with 640x480 resolution. The project will be implemented using Rate Monotonic policy. Three main tasks will be apart of the RM schedule, the scheduler, reading frames and conversion of the images to gray scale. Two separate tasks will store the images in the local machine and one of them will send the images to a server via sockets. Both these tasks will run on separate cores to prevent disturbance to the ore running RM scheduled tasks. Once captured, analysis as to the feasibility as well as deadlines being met will be performed. The board used will be Nvidia Jetson Nano.

This project is developed to provide capturing of frames at 1Hz or 10Hz based on user input.
In order to achieve this certain targets and minimum requirements have been set. The requirements and their capability has been explained in detail below.

Requirements:

A. Minimum Requirements

1. The resolution used will be VGA (640x480).
2. Frames will be acquired in 2 separate rates; 1Hz and 10Hz based on user input taken via a bash script.
3. Images will be stored in PPM P5 gray format.
4. Every frame which is saved will incorporate a timestamp in the header for that file.
5. An additional, "uname -a" will be appended to the file's header.

B. Verification of Minimum Requirements

1. The images captured will be compared with an external wall clock over a duration of 30mins with 1800 frames for 1Hz mode of capture.
2. The maximum error between all the 1800 frames will be less than 1s. Which means, images captured must be apart by 1s for 1Hz mode and 100ms for 10Hz mode. A maximum deviation of One frame from the intended final frame must be observed.
3. In order to achieve this, ten initial frames will be checked for the second's hand in motion. Once such a frame is captured, a time of 500ms from this instant will be taken as the starting point for capturing the frames which will be a part of the 1800 frames at 1Hz or 6000 frames at 10Hz.
4. Deadlines missed will be calculated by taking Start and Stop times for each thread. The difference between them will be the execution time. This execution time will be checked against the deadline for the task. If the time it takes for the tasks to execute is less than their set deadline, no deadline was missed.
5. Frame jitter will be calculated by taking time stamps for each captured image and comparing the time difference between each captured image's timestamp. Ideally, this value must remain constant throughout the entire process. <u>For further confirmation, the captured images will be compared with each successive image to check for a difference of specified time between them.</u>
6. An additional time lapse images of a process such as melting of an ice cube will be captured and the same will be converted to a video file MPEG2/MPEG4. This will be attached in the final project folder. This will be implemented using the "ffmpeg" command. – [**IN PROGRESS**]

C. Target goals

Continuous download of frames over Ethernet so that you can run indefinitely and never run out of space on your flash file system – [**IN PROGRESS**]

D. Stretch Goals

Run at a higher frame rate of 10 Hz with continuous download, this time for 9 minutes, which will produce up to 6000 frames (about 6GB uncompressed for 640x480) and repeat the jitter and accumulated latency verification at this higher rate.

Real-Time requirements

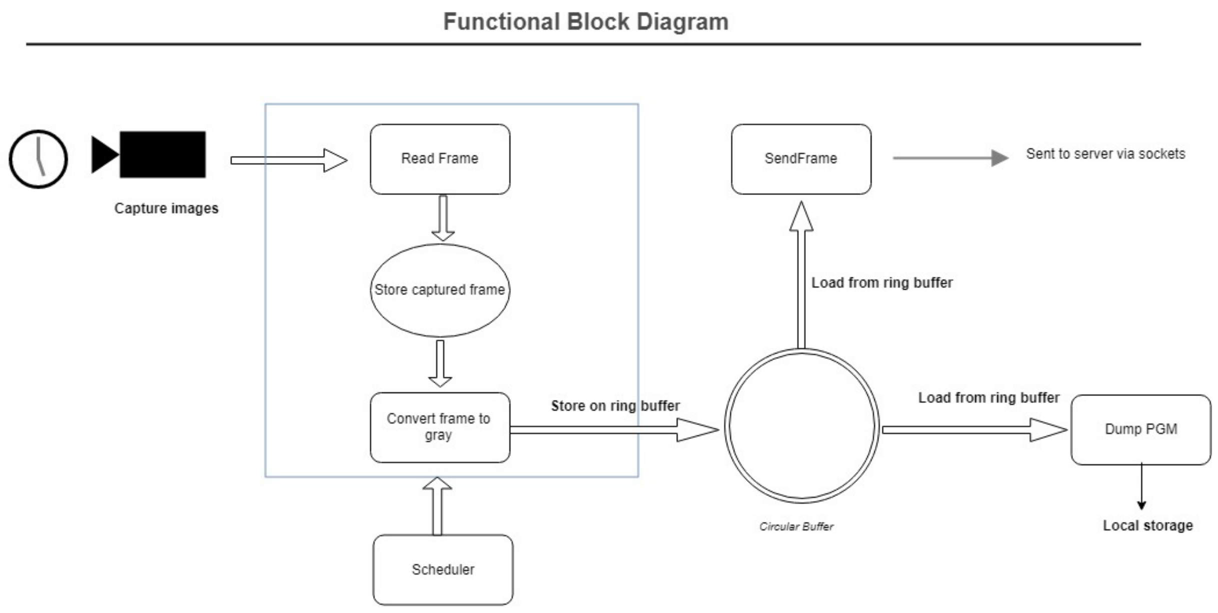
1. **Scheduler**
   This task runs on CPU-3 and is responsible for releasing semaphores at fixed intervals for other 2 tasks; ReadFrame and Gray Conversion. The scheduler runs at every 10ms.
   a. C = Very small around few us.
   b. T = D = 10ms.
   The C is determined by taking the time difference between the start of the scheduler and just before the scheduler loop begins.
   The WCET has been calculated by comparing all the recorded execution times and finding the largest number.
   The runs at a frequency of 100Hz. The scheduler is responsible for releasing the semaphores for the other 2 tasks under RM policy. Since 2 other tasks depend on it, it must run at a frequency which minimizes chances of missed deadlines for the tasks it controls. For example, the scheduler runs every 10ms, the read frame runs at 20ms and gray conversion runs at 100ms. They are all multiples of 10ms – scheduler frequency. Based on the scheduler frequency, these tasks are timed.

