Real Time Embedded Systems – ECEN 5623

Summer 2019

Professor Sam Siewert

Report By: Poorn Mehta

Platform: Jetson Nano (Camera: Logitech C920s)

Homework 6

Q1: Provide all major functional capability requirements for your real-time software system [not just for the proof-of-concept aspect to be demonstrated and analyzed, but the whole design concept envisioned]. You should have at least 5 major requirements or more.

Solution:

Major Functional Capabilities & Requirements

1. The system must be able to capture raw frames from the camera, continuously and in real time till the end of program execution.
2. The captured frames, must be processed in real time, and passed to best effort threads running on other CPU cores for further use.
3. All real time tasks, must be achieved by the use of Rate Monotonic Theory, where these tasks will be implemented by threads, all of which will share a single CPU core.
4. The system must be able to lock frame capture rates, which emulates a real time service with fixed deadline, and periodic request frequency.
5. The system must be able to support at least two different framerates on which it can lock – one of it being 1Hz and another being 10Hz.
6. The system must be logging all significant events, along with timestamps (either absolute or relative).
7. Largely, the system must be free of significant jitter in the operation (continuous as well as accumulated), specifics of which will be clearly defined in the next section.
8. The system must be able to store all captured frames, as long as the total count it within a certain limit (this could vary based on the size of each image, as well as the memory budgeting plans of the user – for now, it is kept at 2000 frames with each frame being 900KB in size).
9. If the user wishes to capture more frames than allowed by the maximum cap to store these, the system must keep on overwriting previous frames. Regardless of the numbering of images, in any case, user should have all latest frames.
10. The system must store all captured frames in .PPM format, each carrying the system information (to indicate the platform used for the project) and absolute timestamps (reflecting when that specific frame was captured).
11. Once the required number of frames have been captured, processed, stored, and sent to the remote (in case this feature is enabled), the system must free any resources taken by it, and exit cleanly.

Q2: Complete a high-level real-time software system functional description and proposal along with a single page block diagram showing major elements (hardware and software) in your system (example). You must include a single page block diagram, but also back this up with more details in corresponding CFD/DFD, state-machine, ERD and flowchart diagrams as you see fit (2 more minimum). Please provide this in your report for this assignment.

Solution:

Software System Functional Description

1. The software developed is capable of handling both – real time requirements, as well as other functional requirements, with given resources, and without any major glitches/bugs.
2. Software is written for Linux in C, primarily because the camera driver interface is readily available, and the project itself is rather a soft real time (it definitely aims to emulate a hard-real time system however, there is no actual loss if it fails to meet those requirements).
3. For this, the software utilizes FIFO scheduling policy for all of the threads operating. This is true even the best efforts ones – to reduce unnecessary complexity in the design, with the assumption that nothing more important than this software itself – is running in the user space at the time of execution.
4. 3 Real time services have been identified, and created – each with different priority.
5. The thread/service having highest priority among them – is called Scheduler. As the name suggests, the task implemented in this thread is relatively simple – to run at a much higher frequency (than required by other real time threads), and release services on fixed periodic basis. Moreover, this thread keeps a track of number of captured frames, and exits when the target is achieved.
6. The medium priority thread is called Cam_Monitor. It polls the camera on periodic basis, and if a frame is available, then it updates the pointer in a shared structure, so that the image processing thread can have the latest data (buffer of image pixels).
7. The thread having lowest priority among all of these – is Cam_RGB, which implements converting the raw captured frame (YUYV – 4:2:2) to RGB. To achieve this, it has to process on pixel level, which is quite CPU intensive.
8. Cam_RGB also prepares the file name, header etc. and shares this – along with a pointer which points to the actual buffer with all pixel data, to Storage thread and to Socket thread (if enabled), using Posix message queues.
9. Both of these threads are running on a separate CPU core, and in best effort mode. Reception of a valid message from Cam_RGB unlocks it, and they execute their specific instructions.
10. Storage thread simply writes the entire buffer to flash, by creating a file, while Socket thread sends it over the Socket connection, in smaller segments (so that it doesn't exceed MTU size and become corrupted).
11. For the 1Hz mode, the software initially runs a Cam_Filter function, job of which is to identify the change in position of the second indicator (on analog clock placed in front of camera). Once it is detected, the function marks it, sets the start time of Scheduler accordingly, and exits. The same procedure is planned for 10Hz; however, it is yet to be implemented.

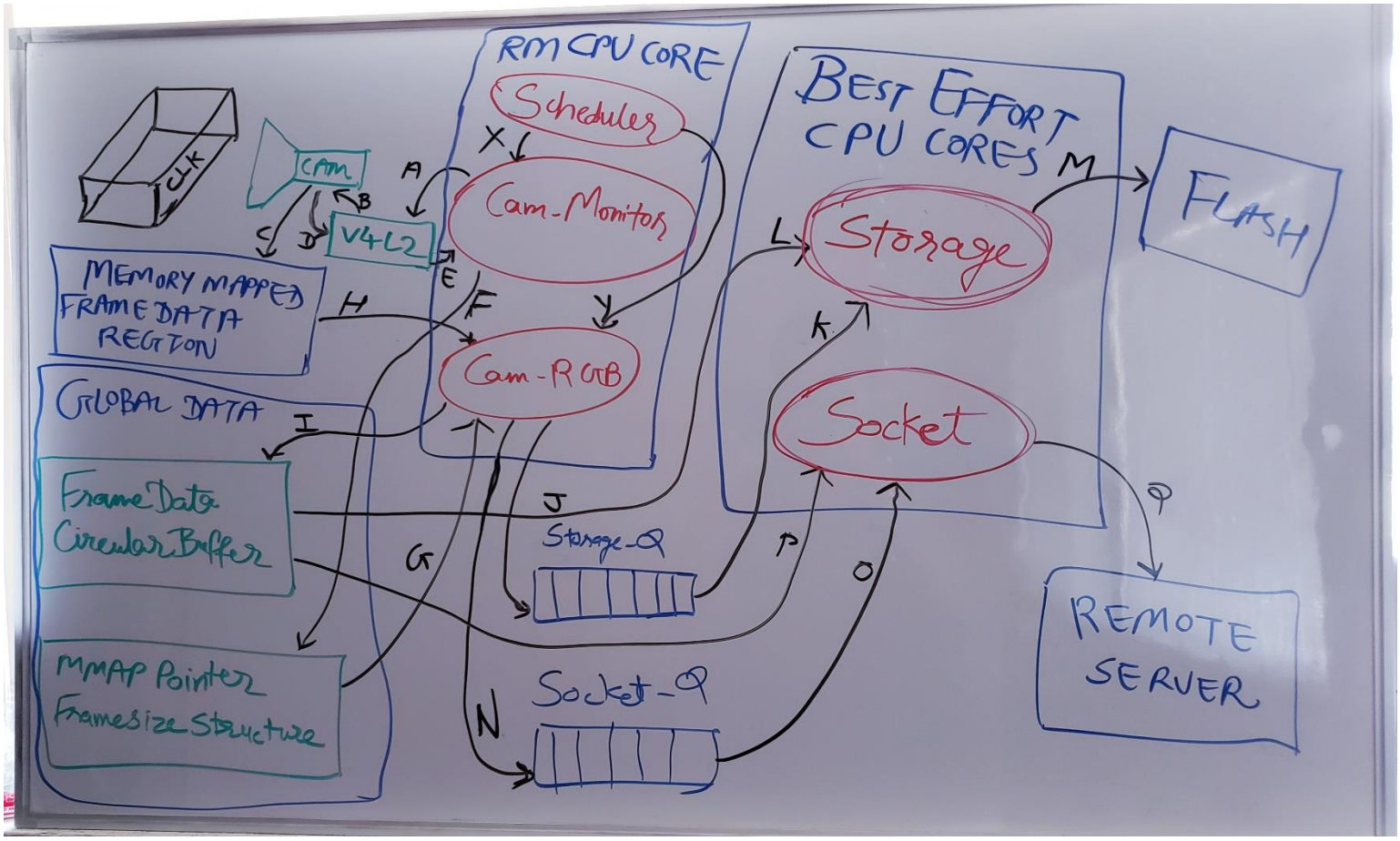
12. Currently, the software runs without Cam_Filter on 10Hz mode, and faces occasional blurred digit frames – however, it is the only problem.
13. All real time threads' frequency (deadlines) are set as per the selected mode – 1Hz or 10Hz. In any case, the scheduler runs at 100x frequency, and others run at 1x.
14. Software also runs a warmup function at the start, since the camera performs auto focus, brightness adjusting, and a few other similar functions at start – which can potentially throw off frame selection (Cam_Filter) processing.
15. The illustrations/diagrams will provide a better insight in the architecture, flow, and functioning of the software.

Software Goals/Proposal

1. Achieve every single thing listed under Major Functional Requirements.
2. The system must not have jitter (on continuous basis) more than 1% of locked FPS deadline, with reference to its own internal clock.
3. The system must not go out of sync (accumulated jitter) with real world clock by more than 1 margin between consecutive frames (for instance, this will be 100ms on 10Hz mode).
4. In 1Hz mode, the system must be able to capture and locally store 3600 frames, while the camera points to a functional ticking analog clock, and all of these frames should be free of any blur in the position of second indicator (blur resulting on the image due to autofocus may be ignored).
5. In 10Hz mode, the system must be able to capture and locally store 6000 frames, while the camera points to a digital online stopwatch, and the system also should have as less blurry frames as possible in the process.
6. The system must never miss a single deadline, under any condition whatsoever.
7. The I/O must be detached from the real time threads, since the it can have very large delays (especially due to flash access in case of the Storage).
8. The system must encase event tracing and analysis of every thread, covering aspects such as worst-case execution time, largest jitter, missed deadlines, etc.
9. The software must prove itself to be fairly reliable, easily testable and verifiable, while satisfying all of the necessities.

Block Diagram and Description of Events

All of the flow charts can be found **here**, in high definition.

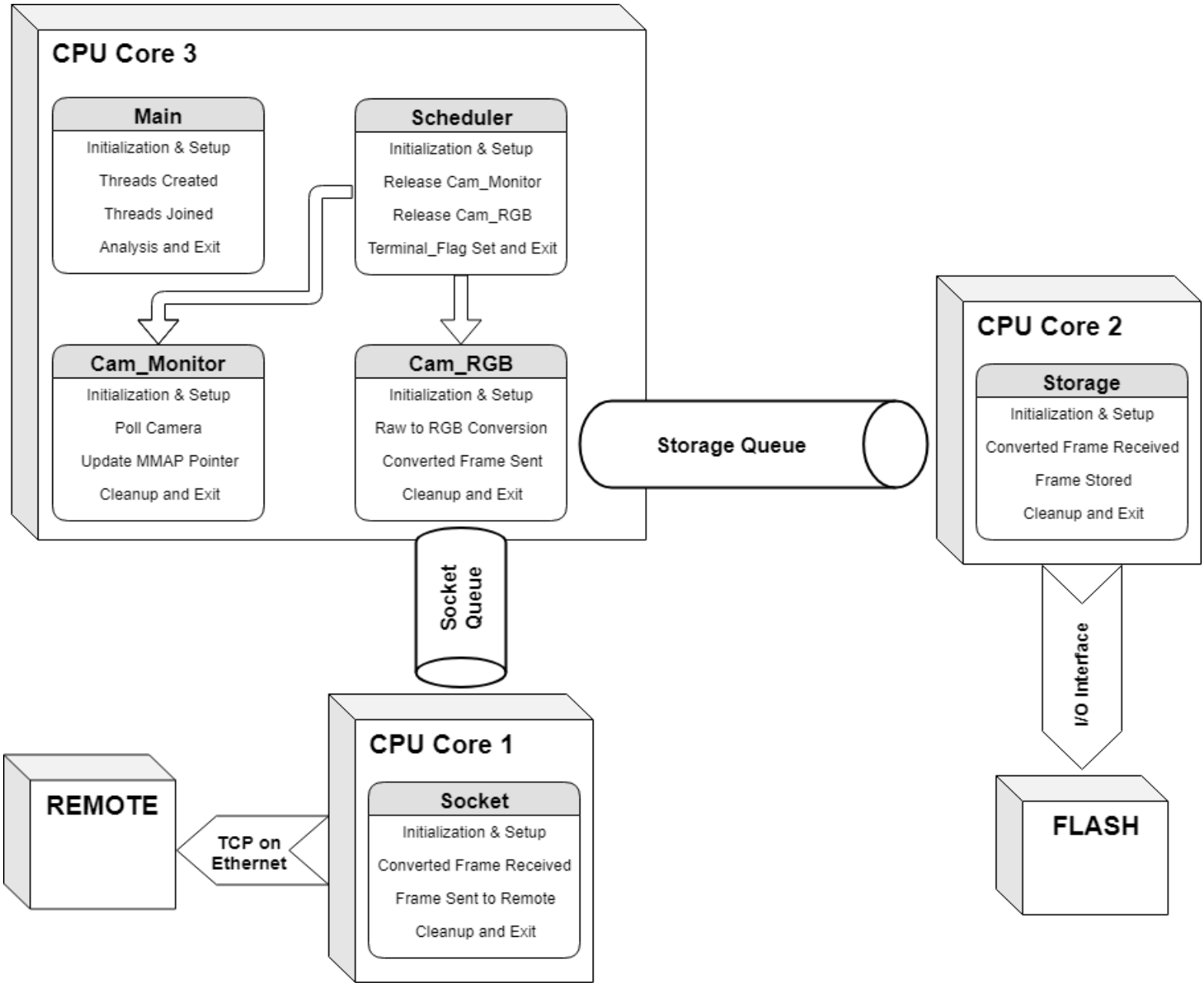


X and Y represents service release signals from Scheduler to Cam_Monitor, and Cam_RGB respectively.

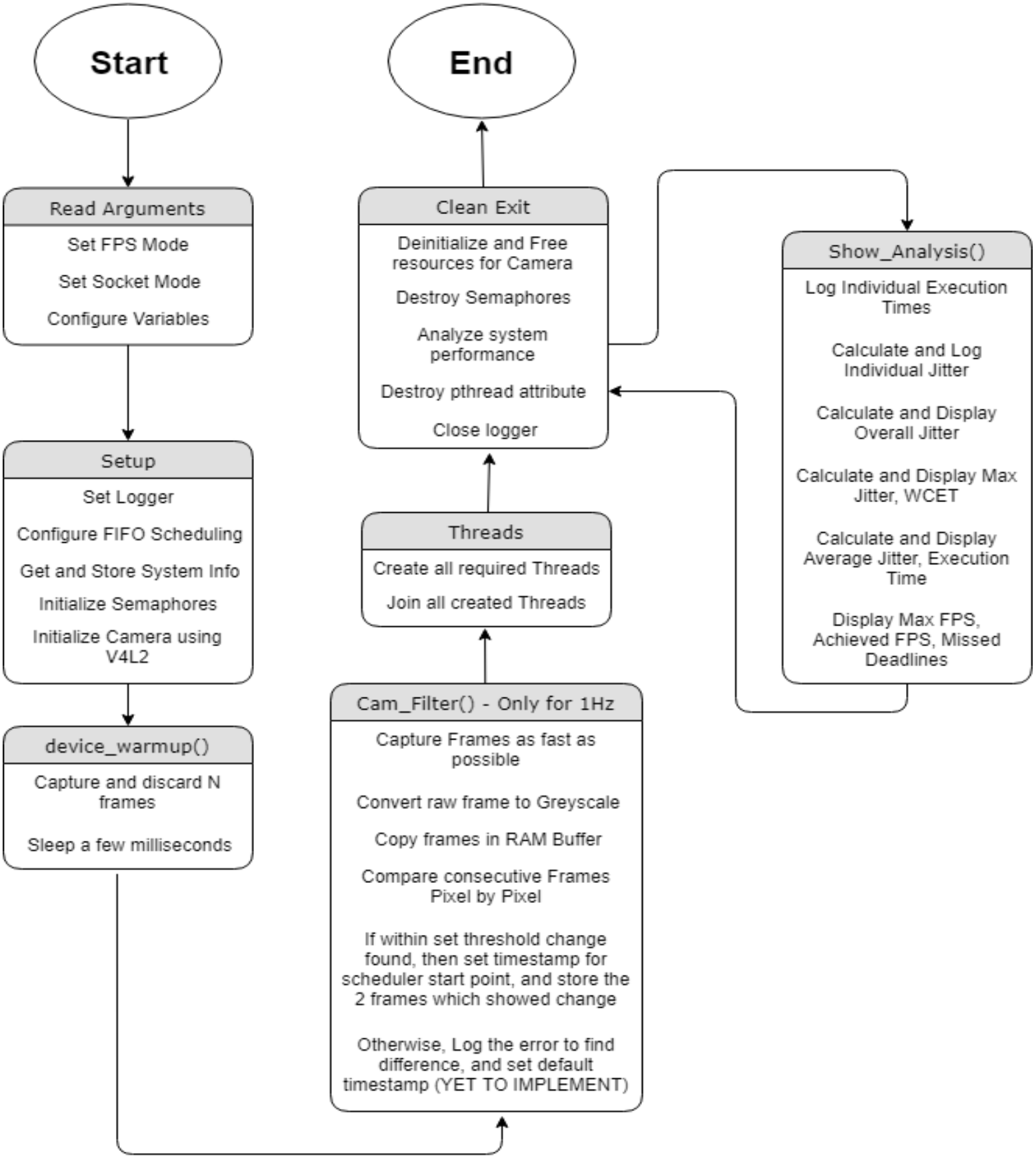
A. Cam_Monitor issues a dequeue request from the Camera using V4L2 API.
B. V4L2 performs the dequeue operation by interfacing with Camera.
C. Camera driver stores the raw frame data in the memory mapped region.
D. Camera sends success/confirmation to the V4L2.
E. V4L2 returns with success to Cam_Monitor.
F. Cam_Monitor updates the pointer – which locates the memory mapped region containing the raw frame data, in globally shared structure.
G. The updated pointer and structure information is read by Cam_RGB.
H. Cam_RGB pulls the raw data out of the memory mapped region, and processes it to convert each pixel into an RGB one.
I. Cam_RGB stores the processed frame pixels in a globally shared circular buffer (statically allocated) residing on RAM.
J. Cam_RGB prepares an image information structure, containing file name, header information, and the pointer pointing to the RAM buffer index – to Storage thread using POSIX Queue.

K. Storage thread receives a valid message from Cam_RGB, creates a file and writes header, using the structure message received.
L. Storage thread pulls image from RAM buffer and starts to write it.
M. Storage thread completes the image writing, and the file is stored in FLASH.
N. Cam_RGB, after conversion, also sends file information structure to Socket thread using POSIX Queue.
O. Socket receives a valid message from Cam_RGB, and starts sending out frame information data via the TCP socket connection (utilizing ethernet port).
P. Socket segments the large file into smaller pieces of 1KB each, and repeatedly sends it until remote computer has received the entire file.