2. **Read Frame**
   This task runs on CPU-3 and is responsible for capturing camera images at fixed intervals.
   The ReadFrame runs at every 20ms.
   a. C avg = 26us, Cwcet = 40us.
   b. T = D = 100ms.
   The C is determined by taking the time difference between the release of semaphore in scheduler and just before the read frame loop ends. The WCET has been calculated by comparing all the recorded execution times and finding the largest number.
   The ReadFrame thread runs every 20ms.That is, it runs at a multiple of the scheduler frequency. It runs at this rate to guarantee a frame is present for gray conversion to process when it runs.

3. **Gray Conversion**
   This task runs on CPU-3 and is responsible for converting images from the circular buffer at fixed intervals of 100ms or 1s.
   The ReadFrame runs at every 20ms.
   a. C = 3ms, Cwcet = 3ms.
   b. T = D = 100ms.
   The execution time is determined by taking the time difference between release of semaphore from the scheduler to completion of an iteration in the loop right before setting the Gray mask bit. The WCET is the maximum of all the captured time differences.
   The frequency it runs at varies depending on user input of 1Hz or 10Hz. It runs after every 100ms if input is set to 10Hz, if not it runs every 1s.

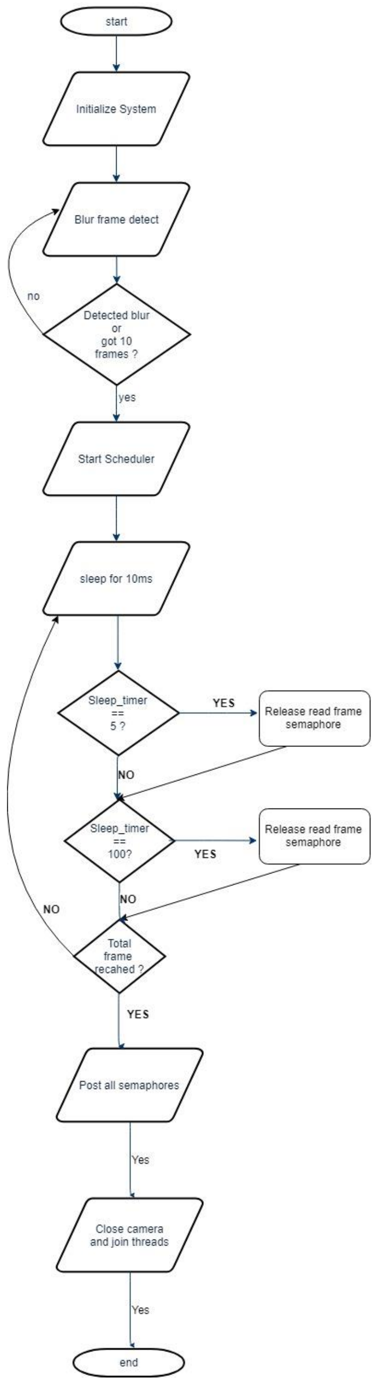
4. **Dump PGM**
   This task runs on CPU-2 and is responsible for storing images in the local PC from the circular buffer. This runs in a continuous loop. Since it is not a part of RM, no deadline has been fixed.
   C = 1ms.

Block Diagrams

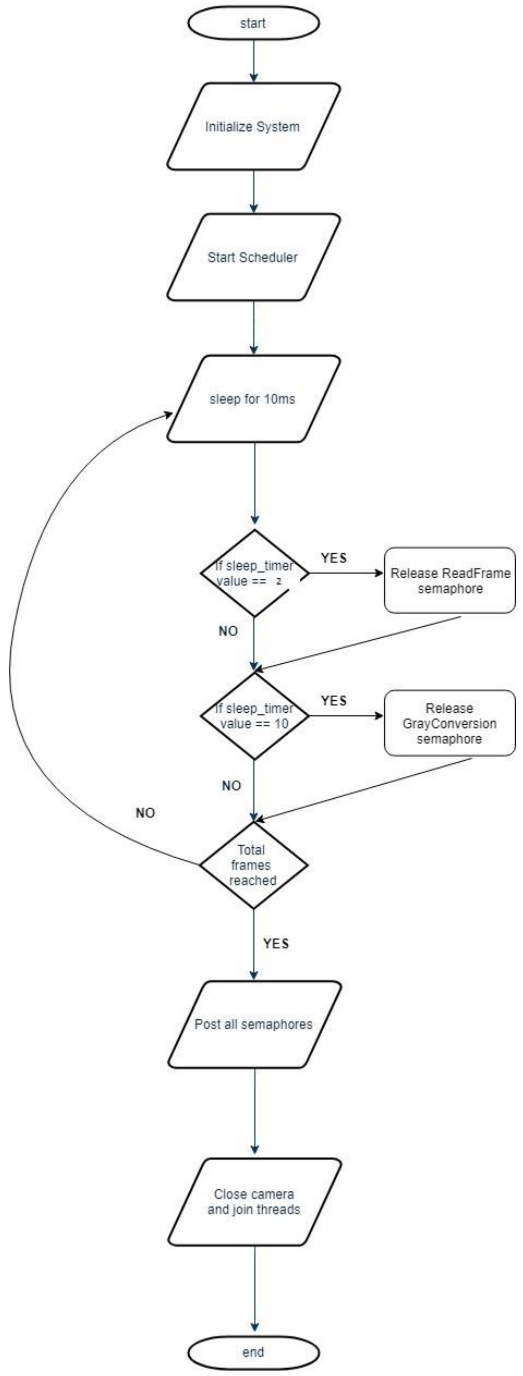


The diagram above describes the hardware and software components of the system. The camera captures images which are read by ReadFrame thread. This thread stores the images in a global variable. This global variable is accessed by Convert to gray thread and stored on the ring buffer. Scheduler is responsible for the releasing the semaphores for ReadFrame and Convert to Gray threads. Parallelly, Dump PGM reads the ring buffer for image data and stores images locally. (***In progress***) Parallelly, SendFrame reads data from ring buffer and sends the data to server via sockets.