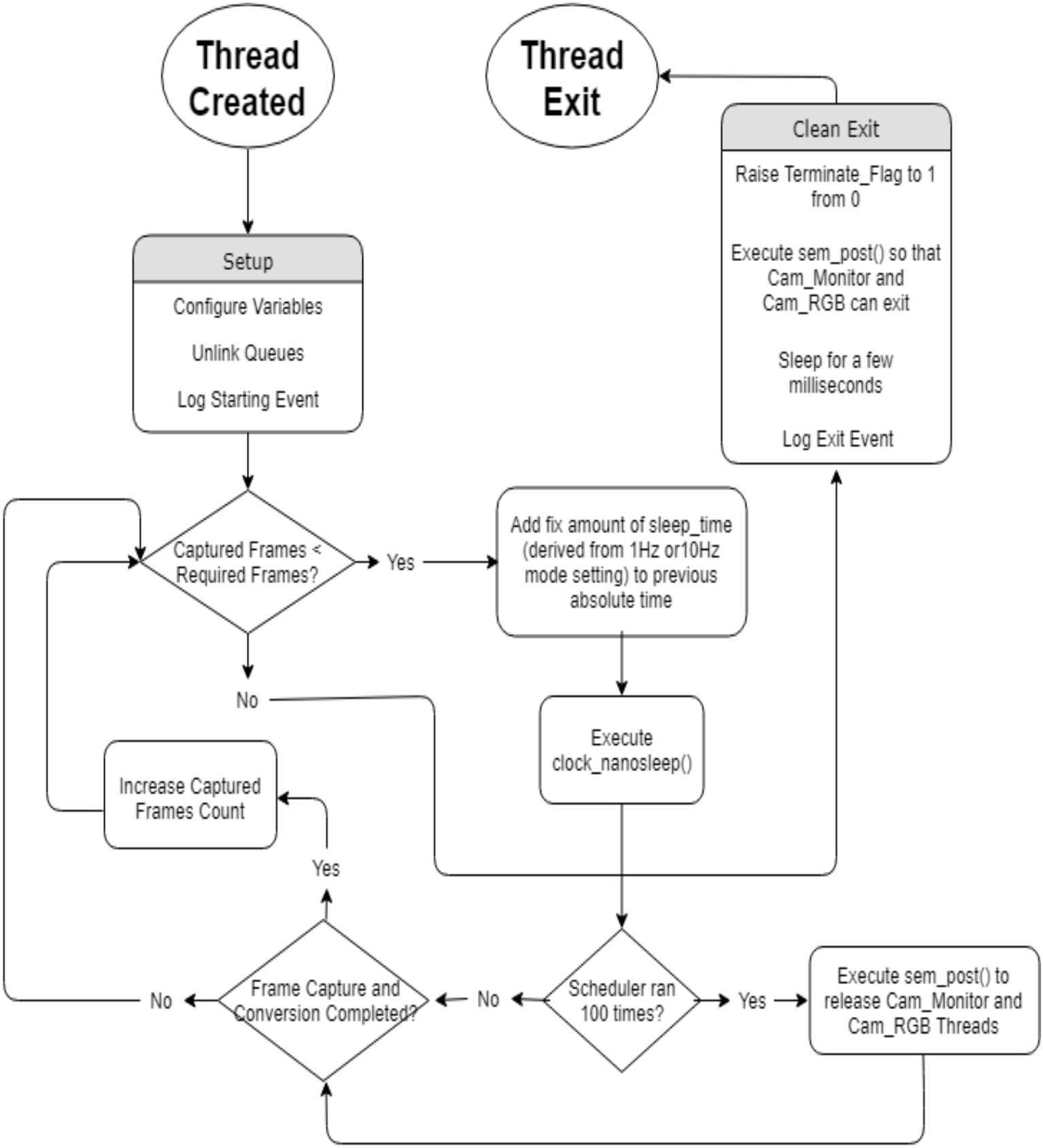
Thread Interaction Diagram



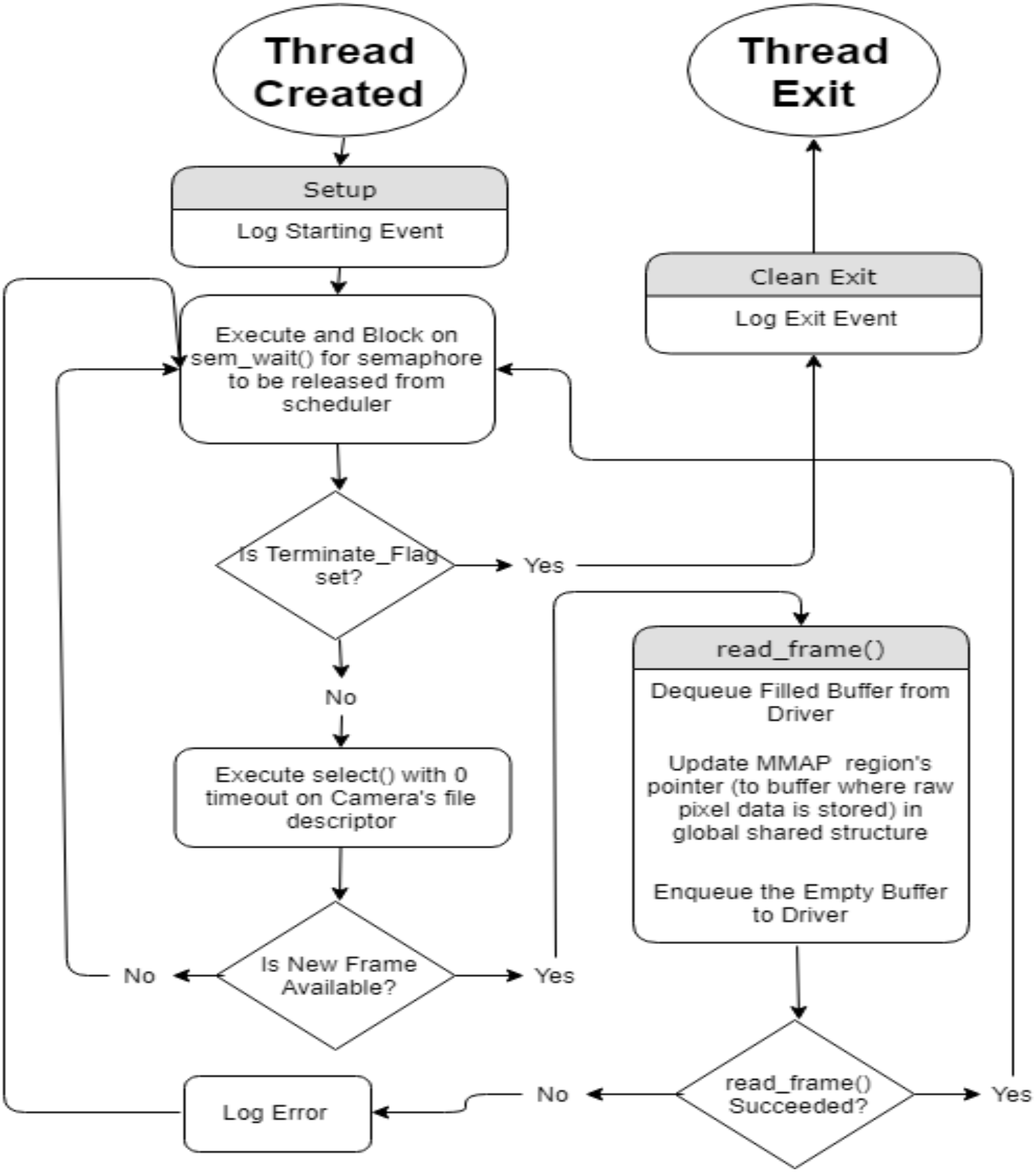
Main Thread Flow Chart



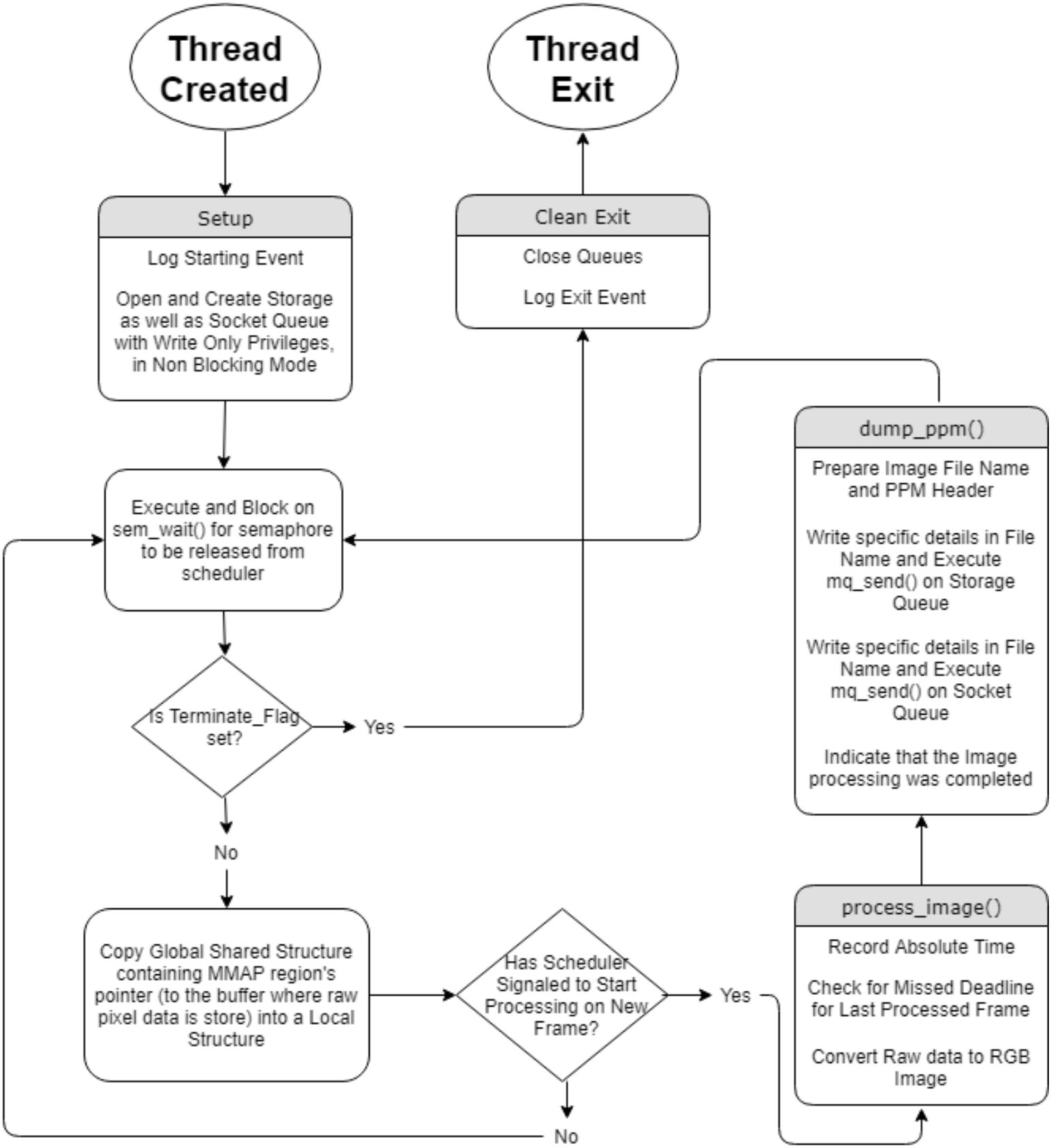
Scheduler Thread Flow Chart



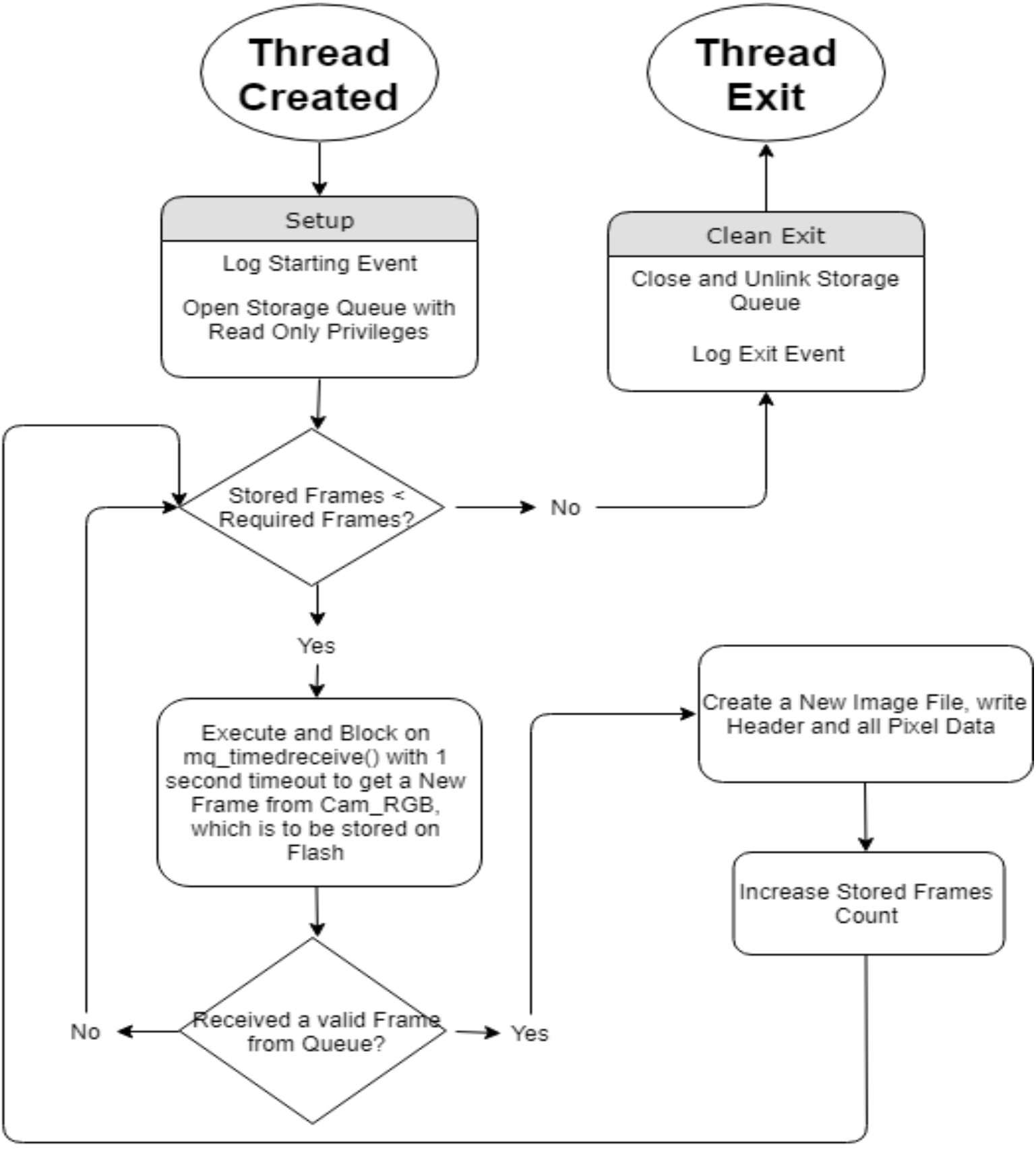
Cam_Monitor Thread Flow Chart



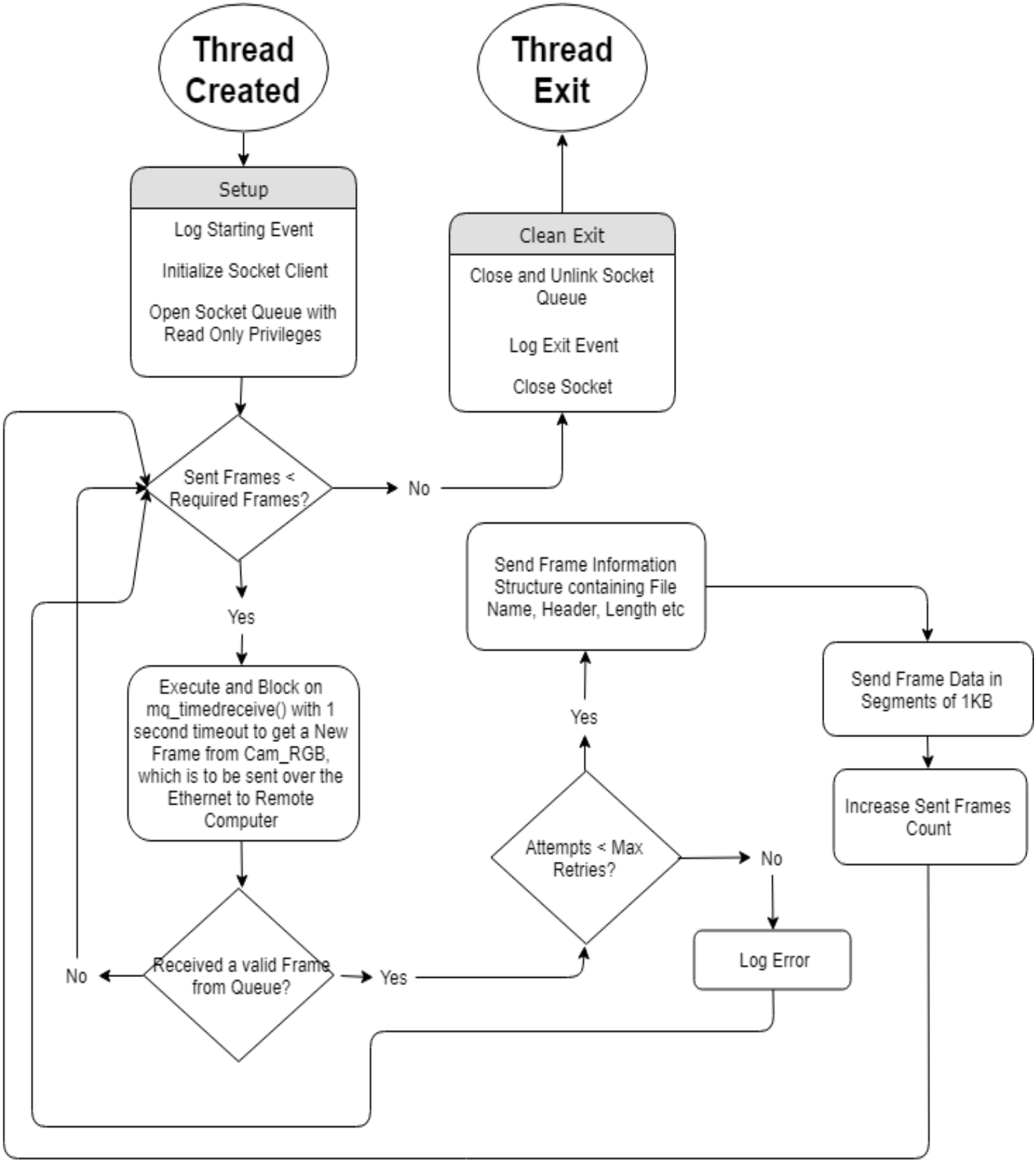
Cam_RGB Thread Flow Chart



Storage Thread Flow Chart



Socket Thread Flow Chart



Q3: Provide all major real-time service requirements with a description of each Si including Ci, Ti, Di with a description of how the request frequency and deadline was determined for each as well as how Ci was determined, estimated or measured as WCET for the service.

Solution:

Real Time Services

The following analysis is based on 10Hz mode. For 1Hz mode, all deadlines will simply be multiplied by the factor of 10.

1. **S_1 – Scheduler:** This is the service which runs 100 times faster than the frequency mode used (runs at 100Hz in case the frames are to be captured at 1Hz, and at 1000Hz in case the frames are to be captured at 10Hz), and is responsible for periodic and timely release of two other services, running under RM scheduling policy.
    - **Priority:** (RT_MAX-1) ~ 98 (in most cases)
    - **C_1:** Negligible (sleeps for the most time)
    - **T_1:** 1ms
    - **D_1:** 1ms
    - **Analysis from 6000+ frame capture Test:**
        - **Average Execution Time (including sleep):** 1.000ms
        - **Worst Case Execution Time (including sleep):** 1.602ms
        - **Overall Jitter/Deviation:** 0.000ms
        - **Average Jitter:** 0.000ms
        - **Absolute Maximum Jitter:** 1.000ms
    - The determination of the deadline/frequency was based on the system design, which favored the scheduler running at a much higher frequency – to provide a robust solution.
    - Capacity was expected to be very low due to a very few lines of code executing in the service, and the execution time (including sleep time) was expected to be much closer to the required.
    - As per the project testing, both are apparently correct.
    - WCET was measured by first filling up a long array of float numbers, with the execution timings of each instance of the Scheduler, and finally the maximum number was found out.

2. **S_2 – Cam_Monitor:** This is the service which is responsible for polling the Camera driver for availability of any new frames, and it runs at 10Hz (100ms), same as the deadline.
    - **Priority:** (RT_MAX-2) ~ 97 (in most cases)
    - **C_2:** 0.062ms
    - **T_2:** 100ms
    - **D_2:** 100ms

- **Analysis from 6000+ frame capture Test:**
  - **Average Execution Time (including sleep):** 0.042ms
  - **Worst Case Execution Time (including sleep):** 0.062ms
  - **Overall Jitter/Deviation:** 218.125ms
  - **Average Jitter:** -99.958ms
  - **Absolute Maximum Jitter:** 100.000ms
- The determination of the deadline/frequency was based on the system design, which simply ran camera polling at the same frequency as of the required FPS. This is done to get just the latest frame in the buffer.
- Capacity was expected to be quite low since the polling was done with 0 timeout (select() call), and interaction with V4L2 API layer isn't significant. This resulted in the entire service getting completed in just about a few microseconds.
- As per the project testing, these assumptions are correct.
- WCET was measured by first filling up a long array of float numbers, with the execution timings of each instance of the Cam_Monitor, and finally the maximum number was found out.

3. **S_3 – Cam_RGB:** This is the service which is responsible for converting raw frame data, to RGB image. It is achieved by processing on every pixel.
   - **Priority:** (RT_MAX-3) ~ 96 (in most cases)
   - **C_3:** 19.542ms
   - **T_3:** 100ms
   - **D_3:** 100ms
   - **Analysis from 6000+ frame capture Test:**
     - **Average Execution Time (including sleep):** 18.292ms
     - **Worst Case Execution Time (including sleep):** 19.542ms
     - **Overall Jitter/Deviation:** 218.250ms
     - **Average Jitter:** -81.708ms
     - **Absolute Maximum Jitter:** 81.797ms
   - The determination of the deadline/frequency was based on the system design, which grabs latest frame from camera using S_2 and thus, this is run after that to convert last acquired frame to RGB.
   - Capacity was expected to be significantly higher – compared to other services, since this service has to process on thousands of pixels for each frame. Therefore, initial estimate was about 50ms, which is not that far away from actual.
   - As per the project testing, these assumptions are correct.
   - WCET was measured by first filling up a long array of float numbers, with the execution timings of each instance of the Cam_Monitor, and finally the maximum number was found out.