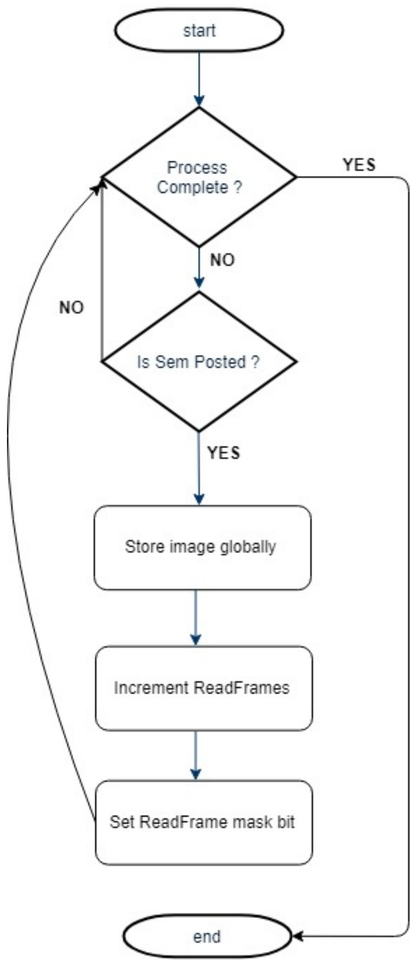
Scheduler Flow Chart 1Hz



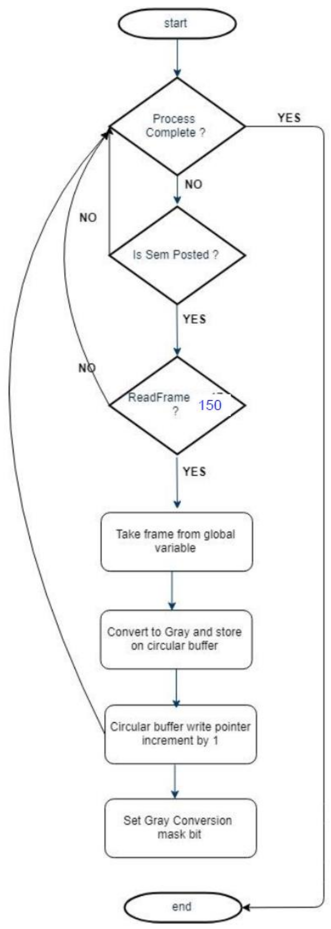
Scheduler Flow Chart at 10Hz


start
Initialize System
Start Scheduler
sleep for 10ms
If sleep_timer value == 2
YES
Release ReadFrame semaphore
NO
If sleep_timer value == 10
YES
Release GrayConversion semaphore
NO
NO
Total frames reached
YES
Post all semaphores
Close camera and join threads
end