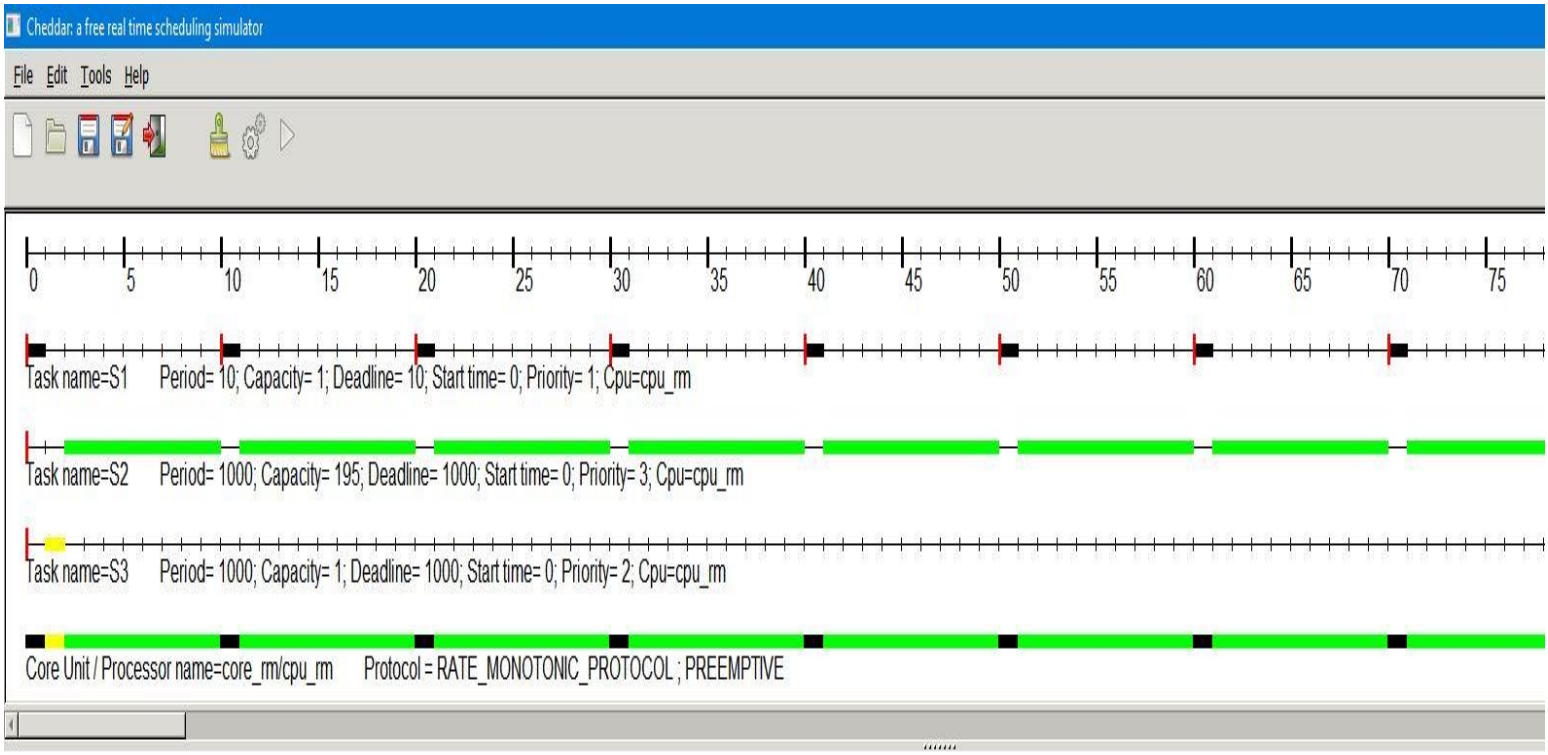
Q4: Provide all major real-time service requirements with a description of each Si including Ci, Ti, Di with a description of how the request frequency and deadline was determined for each as well as how Ci was determined, estimated or measured as WCET for the service.

Solution:

Real Time Analysis with Rate Monotonic Theory

- According to RM Least Upper Bound, having 3 services, if the CPU utilization is less than 77.9%, then the services will definitely be feasible.
- In the present case, total CPU utilization is way below that. Ignoring Scheduler service (since the CPU consumption is extremely low), we have:
  U = ((0.062/100) + (19.542/100)) * 100%
  U = 19.604%
- Even if we consider it to be way higher than this – say – 25%, it still is less than one third of the LUB. Therefore, given that there are no deadlocks or possibility of unbounded priority inversion – the present scenario is easily achievable, as a hard-real time system.
- The only concern could be that probable misbehavior of kernel – which is non preemptable by any of the user space tasks (even FIFO), running on the same CPU core, can throw off the services. However, it hasn't yet been observed – through at least one hundred tests, and thousands of frames captured.
- The I/O is decoupled from the real-time services, and therefore, the system has no problem meeting the deadlines.
- The margin of error is quite low as well, which can be seen from average execution time and WCET.
- This is also evident from Jitter analysis – which can be found **here** and **here**.
- Detailed analysis of execution times, jitter, etc. of each of the services (even the non-real time ones), can be done with the present code, however, will be submitted as an update to this report. That will primarily contain histograms, and probably some further statistical information.
- The screenshots posted in following sections prove that there is not even 1ms of jitter between consecutive frames, nor accumulated jitter causing drift over time.

Cheddar



Scheduling simulation, Processor cpu_rm :
- Number of context switches : 45
- Number of preemptions : 21

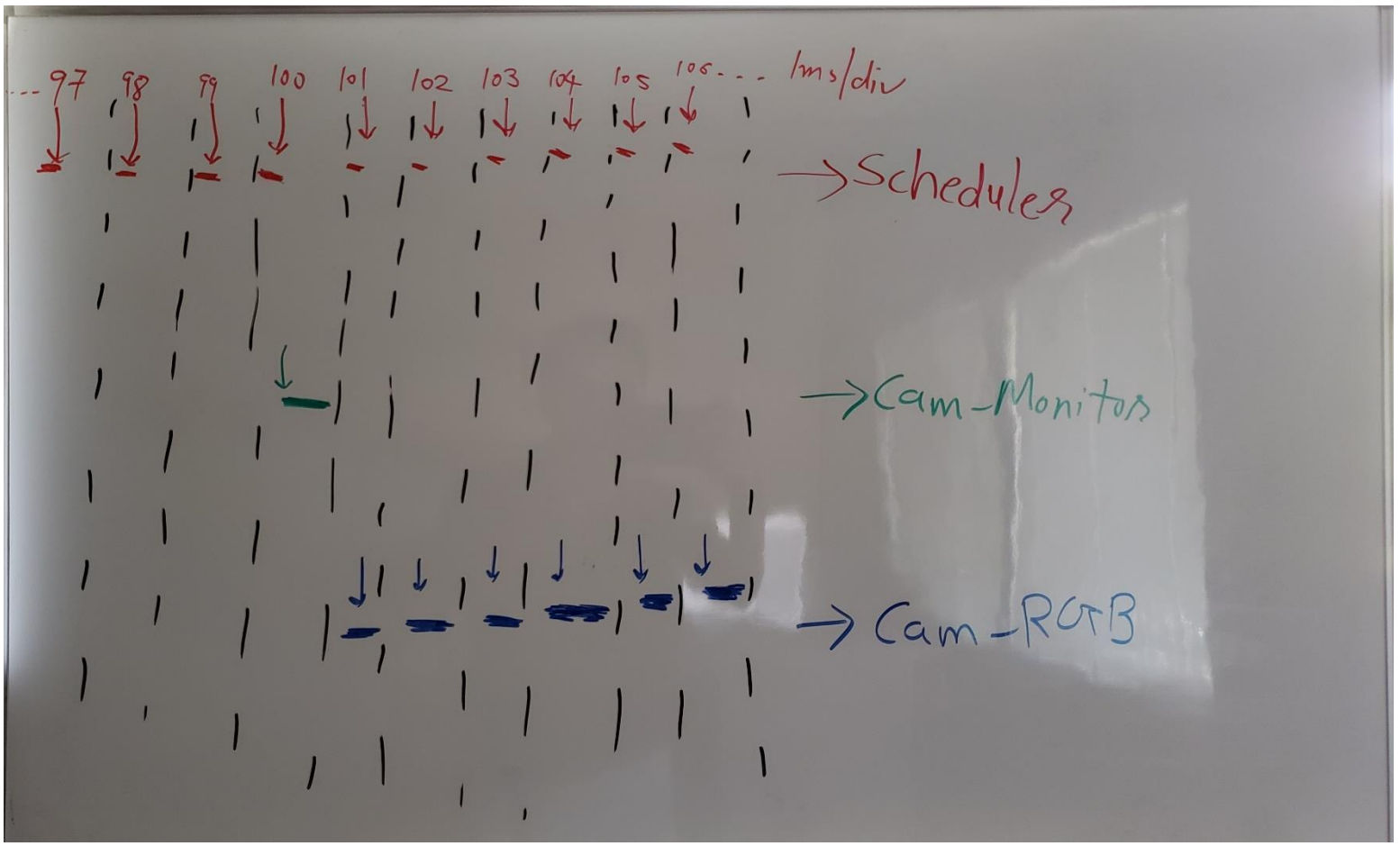
- Task response time computed from simulation :
  S1 => 1/worst
  S2 => 218/worst
  S3 => 2/worst
- No deadline missed in the computed scheduling : the task set is schedulable if you computed the scheduling on the feasibility interval.

Scheduling Point and Completion Time Tests

```
root@poorn-desktop: ~/Workspace/tmp/f_test
root@poorn-desktop:~/Workspace/tmp/f_test# make
gcc -O0 -g   -c feasibility_tests.c
gcc  -O0 -g   -o feasibility_tests feasibility_tests.o -lm
root@poorn-desktop:~/Workspace/tmp/f_test# ./feasibility_tests
******** Completion Test Feasibility Example
Ex-0 U=0.30 (C1=1, C2=1, C3=195; T1=10, T2=1000, T3=1000; T=D): FEASIBLE


******** Scheduling Point Feasibility Example
Ex-0 U=0.30 (C1=1, C2=1, C3=195; T1=10, T2=1000, T3=1000; T=D): FEASIBLE
root@poorn-desktop:~/Workspace/tmp/f_test#
```

Hand Drawn Scheduling



Syslog Screenshots

1Hz

```
Aug  9 21:11:57 poorn-desktop Project_Testing[11429]: <97196.992188ms>!!Socket!! Successfully Sent Frame: 87
Aug  9 21:11:58 poorn-desktop Project_Testing[11429]: <98098.289062ms>!!Cam_Brightness!! Image Processing Start on Frame: 86 of Size: 614400
Aug  9 21:11:58 poorn-desktop Project_Testing[11429]: <98116.968750ms>!!Cam_Brightness!! File Sent to Storage Thread
Aug  9 21:11:58 poorn-desktop Project_Testing[11429]: <98119.453125ms>!!Storage!! Successfully Stored Frame: 88
Aug  9 21:11:58 poorn-desktop Project_Testing[11429]: <98196.601562ms>!!Socket!! Successfully Sent Frame: 88
Aug  9 21:11:59 poorn-desktop Project_Testing[11429]: <99098.289062ms>!!Cam_Brightness!! Image Processing Start on Frame: 87 of Size: 614400
Aug  9 21:11:59 poorn-desktop Project_Testing[11429]: <99117.023438ms>!!Cam_Brightness!! File Sent to Storage Thread
Aug  9 21:11:59 poorn-desktop Project_Testing[11429]: <99119.523438ms>!!Storage!! Successfully Stored Frame: 89
Aug  9 21:11:59 poorn-desktop Project_Testing[11429]: <99196.992188ms>!!Socket!! Successfully Sent Frame: 89
Aug  9 21:12:00 poorn-desktop Project_Testing[11429]: <100098.289062ms>!!Cam_Brightness!! Image Processing Start on Frame: 88 of Size: 614400
Aug  9 21:12:00 poorn-desktop Project_Testing[11429]: <100117.007812ms>!!Cam_Brightness!! File Sent to Storage Thread
Aug  9 21:12:00 poorn-desktop Project_Testing[11429]: <100119.453125ms>!!Storage!! Successfully Stored Frame: 90
Aug  9 21:12:00 poorn-desktop Project_Testing[11429]: <100198.859375ms>!!Socket!! Successfully Sent Frame: 90
Aug  9 21:12:01 poorn-desktop Project_Testing[11429]: <101098.242188ms>!!Cam_Brightness!! Image Processing Start on Frame: 89 of Size: 614400
Aug  9 21:12:01 poorn-desktop Project_Testing[11429]: <101117.125000ms>!!Cam_Brightness!! File Sent to Storage Thread
Aug  9 21:12:01 poorn-desktop Project_Testing[11429]: <101119.695312ms>!!Storage!! Successfully Stored Frame: 91
Aug  9 21:12:01 poorn-desktop Project_Testing[11429]: <101198.148438ms>!!Socket!! Successfully Sent Frame: 91
Aug  9 21:12:02 poorn-desktop Project_Testing[11429]: <102098.257812ms>!!Cam_Brightness!! Image Processing Start on Frame: 90 of Size: 614400
Aug  9 21:12:02 poorn-desktop Project_Testing[11429]: <102117.117188ms>!!Cam_Brightness!! File Sent to Storage Thread
Aug  9 21:12:02 poorn-desktop Project_Testing[11429]: <102119.617188ms>!!Storage!! Successfully Stored Frame: 92
Aug  9 21:12:02 poorn-desktop Project_Testing[11429]: <102196.734375ms>!!Socket!! Successfully Sent Frame: 92
Aug  9 21:12:03 poorn-desktop Project_Testing[11429]: <103098.250000ms>!!Cam_Brightness!! Image Processing Start on Frame: 91 of Size: 614400
Aug  9 21:12:03 poorn-desktop Project_Testing[11429]: <103116.960938ms>!!Cam_Brightness!! File Sent to Storage Thread
Aug  9 21:12:03 poorn-desktop Project_Testing[11429]: <103119.476562ms>!!Storage!! Successfully Stored Frame: 93
Aug  9 21:12:03 poorn-desktop Project_Testing[11429]: <103196.703125ms>!!Socket!! Successfully Sent Frame: 93
Aug  9 21:12:04 poorn-desktop Project_Testing[11429]: <104098.289062ms>!!Cam_Brightness!! Image Processing Start on Frame: 92 of Size: 614400
Aug  9 21:12:04 poorn-desktop Project_Testing[11429]: <104117.078125ms>!!Cam_Brightness!! File Sent to Storage Thread
Aug  9 21:12:04 poorn-desktop Project_Testing[11429]: <104119.507812ms>!!Storage!! Successfully Stored Frame: 94
Aug  9 21:12:04 poorn-desktop Project_Testing[11429]: <104196.804688ms>!!Socket!! Successfully Sent Frame: 94
Aug  9 21:12:05 poorn-desktop Project_Testing[11429]: <105098.234375ms>!!Cam_Brightness!! Image Processing Start on Frame: 93 of Size: 614400
Aug  9 21:12:05 poorn-desktop Project_Testing[11429]: <105117.015625ms>!!Cam_Brightness!! File Sent to Storage Thread
Aug  9 21:12:05 poorn-desktop Project_Testing[11429]: <105119.523438ms>!!Storage!! Successfully Stored Frame: 95
Aug  9 21:12:05 poorn-desktop Project_Testing[11429]: <105196.656250ms>!!Socket!! Successfully Sent Frame: 95
Aug  9 21:12:06 poorn-desktop Project_Testing[11429]: <106098.289062ms>!!Cam_Brightness!! Image Processing Start on Frame: 94 of Size: 614400
Aug  9 21:12:06 poorn-desktop Project_Testing[11429]: <106117.015625ms>!!Cam_Brightness!! File Sent to Storage Thread
Aug  9 21:12:06 poorn-desktop Project_Testing[11429]: <106119.492188ms>!!Storage!! Successfully Stored Frame: 96
Aug  9 21:12:06 poorn-desktop Project_Testing[11429]: <106201.703125ms>!!Socket!! Successfully Sent Frame: 96
Aug  9 21:12:07 poorn-desktop Project_Testing[11429]: <107098.289062ms>!!Cam_Brightness!! Image Processing Start on Frame: 95 of Size: 614400
Aug  9 21:12:07 poorn-desktop Project_Testing[11429]: <107117.023438ms>!!Cam_Brightness!! File Sent to Storage Thread
Aug  9 21:12:07 poorn-desktop Project_Testing[11429]: <107119.570312ms>!!Storage!! Successfully Stored Frame: 97
Aug  9 21:12:07 poorn-desktop Project_Testing[11429]: <107196.781250ms>!!Socket!! Successfully Sent Frame: 97
Aug  9 21:12:08 poorn-desktop Project_Testing[11429]: <108098.250000ms>!!Cam_Brightness!! Image Processing Start on Frame: 96 of Size: 614400
Aug  9 21:12:08 poorn-desktop Project_Testing[11429]: <108117.101562ms>!!Cam_Brightness!! File Sent to Storage Thread
Aug  9 21:12:08 poorn-desktop Project_Testing[11429]: <108119.687500ms>!!Storage!! Successfully Stored Frame: 98
Aug  9 21:12:08 poorn-desktop Project_Testing[11429]: <108196.882812ms>!!Socket!! Successfully Sent Frame: 98
Aug  9 21:12:09 poorn-desktop Project_Testing[11429]: <109098.289062ms>!!Cam_Brightness!! Image Processing Start on Frame: 97 of Size: 614400
Aug  9 21:12:09 poorn-desktop Project_Testing[11429]: <109117.085938ms>!!Cam_Brightness!! File Sent to Storage Thread
Aug  9 21:12:09 poorn-desktop Project_Testing[11429]: <109119.601562ms>!!Storage!! Successfully Stored Frame: 99
Aug  9 21:12:09 poorn-desktop Project_Testing[11429]: <109197.601562ms>!!Socket!! Successfully Sent Frame: 99
Aug  9 21:12:10 poorn-desktop Project_Testing[11429]: <110098.320312ms>!!Cam_Brightness!! Image Processing Start on Frame: 98 of Size: 614400
Aug  9 21:12:10 poorn-desktop Project_Testing[11429]: <110116.429688ms>!!Cam_Brightness!! File Sent to Storage Thread
Aug  9 21:12:10 poorn-desktop Project_Testing[11429]: <110118.906250ms>!!Storage!! Successfully Stored Frame: 100
Aug  9 21:12:10 poorn-desktop Project_Testing[11429]: <110196.015625ms>!!Socket!! Successfully Sent Frame: 100
Aug  9 21:12:11 poorn-desktop Project_Testing[11429]: <111098.296875ms>!!Cam_Brightness!! Image Processing Start on Frame: 99 of Size: 614400
Aug  9 21:12:11 poorn-desktop Project_Testing[11429]: <111116.312500ms>!!Cam_Brightness!! File Sent to Storage Thread
Aug  9 21:12:11 poorn-desktop Project_Testing[11429]: <111118.671875ms>!!Storage!! Successfully Stored Frame: 101
Aug  9 21:12:11 poorn-desktop Project_Testing[11429]: <111195.914062ms>!!Socket!! Successfully Sent Frame: 101
Aug  9 21:12:12 poorn-desktop Project_Testing[11429]: <112098.273438ms>!!Cam_Brightness!! Image Processing Start on Frame: 100 of Size: 614400
```