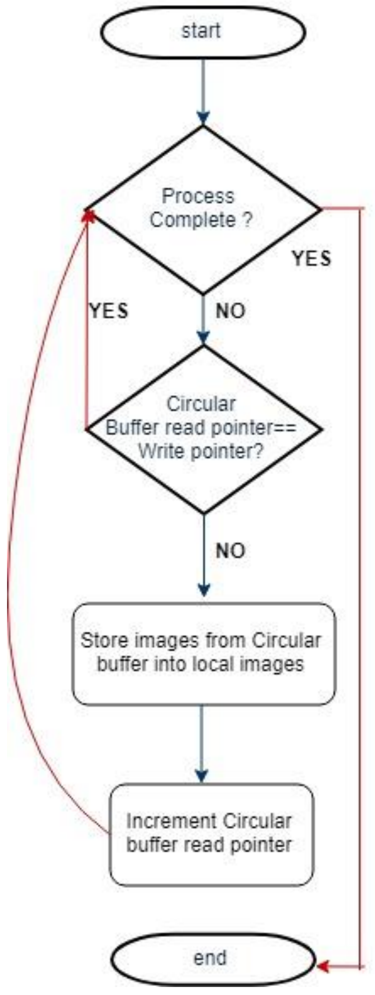
Read Thread Thread Flow Chart



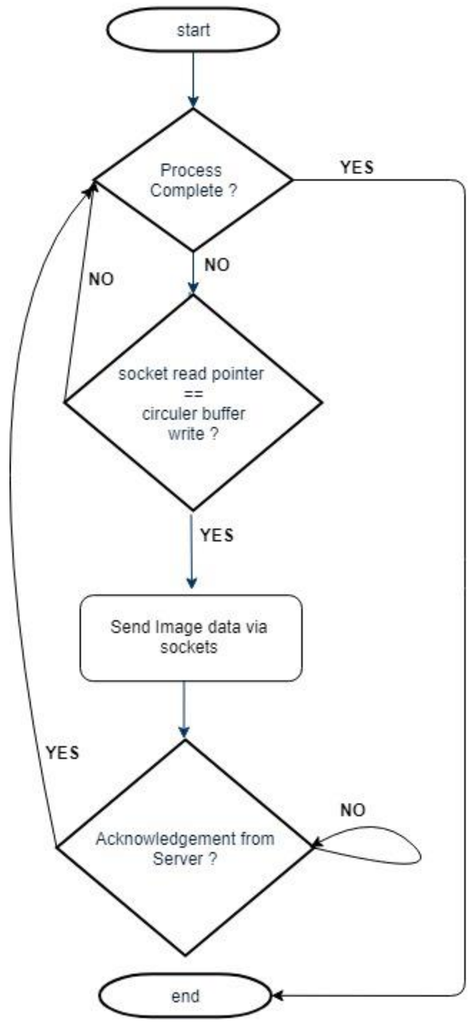
Gray Thread Thread Flow Chart

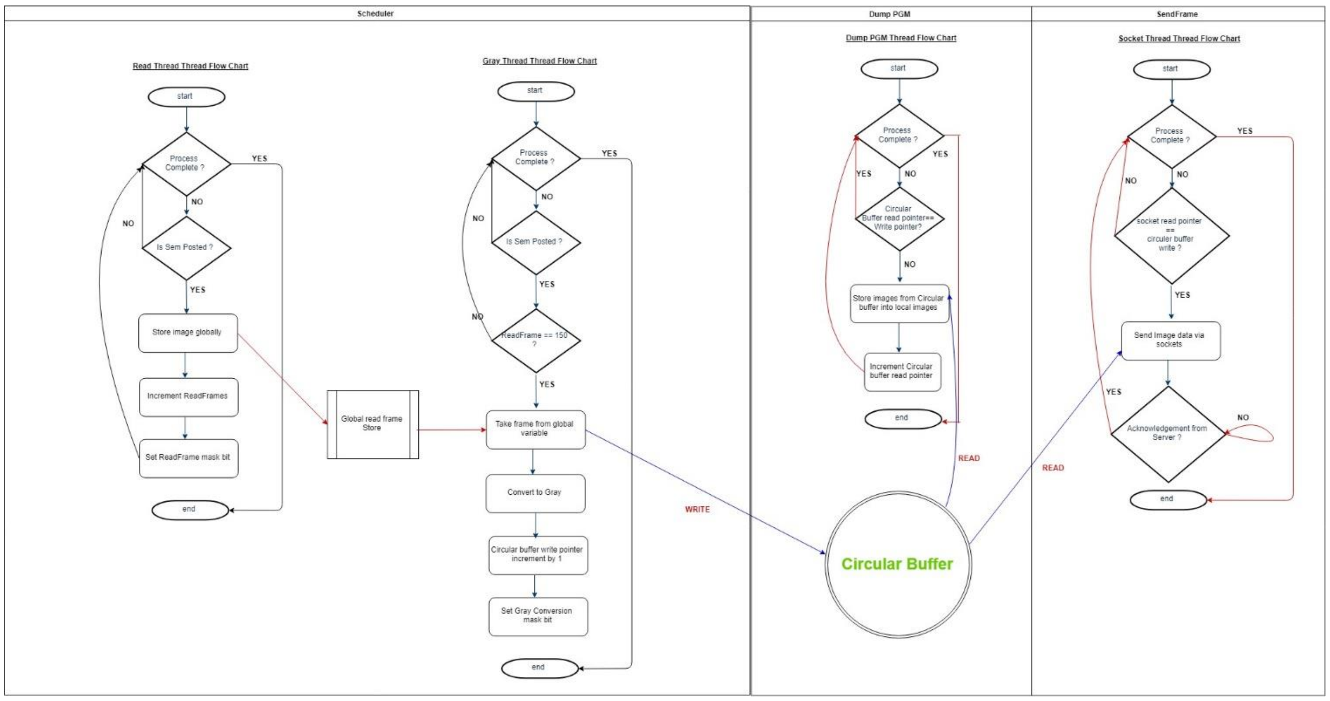

start
Process Complete ?
YES
NO
Is Sem Posted ?
NO
YES
ReadFrame ?
150
NO
YES
Take frame from global variable
Convert to Gray and store on circular buffer
Circular buffer write pointer increment by 1
Set Gray Conversion mask bit
end

Dump PGM Thread Flow Chart


start
Process Complete ?
YES
NO
YES
Circular Buffer read pointer== Write pointer?
NO
Store images from Circular buffer into local images
Increment Circular buffer read pointer
end

Socket Thread Thread Flow Chart


start
Process Complete ?
YES
NO
NO
socket read pointer == circuler buffer write ?
YES
Send Image data via sockets
YES
Acknowledgement from Server ?
NO
end

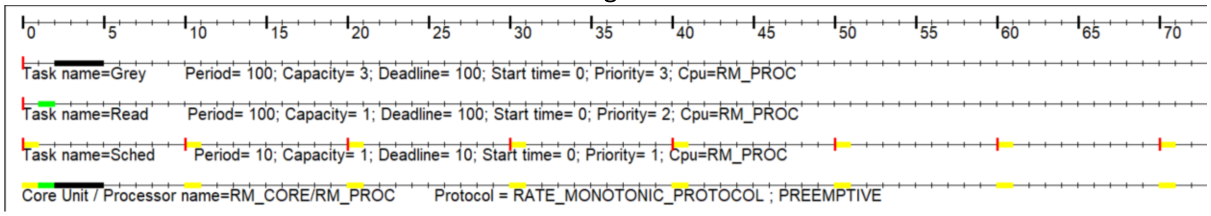

Scheduler
Dump PGM
SendFrame
Read Thread Thread Flow Chart
start
Process Complete ?
YES
NO
Is Sem Posted ?
YES
NO
Store image globally
Increment ReadFrames
Set ReadFrame mask bit
end
Global read frame Store
Gray Thread Thread Flow Chart
start
Process Complete ?
YES
NO
Is Sem Posted ?
YES
NO
ReadFrame == 150 ?
YES
NO
Take frame from global variable
Convert to Gray
Circular buffer write pointer increment by 1
Set Gray Conversion mask bit
end
WRITE
Circular Buffer
Dump PGM Thread Flow Chart
start
Process Complete ?
YES
NO
Circular Buffer read pointer== Write pointer?
YES
NO
Store images from Circular buffer into local images
Increment Circular buffer read pointer
end
READ
Socket Thread Thread Flow Chart
start
Process Complete ?
YES
NO
socket read pointer == circuler buffer write ?
NO
YES
Send Image data via sockets
Acknowledgement from Server ?
YES
NO
end
READ

Analysis of Rate Monotonic tasks

There are 3 tasks running on a common core under RM policy. These tasks include, Scheduler, Read frame and Gray conversion. Each of the tasks have the following specifications:

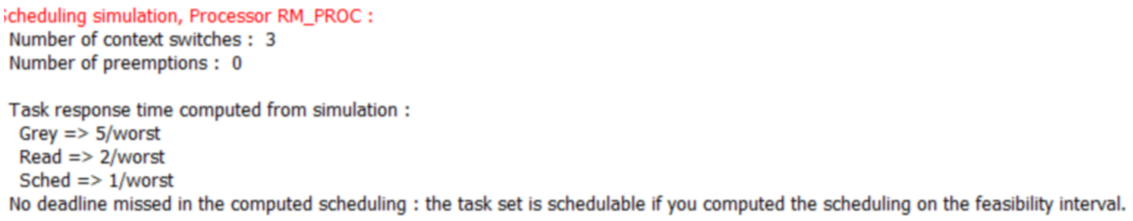
1. **Scheduler**
   a. C = 1ms
   b. T=D = 10ms
2. **Read Frame**
   c. C = 40us
   d. T=D = 100ms
3. **Gray conversion**
   e. C = 3ms
   f. T=D = 100ms

This data was entered in Cheddar and the following results were obtained.