10Hz

```
Aug  9 21:05:18 poorn-desktop Project_Testing[11141]: <5475.332520ms>!!Socket!! Successfully Sent Frame: 7
Aug  9 21:05:18 poorn-desktop Project_Testing[11141]: <5476.645508ms>!!Cam_Brightness!! Image Processing Start on Frame: 6 of Size: 614400
Aug  9 21:05:18 poorn-desktop Project_Testing[11141]: <5495.766113ms>!!Cam_Brightness!! File Sent to Storage Thread
Aug  9 21:05:18 poorn-desktop Project_Testing[11141]: <5498.330078ms>!!Storage!! Successfully Stored Frame: 8
Aug  9 21:05:18 poorn-desktop Project_Testing[11141]: <5575.513672ms>!!Socket!! Successfully Sent Frame: 8
Aug  9 21:05:18 poorn-desktop Project_Testing[11141]: <5576.643555ms>!!Cam_Brightness!! Image Processing Start on Frame: 7 of Size: 614400
Aug  9 21:05:18 poorn-desktop Project_Testing[11141]: <5595.613281ms>!!Cam_Brightness!! File Sent to Storage Thread
Aug  9 21:05:18 poorn-desktop Project_Testing[11141]: <5598.295410ms>!!Storage!! Successfully Stored Frame: 9
Aug  9 21:05:18 poorn-desktop Project_Testing[11141]: <5675.333984ms>!!Socket!! Successfully Sent Frame: 9
Aug  9 21:05:18 poorn-desktop Project_Testing[11141]: <5676.652344ms>!!Cam_Brightness!! Image Processing Start on Frame: 8 of Size: 614400
Aug  9 21:05:18 poorn-desktop Project_Testing[11141]: <5695.828125ms>!!Cam_Brightness!! File Sent to Storage Thread
Aug  9 21:05:18 poorn-desktop Project_Testing[11141]: <5698.463867ms>!!Storage!! Successfully Stored Frame: 10
Aug  9 21:05:19 poorn-desktop Project_Testing[11141]: <5775.546387ms>!!Socket!! Successfully Sent Frame: 10
Aug  9 21:05:19 poorn-desktop Project_Testing[11141]: <5776.630859ms>!!Cam_Brightness!! Image Processing Start on Frame: 9 of Size: 614400
Aug  9 21:05:19 poorn-desktop Project_Testing[11141]: <5795.628906ms>!!Cam_Brightness!! File Sent to Storage Thread
Aug  9 21:05:19 poorn-desktop Project_Testing[11141]: <5798.303711ms>!!Storage!! Successfully Stored Frame: 11
Aug  9 21:05:19 poorn-desktop Project_Testing[11141]: <5875.260742ms>!!Socket!! Successfully Sent Frame: 11
Aug  9 21:05:19 poorn-desktop Project_Testing[11141]: <5876.647461ms>!!Cam_Brightness!! Image Processing Start on Frame: 10 of Size: 614400
Aug  9 21:05:19 poorn-desktop Project_Testing[11141]: <5895.835449ms>!!Cam_Brightness!! File Sent to Storage Thread
Aug  9 21:05:19 poorn-desktop Project_Testing[11141]: <5898.447266ms>!!Storage!! Successfully Stored Frame: 12
Aug  9 21:05:19 poorn-desktop Project_Testing[11141]: <5975.545898ms>!!Socket!! Successfully Sent Frame: 12
Aug  9 21:05:19 poorn-desktop Project_Testing[11141]: <5976.643066ms>!!Cam_Brightness!! Image Processing Start on Frame: 11 of Size: 614400
Aug  9 21:05:19 poorn-desktop Project_Testing[11141]: <5995.834961ms>!!Cam_Brightness!! File Sent to Storage Thread
Aug  9 21:05:19 poorn-desktop Project_Testing[11141]: <5998.492188ms>!!Storage!! Successfully Stored Frame: 13
Aug  9 21:05:19 poorn-desktop Project_Testing[11141]: <6075.524414ms>!!Socket!! Successfully Sent Frame: 13
Aug  9 21:05:19 poorn-desktop Project_Testing[11141]: <6076.633301ms>!!Cam_Brightness!! Image Processing Start on Frame: 12 of Size: 614400
Aug  9 21:05:19 poorn-desktop Project_Testing[11141]: <6095.655762ms>!!Cam_Brightness!! File Sent to Storage Thread
Aug  9 21:05:19 poorn-desktop Project_Testing[11141]: <6098.269043ms>!!Storage!! Successfully Stored Frame: 14
Aug  9 21:05:19 poorn-desktop Project_Testing[11141]: <6175.336426ms>!!Socket!! Successfully Sent Frame: 14
Aug  9 21:05:19 poorn-desktop Project_Testing[11141]: <6176.639160ms>!!Cam_Brightness!! Image Processing Start on Frame: 13 of Size: 614400
Aug  9 21:05:19 poorn-desktop Project_Testing[11141]: <6195.786133ms>!!Cam_Brightness!! File Sent to Storage Thread
Aug  9 21:05:19 poorn-desktop Project_Testing[11141]: <6198.379395ms>!!Storage!! Successfully Stored Frame: 15
Aug  9 21:05:19 poorn-desktop Project_Testing[11141]: <6275.502930ms>!!Socket!! Successfully Sent Frame: 15
Aug  9 21:05:19 poorn-desktop Project_Testing[11141]: <6276.629395ms>!!Cam_Brightness!! Image Processing Start on Frame: 14 of Size: 614400
Aug  9 21:05:19 poorn-desktop Project_Testing[11141]: <6295.602539ms>!!Cam_Brightness!! File Sent to Storage Thread
Aug  9 21:05:19 poorn-desktop Project_Testing[11141]: <6298.213379ms>!!Storage!! Successfully Stored Frame: 16
Aug  9 21:05:19 poorn-desktop Project_Testing[11141]: <6375.224121ms>!!Socket!! Successfully Sent Frame: 16
Aug  9 21:05:19 poorn-desktop Project_Testing[11141]: <6376.642578ms>!!Cam_Brightness!! Image Processing Start on Frame: 15 of Size: 614400
Aug  9 21:05:19 poorn-desktop Project_Testing[11141]: <6395.835449ms>!!Cam_Brightness!! File Sent to Storage Thread
Aug  9 21:05:19 poorn-desktop Project_Testing[11141]: <6398.456055ms>!!Storage!! Successfully Stored Frame: 17
Aug  9 21:05:19 poorn-desktop Project_Testing[11141]: <6475.638672ms>!!Socket!! Successfully Sent Frame: 17
Aug  9 21:05:19 poorn-desktop Project_Testing[11141]: <6476.638184ms>!!Cam_Brightness!! Image Processing Start on Frame: 16 of Size: 614400
Aug  9 21:05:19 poorn-desktop Project_Testing[11141]: <6495.759277ms>!!Cam_Brightness!! File Sent to Storage Thread
Aug  9 21:05:19 poorn-desktop Project_Testing[11141]: <6498.293945ms>!!Storage!! Successfully Stored Frame: 18
Aug  9 21:05:19 poorn-desktop Project_Testing[11141]: <6575.309570ms>!!Socket!! Successfully Sent Frame: 18
Aug  9 21:05:19 poorn-desktop Project_Testing[11141]: <6576.648438ms>!!Cam_Brightness!! Image Processing Start on Frame: 17 of Size: 614400
Aug  9 21:05:19 poorn-desktop Project_Testing[11141]: <6595.670898ms>!!Cam_Brightness!! File Sent to Storage Thread
Aug  9 21:05:19 poorn-desktop Project_Testing[11141]: <6598.218750ms>!!Storage!! Successfully Stored Frame: 19
Aug  9 21:05:19 poorn-desktop Project_Testing[11141]: <6675.234375ms>!!Socket!! Successfully Sent Frame: 19
Aug  9 21:05:19 poorn-desktop Project_Testing[11141]: <6676.643555ms>!!Cam_Brightness!! Image Processing Start on Frame: 18 of Size: 614400
Aug  9 21:05:19 poorn-desktop Project_Testing[11141]: <6695.852539ms>!!Cam_Brightness!! File Sent to Storage Thread
Aug  9 21:05:19 poorn-desktop Project_Testing[11141]: <6698.399902ms>!!Storage!! Successfully Stored Frame: 20
Aug  9 21:05:20 poorn-desktop Project_Testing[11141]: <6775.537109ms>!!Socket!! Successfully Sent Frame: 20
Aug  9 21:05:20 poorn-desktop Project_Testing[11141]: <6776.639648ms>!!Cam_Brightness!! Image Processing Start on Frame: 19 of Size: 614400
Aug  9 21:05:20 poorn-desktop Project_Testing[11141]: <6795.851074ms>!!Cam_Brightness!! File Sent to Storage Thread
Aug  9 21:05:20 poorn-desktop Project_Testing[11141]: <6798.420898ms>!!Storage!! Successfully Stored Frame: 21
Aug  9 21:05:20 poorn-desktop Project_Testing[11141]: <6875.547852ms>!!Socket!! Successfully Sent Frame: 21
Aug  9 21:05:20 poorn-desktop Project_Testing[11141]: <6876.642578ms>!!Cam_Brightness!! Image Processing Start on Frame: 20 of Size: 614400
```

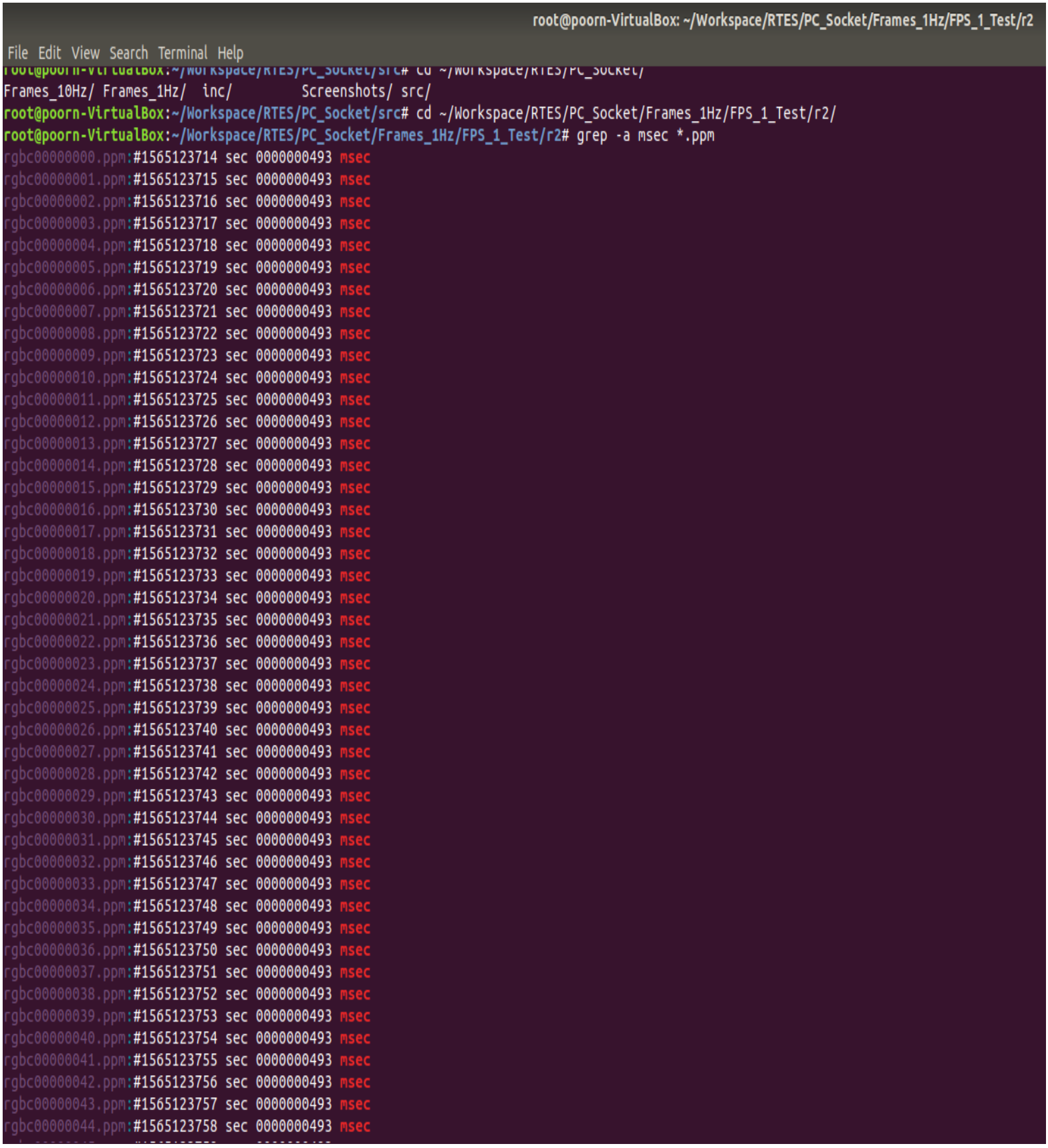
Q5: Analyze your system in terms of how well it meets real-time requirements as outlined and used in Lab Exercise #5, found here. Present the results along with a description of how predictable the responses are relative to the request times as well as how constant the request frequency is for your system design.

Solution:

System Analysis based on Tests, and Verification Outputs

- The designed system has found out to be working almost perfectly, under all the varying conditions.
- The only thing that can use some improvement – is addition of frame filtering either at startup/runtime in 10Hz mode. Currently none is there, and approximately 2% of all captured frames, have a blurred $1/10^{th}$ second indicator's digit.
- The system has 0 millisecond of drift over long period of times (tested for 2 hours under 1Hz mode) or after capturing thousands of frames (10000 frames in 10Hz).
- The difference between two consecutive frames, exactly meets the deadline (ignoring a few microseconds), which is evident from both – syslog outputs as well as, timestamps embedded in frame headers.
- At 1Hz mode, the system has never failed to detect the change in position of the second indicator – in an analog clock (ticking version), at the startup, while executing Cam_Filter() function. This then provides a solid base for all future locked frame captures, which never results in a single blurry frame even.
- The Storage thread, which runs in best efforts – has been never observed to fail either. Nor does it lag so much that circular buffer of length 100 is overflowed.
- The same observation has also been made for Socket thread as well. Sending out Gigabytes of data over the ethernet using only C, was somewhat intimidating at first, however, in-depth understanding of the APIs, TCP Socket Connection, along with a good algorithm to manually segment the large image into smaller pieces of 1KB each – worked out very well.
- All of the behavior described above, are constant, regardless of the operating mode, configuration of socket (on or off), number of processes running in the background, lighting conditions surrounding the camera etc.
- Therefore, I would like to claim that the system developed is reasonably reliable.
- Please find the screenshots supporting these claims below, as well as all frames and video **here** and **here**.