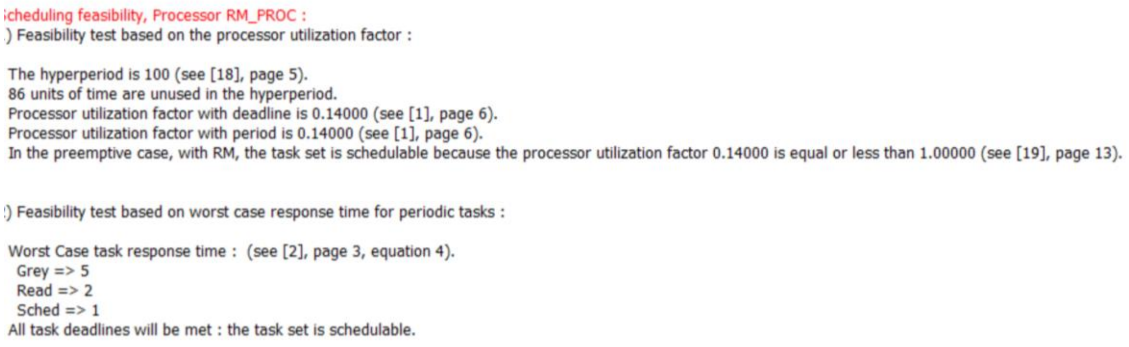


```
Task name=Grey      Period= 100; Capacity= 3; Deadline= 100; Start time= 0; Priority= 3; Cpu=RM_PROC
Task name=Read      Period= 100; Capacity= 1; Deadline= 100; Start time= 0; Priority= 2; Cpu=RM_PROC
Task name=Sched     Period= 10; Capacity= 1; Deadline= 10; Start time= 0; Priority= 1; Cpu=RM_PROC
Core Unit / Processor name=RM_CORE/RM_PROC     Protocol = RATE_MONOTONIC_PROTOCOL ; PREEMPTIVE
```

Fig. Cheddar Analysis over 100ms



```
cheduling simulation, Processor RM_PROC :
 Number of context switches :  3
 Number of preemptions :  0

 Task response time computed from simulation :
   Grey => 5/worst
   Read => 2/worst
   Sched => 1/worst
 No deadline missed in the computed scheduling : the task set is schedulable if you computed the scheduling on the feasibility interval.
```

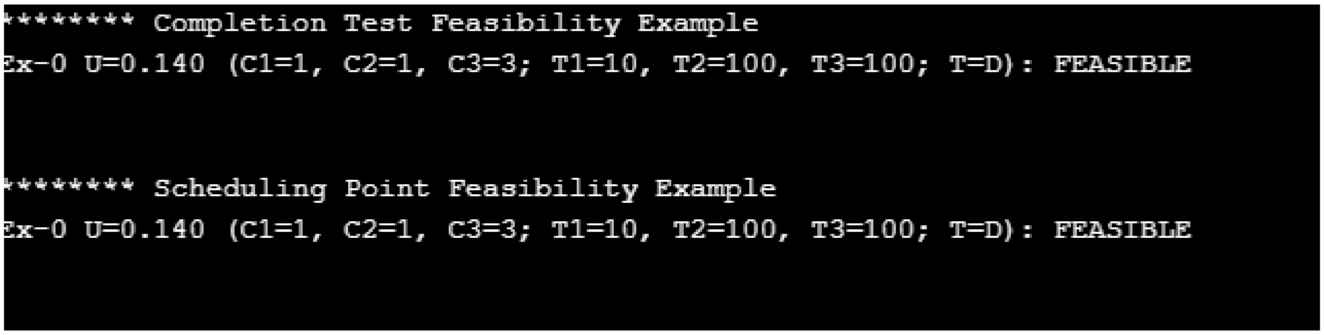


```
cheduling feasibility, Processor RM_PROC :
) Feasibility test based on the processor utilization factor :

 The hyperperiod is 100 (see [18], page 5).
 86 units of time are unused in the hyperperiod.
 Processor utilization factor with deadline is 0.14000 (see [1], page 6).
 Processor utilization factor with period is 0.14000 (see [1], page 6).
 In the preemptive case, with RM, the task set is schedulable because the processor utilization factor 0.14000 is equal or less than 1.00000 (see [19], page 13).


) Feasibility test based on worst case response time for periodic tasks :

 Worst Case task response time :  (see [2], page 3, equation 4).
   Grey => 5
   Read => 2
   Sched => 1
 All task deadlines will be met : the task set is schedulable.
```

Fig. Cheddar Analysis over 100ms

The Cheddar analysis shows that the tasks are schedulable even under worst case execution time.