Initial Frames @ 1Hz



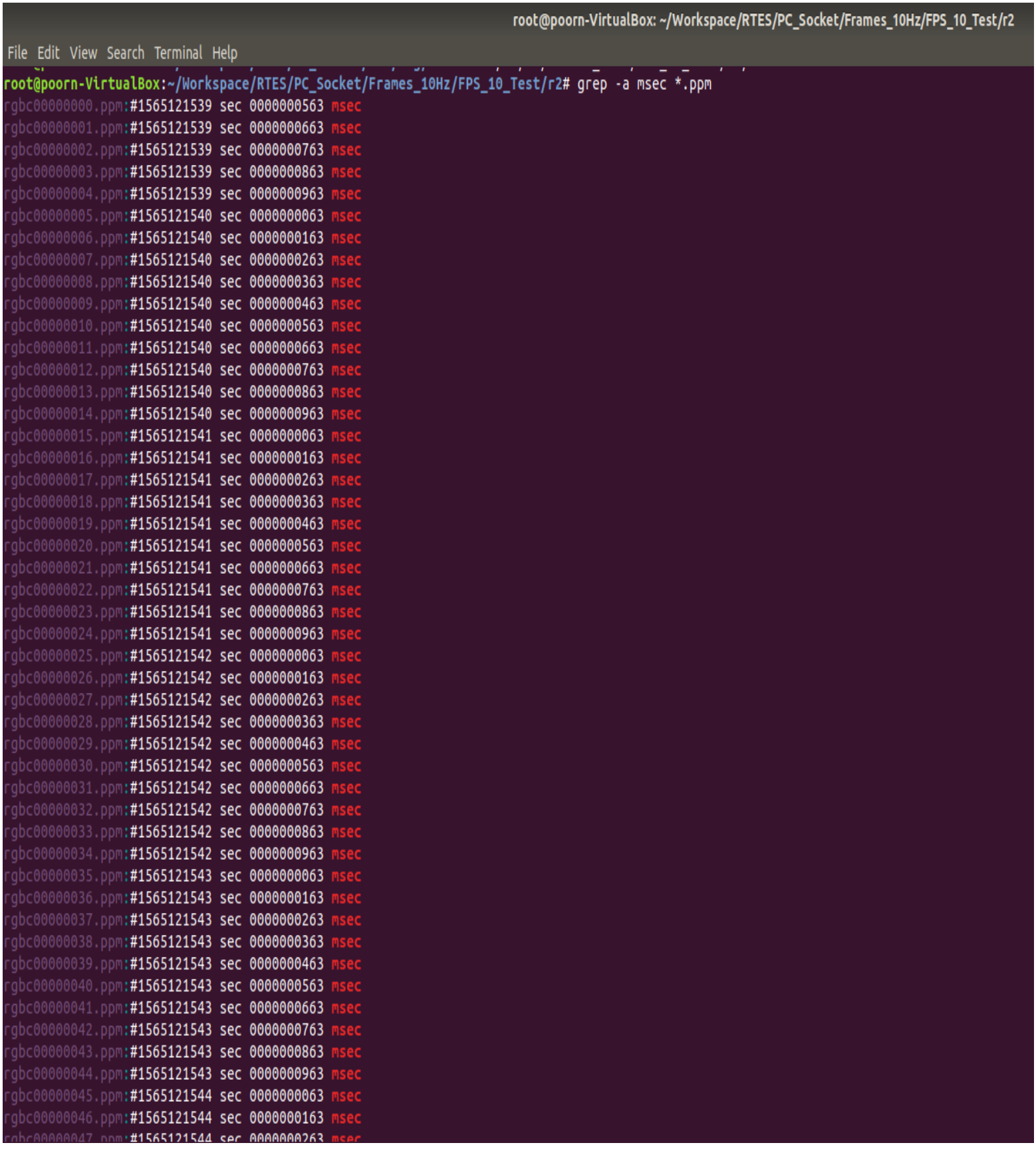
Last Frames @ 1Hz

root@poorn-VirtualBox: ~/Workspace/RTES/PC_Socket/Frames_1Hz/FPS_1_Test/r2

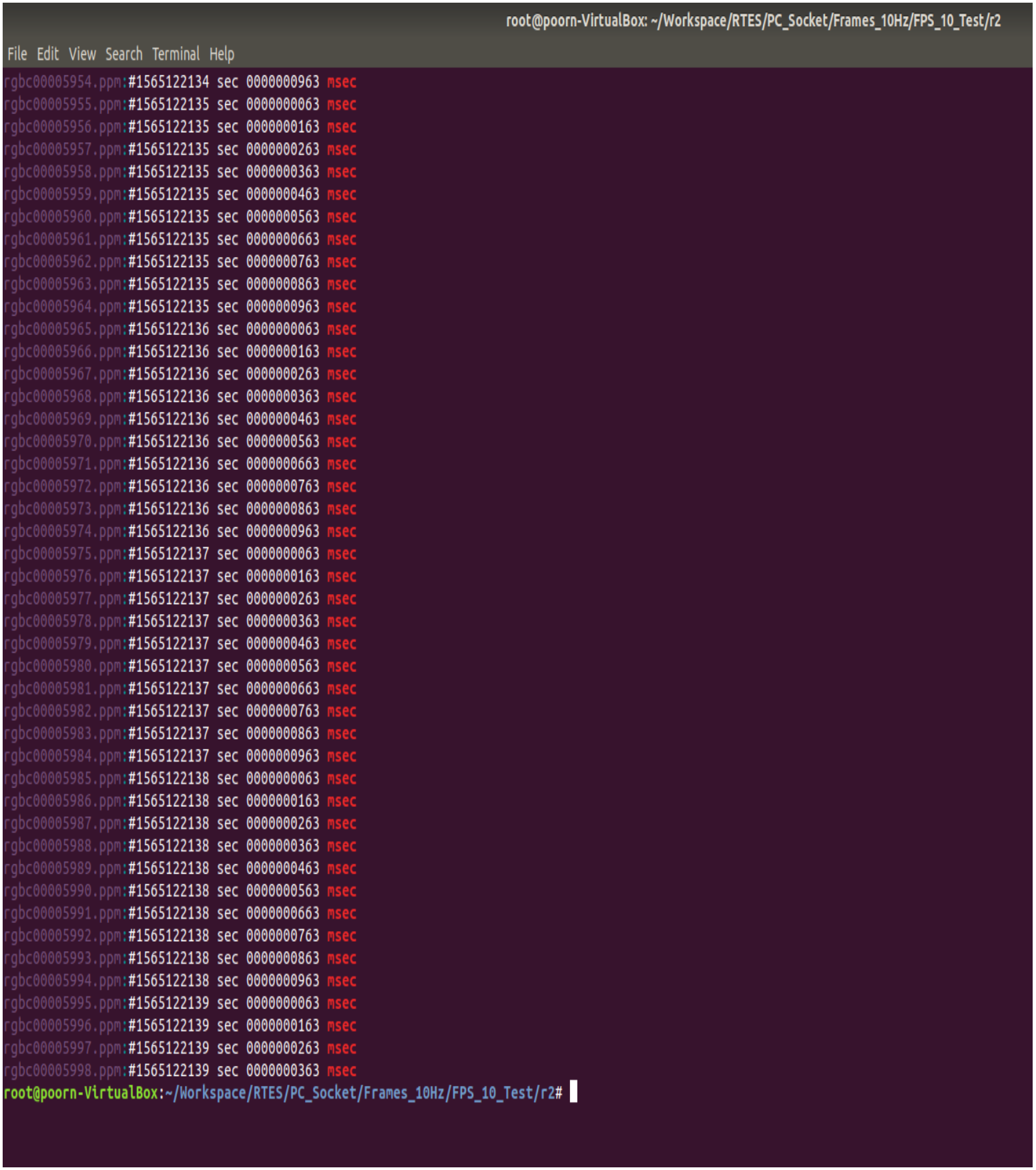
File Edit View Search Terminal Help

```
rgbc00003554.ppm:#1565127268 sec 0000000493 msec
rgbc00003555.ppm:#1565127269 sec 0000000493 msec
rgbc00003556.ppm:#1565127270 sec 0000000493 msec
rgbc00003557.ppm:#1565127271 sec 0000000493 msec
rgbc00003558.ppm:#1565127272 sec 0000000493 msec
rgbc00003559.ppm:#1565127273 sec 0000000493 msec
rgbc00003560.ppm:#1565127274 sec 0000000493 msec
rgbc00003561.ppm:#1565127275 sec 0000000493 msec
rgbc00003562.ppm:#1565127276 sec 0000000493 msec
rgbc00003563.ppm:#1565127277 sec 0000000493 msec
rgbc00003564.ppm:#1565127278 sec 0000000493 msec
rgbc00003565.ppm:#1565127279 sec 0000000493 msec
rgbc00003566.ppm:#1565127280 sec 0000000493 msec
rgbc00003567.ppm:#1565127281 sec 0000000493 msec
rgbc00003568.ppm:#1565127282 sec 0000000493 msec
rgbc00003569.ppm:#1565127283 sec 0000000493 msec
rgbc00003570.ppm:#1565127284 sec 0000000493 msec
rgbc00003571.ppm:#1565127285 sec 0000000493 msec
rgbc00003572.ppm:#1565127286 sec 0000000493 msec
rgbc00003573.ppm:#1565127287 sec 0000000493 msec
rgbc00003574.ppm:#1565127288 sec 0000000493 msec
rgbc00003575.ppm:#1565127289 sec 0000000493 msec
rgbc00003576.ppm:#1565127290 sec 0000000493 msec
rgbc00003577.ppm:#1565127291 sec 0000000493 msec
rgbc00003578.ppm:#1565127292 sec 0000000493 msec
rgbc00003579.ppm:#1565127293 sec 0000000493 msec
rgbc00003580.ppm:#1565127294 sec 0000000493 msec
rgbc00003581.ppm:#1565127295 sec 0000000493 msec
rgbc00003582.ppm:#1565127296 sec 0000000493 msec
rgbc00003583.ppm:#1565127297 sec 0000000493 msec
rgbc00003584.ppm:#1565127298 sec 0000000493 msec
rgbc00003585.ppm:#1565127299 sec 0000000493 msec
rgbc00003586.ppm:#1565127300 sec 0000000493 msec
rgbc00003587.ppm:#1565127301 sec 0000000493 msec
rgbc00003588.ppm:#1565127302 sec 0000000493 msec
rgbc00003589.ppm:#1565127303 sec 0000000493 msec
rgbc00003590.ppm:#1565127304 sec 0000000493 msec
rgbc00003591.ppm:#1565127305 sec 0000000493 msec
rgbc00003592.ppm:#1565127306 sec 0000000493 msec
rgbc00003593.ppm:#1565127307 sec 0000000493 msec
rgbc00003594.ppm:#1565127308 sec 0000000493 msec
rgbc00003595.ppm:#1565127309 sec 0000000493 msec
rgbc00003596.ppm:#1565127310 sec 0000000493 msec
rgbc00003597.ppm:#1565127311 sec 0000000493 msec
rgbc00003598.ppm:#1565127312 sec 0000000493 msec
rgbc00003599.ppm:#1565127313 sec 0000000493 msec
rgbc00003600.ppm:#1565127314 sec 0000000493 msec
root@poorn-VirtualBox:~/Workspace/RTES/PC_Socket/Frames_1Hz/FPS_1_Test/r2#
```

Initial Frames @ 10Hz



Last Frames @ 10Hz



Q6 – Report, Attached Separately.