This can be further clarified by running Schedule-Point tests and Completion-Point tests. The results for which are given below:



```
******** Completion Test Feasibility Example
Ex-0 U=0.140 (C1=1, C2=1, C3=3; T1=10, T2=100, T3=100; T=D): FEASIBLE


******** Scheduling Point Feasibility Example
Ex-0 U=0.140 (C1=1, C2=1, C3=3; T1=10, T2=100, T3=100; T=D): FEASIBLE
```

Fig. Schedule Point test and Completion Test

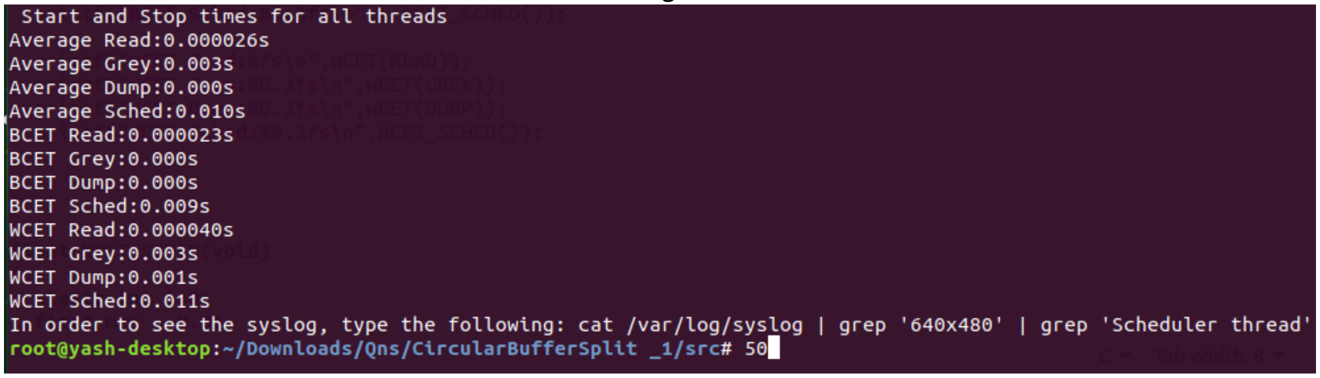
Hand-made analysis attached in the link below:
[INSERT LINK]

Safety Margin

The total time taken for Read Frame and Gray conversion under worst case scenario is 5ms. The deadline given in 100ms for 10Hz and 1s for 1Hz. This shows that the tasks have a safety margin of 96ms for 100ms deadline and s for a deadline of 995ms.

Feasibility Margin

The variations in C and WCET can be seen from the image below:



```
 Start and Stop times for all threads
Average Read:0.000026s
Average Grey:0.003s
Average Dump:0.000s
Average Sched:0.010s
BCET Read:0.000023s
BCET Grey:0.000s
BCET Dump:0.000s
BCET Sched:0.009s
WCET Read:0.000040s
WCET Grey:0.003s
WCET Dump:0.001s
WCET Sched:0.011s
In order to see the syslog, type the following: cat /var/log/syslog | grep '640x480' | grep 'Scheduler thread'
root@yash-desktop:~/Downloads/Qns/CircularBufferSplit _1/src# 50
```

Fig. Start and Stop times for all threads

The difference between the best-case execution time and worst-case execution time:

1. Scheduler thread: 0ms (value will be in us)
2. Read Thread: 40us-26us = 14us
3. Gray thread: 0.003-0.003 = 0ms

Error margins between BCET and WCET:

1. Scheduler thread = about 0%
2. Read Thread = 42.5%
3. Gray thread = about 0%

Frame Jitter

Over one entire cycle of 100ms, the CPU utilization can be calculated as follows:
Scheduler runs 10 times in 100ms = 10*(WCET of scheduler) = 10us*10 = 100us = 0.1ms
Read frame runs at 2 times in 100ms = 2*(WCET of read frame) = 2us*40us = 80us = 0.08ms
Gray conversion runs once in 100ms = 1*(WCET of gray conversion) = 3ms

Total values = 0.1 + 0.08 + 3 = 3.18ms

Out of 100ms, 100-3.18 = 96.82ms
Out of 1s, 1000-3.09 = 996.82ms.

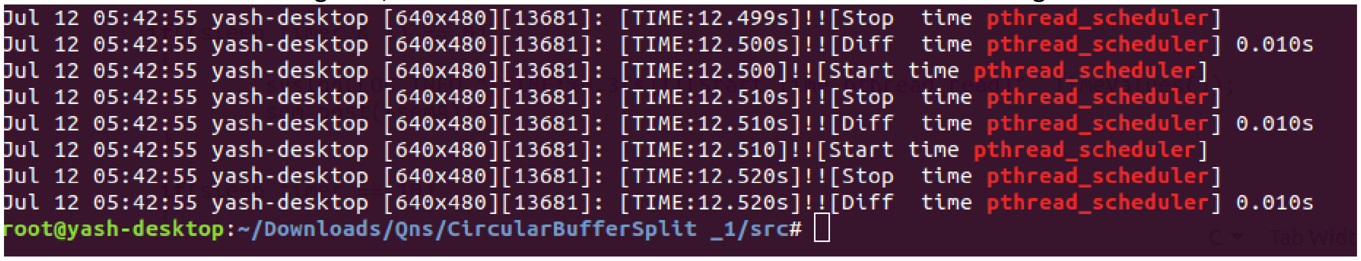
Real-Time system analysis

The images below give the request time (start time), execution time and stop time of each thread in the system.

1. Scheduler Thread

The thread should be requested every 10ms, as it can be noted from the image, each instance of the scheduler thread request/start time is at 10ms. The time for execution comes out as 10ms as well because, the time difference is calculated before the clock_nansleep and after the end of the loop. The 10ms is the time for the sleep loop and the actual execution time, is negligible.

In terms of deadlines being met, it is evident that none are missed from the WCET image above.



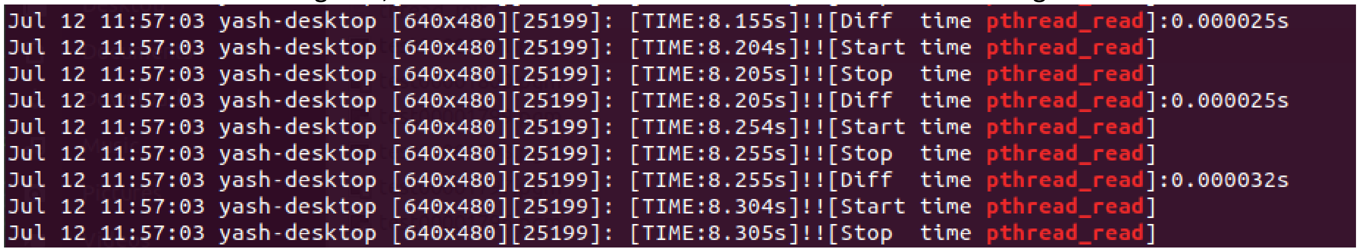
```
Jul 12 05:42:55 yash-desktop [640x480][13681]: [TIME:12.499s]!![Stop  time pthread_scheduler]
Jul 12 05:42:55 yash-desktop [640x480][13681]: [TIME:12.500s]!![Diff  time pthread_scheduler] 0.010s
Jul 12 05:42:55 yash-desktop [640x480][13681]: [TIME:12.500]!![Start time pthread_scheduler]
Jul 12 05:42:55 yash-desktop [640x480][13681]: [TIME:12.510s]!![Stop  time pthread_scheduler]
Jul 12 05:42:55 yash-desktop [640x480][13681]: [TIME:12.510s]!![Diff  time pthread_scheduler] 0.010s
Jul 12 05:42:55 yash-desktop [640x480][13681]: [TIME:12.510]!![Start time pthread_scheduler]
Jul 12 05:42:55 yash-desktop [640x480][13681]: [TIME:12.520s]!![Stop  time pthread_scheduler]
Jul 12 05:42:55 yash-desktop [640x480][13681]: [TIME:12.520s]!![Diff  time pthread_scheduler] 0.010s
root@yash-desktop:~/Downloads/Qns/CircularBufferSplit _1/src#
```

Fig. Scheduler thread

2. Read Thread

The thread should be requested every 20ms, as it can be noted from the image, each instance of the read frame thread request/start time is at 20ms. The time for execution comes out as 40us. The response is quite predictable based on the values below.

In terms of deadlines being met, it is evident that none are missed from the WCET image above.



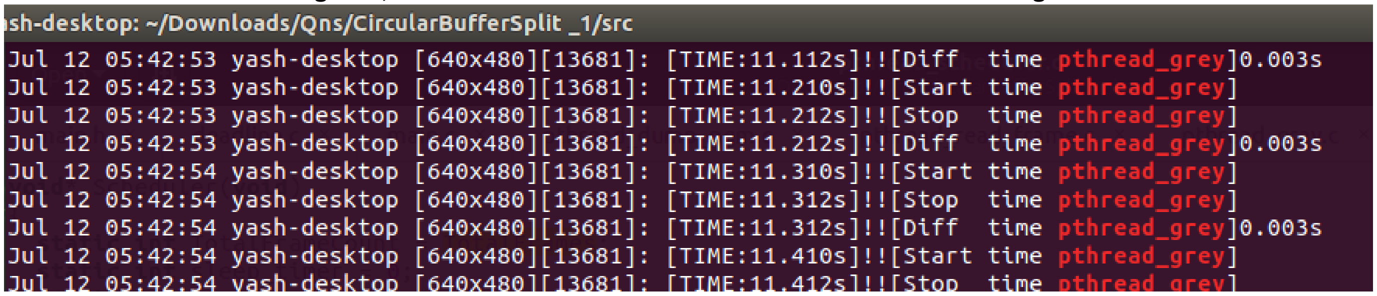
```
Jul 12 11:57:03 yash-desktop [640x480][25199]: [TIME:8.155s]!![Diff  time pthread_read]:0.000025s
Jul 12 11:57:03 yash-desktop [640x480][25199]: [TIME:8.204s]!![Start time pthread_read]
Jul 12 11:57:03 yash-desktop [640x480][25199]: [TIME:8.205s]!![Stop  time pthread_read]
Jul 12 11:57:03 yash-desktop [640x480][25199]: [TIME:8.205s]!![Diff  time pthread_read]:0.000025s
Jul 12 11:57:03 yash-desktop [640x480][25199]: [TIME:8.254s]!![Start time pthread_read]
Jul 12 11:57:03 yash-desktop [640x480][25199]: [TIME:8.255s]!![Stop  time pthread_read]
Jul 12 11:57:03 yash-desktop [640x480][25199]: [TIME:8.255s]!![Diff  time pthread_read]:0.000032s
Jul 12 11:57:03 yash-desktop [640x480][25199]: [TIME:8.304s]!![Start time pthread_read]
Jul 12 11:57:03 yash-desktop [640x480][25199]: [TIME:8.305s]!![Stop  time pthread_read]
```

Fig. Read Frame thread

3. Gray Thread

The thread should be requested every 100ms, as it can be noted from the image, each instance of the Gray conversion thread request/start time is at 100ms. The time for execution comes out as 3ms.

In terms of deadlines being met, it is evident that none are missed from the WCET image above.



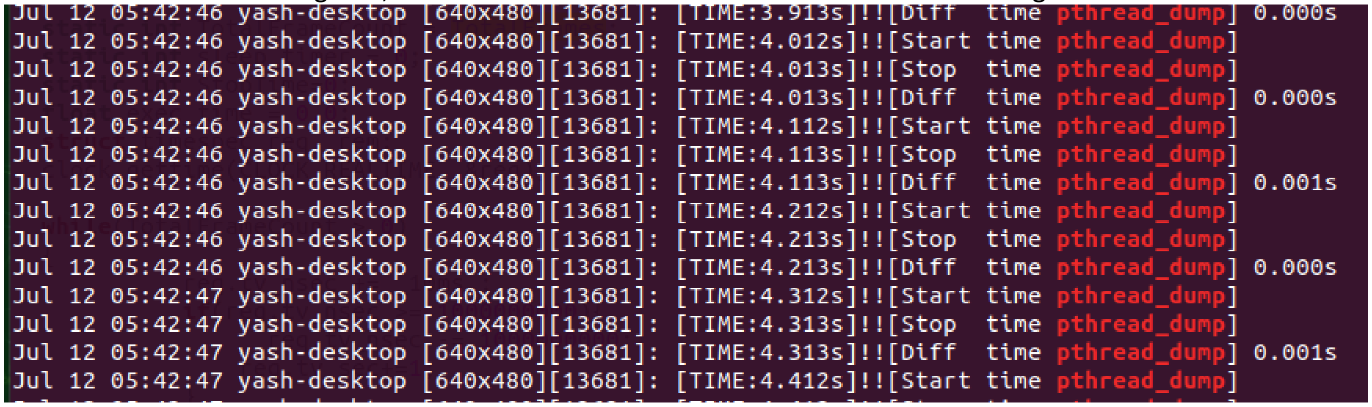
```
sh-desktop: ~/Downloads/Qns/CircularBufferSplit _1/src
Jul 12 05:42:53 yash-desktop [640x480][13681]: [TIME:11.112s]!![Diff  time pthread_grey]0.003s
Jul 12 05:42:53 yash-desktop [640x480][13681]: [TIME:11.210s]!![Start time pthread_grey]
Jul 12 05:42:53 yash-desktop [640x480][13681]: [TIME:11.212s]!![Stop  time pthread_grey]
Jul 12 05:42:53 yash-desktop [640x480][13681]: [TIME:11.212s]!![Diff  time pthread_grey]0.003s
Jul 12 05:42:54 yash-desktop [640x480][13681]: [TIME:11.310s]!![Start time pthread_grey]
Jul 12 05:42:54 yash-desktop [640x480][13681]: [TIME:11.312s]!![Stop  time pthread_grey]
Jul 12 05:42:54 yash-desktop [640x480][13681]: [TIME:11.312s]!![Diff  time pthread_grey]0.003s
Jul 12 05:42:54 yash-desktop [640x480][13681]: [TIME:11.410s]!![Start time pthread_grey]
Jul 12 05:42:54 yash-desktop [640x480][13681]: [TIME:11.412s]!![Stop  time pthread_grey]
```

Fig. Gray conversion

4. Dump Thread

The thread should be requested every 100ms, as it can be noted from the image, each instance of the Gray conversion thread request/start time is at 100ms. The time for execution comes out as 1ms.

In terms of deadlines being met, it is evident that none are missed from the WCET image above.



```
Jul 12 05:42:46 yash-desktop [640x480][13681]: [TIME:3.913s]!![Diff  time pthread_dump] 0.000s
Jul 12 05:42:46 yash-desktop [640x480][13681]: [TIME:4.012s]!![Start time pthread_dump]
Jul 12 05:42:46 yash-desktop [640x480][13681]: [TIME:4.013s]!![Stop  time pthread_dump]
Jul 12 05:42:46 yash-desktop [640x480][13681]: [TIME:4.013s]!![Diff  time pthread_dump] 0.000s
Jul 12 05:42:46 yash-desktop [640x480][13681]: [TIME:4.112s]!![Start time pthread_dump]
Jul 12 05:42:46 yash-desktop [640x480][13681]: [TIME:4.113s]!![Stop  time pthread_dump]
Jul 12 05:42:46 yash-desktop [640x480][13681]: [TIME:4.113s]!![Diff  time pthread_dump] 0.001s
Jul 12 05:42:46 yash-desktop [640x480][13681]: [TIME:4.212s]!![Start time pthread_dump]
Jul 12 05:42:46 yash-desktop [640x480][13681]: [TIME:4.213s]!![Stop  time pthread_dump]
Jul 12 05:42:46 yash-desktop [640x480][13681]: [TIME:4.213s]!![Diff  time pthread_dump] 0.000s
Jul 12 05:42:47 yash-desktop [640x480][13681]: [TIME:4.312s]!![Start time pthread_dump]
Jul 12 05:42:47 yash-desktop [640x480][13681]: [TIME:4.313s]!![Stop  time pthread_dump]
Jul 12 05:42:47 yash-desktop [640x480][13681]: [TIME:4.313s]!![Diff  time pthread_dump] 0.001s
Jul 12 05:42:47 yash-desktop [640x480][13681]: [TIME:4.412s]!![Start time pthread_dump]
```

Fig. Dump image

Taking into consideration the execution times of all the RM threads with WCET: 0.001 + 0.001 + 0.003 = 0.005s or 5ms. With a deadline of 100ms, the time margin left is 95ms. This means a margin of 995ms still remains after execution in each cycle. If 1s intervals are considered, the amount of time taken will be the same but there will still be about 995ms margin.

Proof of Concept with images

https://drive.google.com/open?id=1ynCMmL5F4jtuZw0jr19SS8MeCxzuhNKc

Conclusion

In the project, frames were captured at two separate rates of 10Hz and 1Hz. The captured images were stored and subsequently processed by converting them to gray scale images of 640x480 resolution. The I/O was decoupled to facilitate independent write of images in the local storage as well as to send frames via sockets. The tasks responsible for achieving this objective were based on Rate Monotonic policy. These tasks included Scheduler, Read Frame and Gray conversion. Additional tasks Dump PGM and SendFrames ran on separate cores. This prevented them from disturbing the timing of the RM based tasks. Analysis for schedule feasibility and WCET was performed. All the frames were captured successfully without missing deadlines or frames.

Formal References

People consulted:
Steve Rizor – SA ECEN-5623

Papers and Websites consulted:

1. https://en.wikipedia.org/wiki/Network_socket
2. https://en.wikipedia.org/wiki/Network_Time_Protocol
3. https://en.wikipedia.org/wiki/International_Atomic_Time
4. https://en.wikipedia.org/wiki/Coordinated_Universal_Time
5. https://www.researchgate.net/publication/221224202_Rate_monotonic_scheduling_of_real-time_control_systems_with_the_minimum_number_of_priority_levels
6. https://onlinelibrary.wiley.com/doi/abs/10.1002/pra2.2018.14505501018
7. http://ijcsit.com/docs/Volume%205/vol5issue03/ijcsit20140503462.pdf

Codes Referred:

1. http://ecee.colorado.edu/~ecen5623/ecen/ex/Linux/code/Feasibility/feasibility_tests.c - Feasibility test.
2. https://www.geeksforgeeks.org/socket-programming-cc/ - Socket programming

Appendix

1. Link to bash script:
2. Link to images:
3. Link to videos:
4. Link to codes:
5. Link to analysis: