Exercise #5 – Real-Time Continuous Media Systems

1) Research, identify and briefly describe the 3 worst software and hardware computer engineering design flaws and/or software defects of all time.

a) Some options for product failures are Blackberry Storm, Windows Genuine Advantage, Windows 8, Windows ME, Apple Lisa, Pentium FPU bug

b) Some options for mission failures include: NORAD false alarms, Mars Express Beagle 2, Challenger and Columbia Shuttle Loss, and Boeing MCAS design flaws more recently corrected summarized in this documentary and fully analyzed.

Pentium FDIV bug:

The Pentium FDIV bug was caused by an error in a lookup table that was part of the SRT algorithm which was implemented to be a faster way of calculating floating-point division. The hardware bug affected the FPU of the early intel Pentium processors. The SRT algorithm is implemented as a programmable logic array with 2048 cells of which 1066 should have been populated with one of the 5 values: -2, -1, 0, +1, +2. However, on the buggy chips, 5 cells that should have returned the value +2 were returning 0 instead. Though it could be argued that the errors seen would occur rarely and result in only small deviations from the correct output values, it was observed that these errors can accumulate repeatedly owning to the recursive nature of the SRT algorithm. In some extreme cases, the error could reach the 4^{th} significant digit of the result. The most common example where this was observed is by dividing 4,195,835 by 3,145,727: the converted hexadecimal numbers would trigger access to the 'empty' array cells. Thus, the bug can be concluded to have the following characteristics:

1. Error can occur in any of the 3 operating precisions: single, double, or extended for divide instruction
2. The error occurrence and degree of it is highly depended on the input data

This bug also gave rise to the field of formal verification as the traditional method was performed via partial testing as exhaustive testing of 64-bit microprocessor would require a lot of computation time. Formal verification aims to proof that a given hardware design of a chip matches the original logic design.

Boeing MCAS:

The Boeing MCAS design led to two tragic planes crashes: Lion Air flight 610 and Ethiopian Airlines Flight 302. The Manoeuvring Characteristics Augmentation system (MCAS) is a flight control system implemented in the 737 MAX planes. There were many problems in the overall system failure that led to these crashes.

The MCAS software was implemented to compensate for changes in size and placement of engines, mainly to keep up with the innovations of Airbus. The repositioning of the engines in the flights designed same as 737 family significantly changed the aerodynamics of the aircraft and created the possibility of nose-up stall. When the Angle-of-attack sensor was triggered, the MCAS system would detect this condition and control the horizontal stabilizer. This physically moves the pilot's control column forward and engages the airplanes' elevators. Relying on only one AOA sensor in the design essentially led to a SPOF (single point of failure). The aircraft was also more susceptible to the erroneous sensor readings. In both the crashes, the sensor malfunctioned and repeatedly activated the MCAS without letting any system override by the pilots. Furthermore, in the later designs of two sensors model aircrafts, the 'sensor disagree' indicator light was implemented as an optional equipment package. Thus, mainly the issues were: underestimation of the power of MCAS to move the plane's horizontal tail and thus difficult it would be for pilots to maintain control of the aircraft,

system did not account for MCAS deploying multiple times and lastly, placing a design constraint on a sensor input which did not have built-in redundancy.

Columbia shuttle Loss:

The Columbia Space shuttle disaster occurred during the re-entry phases of the STS-107 mission. Initially there was a loss of temperature readings from the left wing and then on the orbiter, there was a loss of left main landing gear pressure readings. During the launch of the shuttle, the external tank had shed a large piece of foam insulation, which struck the orbiter's left wing, damaging its thermal protection system. The super-heated air would rush through the broken left wing, cutting through the sensor wiring and burning into the left landing gear wheel well. The wing would eventually fold over or break away from the fuselage, throwing the ship out of control. Occurrences of foam shedding were well known before the disaster but the foam strikes were accepted as the flight missions were successful. The risk analysis of the shuttle was influenced by this deeming of the shuttle 'operational' even when the flight did not perform as expected and by accepting risks inherent in anomalous performance. Further, the shuttle was not equipped with robot arm, tools or materials to repair major heat shield damage when recognised on-board. During the launch, enhanced real-time imaging capabilities could have provided more detailed information and the steps to rectify the destruction or contingency plan to rescue the crew could have been developed.

c) State why the 3 you have selected are the worst in terms of negative impact, negligence, and bad decisions made, but why the failure was not really a real-time or interactive systems design flaw. Rank them from worst to least-worst.

All the 3 incidents involve some degree of negligence and corrupted management and none of the failures were inherently a real-time flaw. Ranking them in the worst to least-worst order:

- The MCAS software system was developed to offset the structural differences observed in the aircraft after heavier, more fuel-efficient engines were implemented. The flawed AOA sensor with no redundancy, would read erroneous values during the testing stage itself. Further, this would cause the MCAS system to take control of the aircraft overriding the pilot's observation. This is flawed engineering design. In the negligence aspect of the case, the pilots were not provided with the documentation on MCAS or training with the system to reduce costs and inspection time. The pilots associated with the 737 MAX aircraft would first come to learn of the MCAS system after the Lion Air flight crash. After the initial crash, there was a two-sensor redundancy system implemented as an optional package which the Ethiopian Airlines did not avail.
- The Columbia space shuttle loss could have been prevented if proper risk analysis had been conducted and if proper contingency plans were in place. NASA management knew of the foam shedding and if proper attention was given to the engineers who had reported the impact of this design flaw, the thermal protection system could have been redesigned. But the issues were swept under the rug to maintain the timeline to meet once-per-week operational launches. This meant developing 50 shuttle payloads every year.
- The intel Pentium chips manufactured with a clock speed of less than 120Mhz is most likely to be susceptible to this bug. When Dr. Nicely first suspected the Pentium chips to be the source of the error in his research after crossing out other options, it was discovered that Intel has already known of the error at this point but decided to keep quiet. The bug was fixed on newer chips but intel went on to devise a software solution for the bugged chips. With the threat of a class action lawsuit, Intel agreed to replace the flawed processors. Many scientific applications require precision and one such example is Mathworks. With the bug in place, Mathworks had to rewrite much of Matlab software to ensure that the processor would not produce incorrect results.

2) Research, identify and briefly describe the 3 worst real-time mission critical designs errors (and/or implementation errors) of all time.

a) Some candidates are Three Mile Island, Mars Observer, Ariane 5-501, Cluster spacecraft, Mars Climate Orbiter, ATT 4ESS Upgrade, Therac-25, Toyota ABS Software.

b) Note that Apollo 11 and Mars Pathfinder had anomalies while on mission, but quick thinking and good design helped save those missions and we discuss these in class, so please don't pick these two near failures.

c) State why the systems failed in terms of real-time requirements (deterministic or predictable response requirements) and if a real-time design error can be considered the root cause.

Mars climate orbiter: After the launch, the spacecraft was put into a Hohmann transfer orbit to intersect with Mars. It performed four course corrections and finally, the orbiter began its Mars orbit insertion burn as planned. However, after passing behind Mars, no signals were received from the spacecraft. It was thought that it encountered Mars on a trajectory that brought it too close to the planet, and it was either destroyed in the atmosphere or escaped the planet's vicinity and entered an orbit around the Sun. In the 4^{th} course correction manoeuvre, the optimal position was calculated to be 226 kms. However, the navigation team later reported that it could be lower than planned- around 150kms. 24 hours prior to the orbital insertion, calculations reported 110kms; where 80 kms was the minimum altitude that the spacecraft could handle. However, in the post-failure analysis, it was observed that the spacecraft trajectory would have taken it to within 57 kms of the surface. The failure is attributed to a measurement mismatch between two software systems: ground software supplied by Lockheed Martin which used US customary units while, a second system supplied by NASA expected those results to be in SI units. This is more specifically, the software that calculates total impulse produced by thruster firings (in pound-force seconds). The trajectory calculation software expected these to be N-s to update the predicted position of the aircraft. Thus, real-time design error would not be the root cause. It would be attributed to human error or lack of testing protocols.

Therac-25: The Therac-25 was a radiation therapy machine manufactured by AECL, which offered dual treatment mode to treat cancer. The linear accelerator could fire a beam of low flow-energy electrons and a beam of higher-energy X-Ray photons. The 25 version is different from the previous versions, 6 and 20, in that it was designed to use software-based safety system rather than hardware controls. The preceding models used separate circuits to monitor radiation intensity, and hardware interlocks to ensure that spreading magnets were correctly positioned. One of the accidents occurred when the radiology technician performed the data entry quite fast and accidently pressed 'x' for X-Ray rather than 'e' for Electron, and then choose the correct mode by pressing the up key. However, the system displayed the error-Malfunction 54 and due to the high frequency of the malfunction messages, the error was ignored. The high-power beam had been used without the correct hardware structure (spreading the magnets in place). This would have been prevented by hardware safety controls in Therac-20 (the code base for Therac-25 is structured from Therac-20). The codebase consists of several routines running concurrently: data entry and keyboard handler routine shared a single variable which recorded where the technician had completed entering commands. If the entry was marked complete, and there if there were edits applied during the 8 second magnet setting phase, it would not be applied to the machine hardware, due to the variable which indicates competition. In another incident: a one-byte shared variable would indicate whether the hardware is configured correctly (a non-zero value indicates failure). However, on the 256^{th} attempt of setup, the shared variable would be set to 0 inherently. If the operator verifies the machine position and presses 'set' at this moment, the program would fire an X-ray beam. This is another concurrency bug due to an

overflow of one-byte variable. Thus, this is a real-time design error which would have been prevented if race conditions were analysed in the codebase.

Ariane 501: The failure is attributed to loss of information due to design error in the software of inertial reference system. The rocket uses an SRI system to determine the directions ie. horizontal bias (BH value). It is represented by a 64-bit floating variable. The SRI system used is similar to that of one used in Ariane 4. The part of the software which caused the interruption in the inertial system computers is used before launch to align the inertial reference system and, in Ariane 4, also to enable a realignment of the system in case of a late countdown. However, this was not required in Ariane 5. The system was also not protected from being made inoperative by an excessive BH variable, as it was not fully understood what range of values the variable would hold when the alignment software was allowed to operate after lift-off. This was not an issue in Ariane 4. However, in Ariane 5, there was a high horizontal velocity. The problem arose when the software was saving this 64-bit variable in a 16-bit integer. Here, the processor encountered an operand error and populated the BH variable with a diagnostic value. The backup Inertial reference system also failed due to the same error condition. This was interpreted as actual flight data, and caused the engines to over-correct by thrusting in the wrong direction. Thus, this isn't a real-time design error, instead it is a system-level design error which could have been prevented if the tests performed on-ground included the appropriate trajectory data.

3) The USS Yorktown is reported to have had a real-time interactive system failure after an upgrade to use a distributed Windows NT mission operations support system. Papers on the incident that left the USS Yorktown disabled at sea have been uploaded to Canvas for your review, but please also do your own research on the topic.

a) Provide a summary of key findings by Gregory Slabodkin as reported in GCN (https://gcn.com/1998/07/software-glitches-leave-navy-smart-ship-dead-in-thewater/290995/)

The Yorktown smart ship was a success in reducing manpower, maintenance and costs. The ship used dual 200-MHz Pentium Pros from Intergraph Corp. The PCs and server run NT 4.0 over a high-speed, fiber-optic LAN. The navy began running shipboard applications under Microsoft NT so fewer manpower would be required to control key ship functions. The incident caused the ship to be towed into the Naval base at Norfolk, Va.

Key findings:

- A database overflow caused the ship's propulsion system to fail. The computers were unable to divide by the number zero. The incident occurred when sys admin entered zero into the data field for the Remote Database manager program. This caused the database to overflow and crash all LAN consoles and miniature remote terminal units.
- The source of the problems is attributed to the operating system: NT. Applications on the Yorktown provide damage control, run the ship's control centre on the bridge, monitor the engines and navigate the ship as well. It has been agreed that Unix is a better system for control of equipment and data while NT is appropriate for transfer of information and data.
- The system design of shipboard LANs is such that there is very little segregation of error when software shares bad data. The LAN computers experience a chain reaction of computer failures.
- The Yorktown smart ship did not have any backup systems. Designing a system without any redundancy is a major design flaw. Installing the systems and resolving issues as the project progresses is also a costly effort.

b) Can you find any papers written by other authors that disagree with the key findings of "Gregory Slabodkin as reported in GCN" story above? If so, what are the alternate key findings?

Alternate papers referenced: https://www.route-fifty.com/digital-government/1998/08/smart-ship-inquiry-a-go/291581/

Alternate key findings:

- The responsibility for ensuring ship operations does not rest with the OS- Windows NP. But with the software programmers, who should have safeguarded the application from propagating errors.
- The Yorktown is equipped with two FFG-7 emergency power units in the event of a propulsion system failure.
- Windows NT was chosen simply due to the fact that it was specified in the Statement of Work as the operating system for the ships' workstations. There was no through analysis of Unix or NT. A statement released by the Navy stated that Unix would be considered for future Smart Ship technologies as many of their system had already been utilizing Unix-based systems.

c) Based on your understanding, describe what you believe to be the root cause of the fatal and near fatal accidents involving this machine and whether the root cause at all involves the operator interface.

Root causes:

- Software system components should be fault tolerant. The dependency graph of the Smart ship components would show that active fault in a foundational one such as SMCS could result in failure for all.
- Software applications should validate input data before processing it. Such validation can prevent operator error and maintain system reliability. Problems such as arithmetic overflow, buffer overflow can occur if not not handled correctly.
- Another issue faced was the lack of time invested to design, construct and test the software on-board.

d) Do you believe that upgrade of the Aegis systems to RedHawk Linux (https://concurrentrt.com/products/software/redhawk-linux/) or another variety of real-time Linux would help reduce operational anomalies and defects compared to adaptation of Windows or use of a traditional RTOS or Cyclic Executive? Please give at least 2 reasons why or why you would or would not recommend Linux.

I believe upgradation of the Aegis system to Redhawk Linux would have helped operational anomalies and defects compared to the usage of Windows NT as the base operating system. Even though the errors seen was not completely due to the OS selected, Linux OS would have made for a better system. However, traditional RTOS or cyclic executive would have been more difficult to incorporate with the other application code being used on the test bed.

- Windows NT was not originally designed for a client/server environment. In Linux, however, shared network filesystems can be mounted at any point in the directory structure and can span multiple disk drives. This allows system administrators to maintain pre-existing directory structures which are known to the user, while allowing them to expand the available disk space on the server.
- Since Windows NT is GUI-based, it requires human intervention for running any executables. However, running automated tasks is made useful in Linux.

- Even though Windows NT had considerably more stability than its previous predecessors, it was still prone to frequent system crashes due to even day-to-day operations like leaving the system on too long while running a myriad of applications. The only method of recovering is powering the machine off and rebooting them. This is called "Blue screen of death" and it is often difficult to troubleshoot it due to cryptic or non-existent error reporting. Typical cause for failure includes memory access violations, I/O errors, and erroneous implementation of tcp/ip stack. Further, Windows NT was particularly prone to virus attacks on Intel-based hardware, which wasn't an issue on UNIX operating systems. This is an important fact to consider in a mission critical environment, such as the Yorktown Smart ship. In general, Linux systems experience kernel panics but only because of hard drive failures, kernel upgradations or power outages.
- Linux systems' kernels can be custom-compiled to contain only the software required for the system. Any operating system requiring fewer resources can outperform more common OS like windows NT and thus they can function more efficiently. Especially systems that require graphics will take up more disk space space and memory and can cause harsher system performances.

4) Form a team of 1, 2, or at most 3 students and propose a final Real-Time prototype, experiment, or design exercise to do as an individual or a small team. Creative projects are possible if they have real-time requirements and timing constraints. However, for the summer RT-Systems class, I recommend that in the interest of time you complete a simple real-time machine vision synchronome (which is based upon prior creative project requirements for time-lapse monitoring design as outlined in time-lapse creative project requirements with more details and up-to-date goals and objectives provided in these background notes to be graded according to this rubric.

a) Based upon requirements and your understanding of the UVC driver and camera systems with embedded Linux on your Raspberry Pi, evaluate the services you expect to run in real-time with Cheddar first. Then, once implemented, plan your runtime verification analysis including methods to test with tracing and profiling to determine how well your design should work for predictable response. Describe in this proposal in good detail.

b) You will complete this effort in Lab Exercise #6, making this an extended lab exercise, and ultimately you will be graded overall according to this rubric and you must submit a final report with Lab Exercise #6 as outlined here, but at this point, you are asked simply to propose your solution at this point. You may find the "Final Project" videos on various design options helpful as you come up with your proposal (along with Coursera Lightboard videos).

c) Your Group should submit a proposal that outlines your response to the requirements, your design concept and methods of modeling and implementation analysis (Cheddar and tracing on Linux). This should include some research with citations (at least 3) or papers read, key methods to be used (from book references), and what you read and consulted.

Project Proposed:

Standard synchronome: Synchronization of capture of digital camera frames, which has a standard frame rate of around 24 FPS, to the ticking of a 1 Hz analog clock and 10 Hz digital stopwatch. The captured frames should be without any glitch ie. skip, repeat, blurs.

Project Requirements:

1. Resolution must be at least VGA (640x480)
2. Frame acquisition service samples at 1 Hz based on an interval timer but the camera can continuously encode frames at a rate of 24 frames/sec

3. Image file format must be PPM "P3" or "P6" RGB or PGM “P2” or “P5” format
4. Frame acquisition and timestamping should be accurate such that there is no ‘observable’ error in 2000 frames or a duration of 33 minutes and 20 seconds.
5. Timestamps, and host/target platform name is embedded into the PPM/PGM header of every frame saved.

Verifications:

1. Verification of the program run time with the timestamps of the first and last frame.
2. Computation of average frame jitter and accumulated latency.
3. Observation of another physical process to prove the project is a true time-lapse sequence.

Target goals:

1. Compression of frames so more than 2000 frames can be stored.
2. Continuous download of frames over ethernet so that the program can run indefinitely and never run of space on flash file system.
3. Implementation of frame transformation ie. image processing as an on/off feature.

Stretch goals:

1. Run the program at a higher frame rate of 10 Hz with continuous download for 9 minutes- this provides 6000 frames. Repeat jitter and accumulated latency.

Proposal: The following real-time services are proposed:

- Sequencer: The sequencer would schedule all the real-time threads in the system and would run at the highest frequency. The other threads would run at a factor rate of this highest frequency. Let the frequency of the sequencer be 100 Hz. The sequencer thread would be controlled by an interval timer which runs on the system oscillator.
- Image acquisition and frame processing: The following table was measured in the previous questions of the course. Here we see that the maximum WCET of the entire process is 74.5 milliseconds. For just the frame acquisition and frame processing, the average time taken together is around 15ms. The maximum WCET for this is considered to be 20ms.

Process	Time taken (seconds)	FPS
Total capture time	9.193875	19.7957
Frame acquisition time (avg)	0.000024	41025.59
Frame processing time (avg)	0.012961	77.15
Frame storage time (avg)	0.003277	305.1131
WCET (avg)	0.052606	
WCET (max)	0.074565	

- Frame storage in ring buffer: The captured image will be stored in a ring buffer. This is to provide IO decoupling as the frame storage has the worst WCET and is depended on the flash file storage system [1]. The WCET will keep varying depended on the wear-levelling of the storage system. IO decoupling will provide a more deterministic system. The WCET for this is considered to be 10ms approximately.

Best-effort services:

- Frame storage: The image buffer will be stored as a ppm/pgm file on the system. Since the storage takes an indeterministic amount of time, it is proposed as a best-effort service.

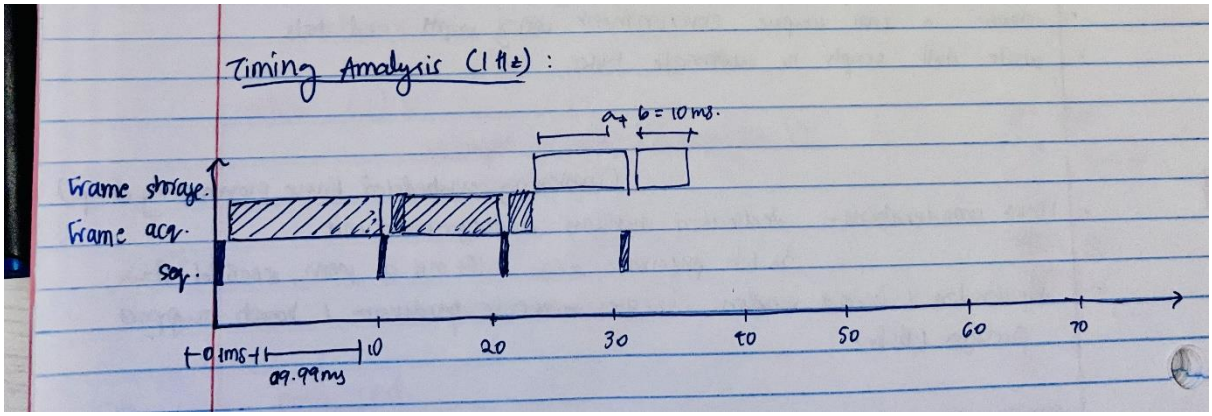
- Server: For the file to be transferred over the Ethernet over a socket, this service is proposed. The host machine with the file and file header will serve as a server which will send the files to the client machine on the same IP address.

For 1Hz: Using the rate-monotonic policy where period=deadlines is used to evaluate the feasibility of the services:

Services	Freq f	Period	(ms)	WCET	(ms)	Utility			
Sequencer	100Hz	T1	10	C1	0.1	U1	0.01	**LCM**	**1000**
Frame acq	1Hz	T2	1000	C2	20	U2	0.02	**LUB**	**77.98%**
Frame storage	1Hz	T3	1000	C3	10	U3	0.01	**Utot**	**4%**

```
Scheduling simulation, Processor CPU1 :
- Number of context switches :  9
- Number of preemptions :  3

- Task response time computed from simulation :
   S1 => 1/worst
   S2 => 304/worst
   S3 => 102/worst
- No deadline missed in the computed scheduling : the task set is schedulable if you computed the scheduling on the feasibility interval.
```

```
Scheduling feasibility, Processor CPU1 :
1) Feasibility test based on the processor utilization factor :

- The hyperperiod is 10000 (see [18], page 5).
- 9600 units of time are unused in the hyperperiod.
- Processor utilization factor with deadline is 0.04000 (see [1], page 6).
- Processor utilization factor with period is 0.04000 (see [1], page 6).
- In the preemptive case, with RM, the task set is schedulable because the processor utilization factor 0.04000 is equal or less than 1.00000 (see [19], page 13).


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

- Worst Case task response time :  (see [2], page 3, equation 4).
   S2 => 304
   S3 => 102
   S1 => 1
- All task deadlines will be met : the task set is schedulable.
```



For 10Hz:

Services	Freq f	Period	(ms)	WCET	(ms)	Utility			
Sequencer	100Hz	T1	10	C1	0.1	U1	0.01	**LCM**	**1000**
Frame acq	10Hz	T2	100	C2	20	U2	0.2	**LUB**	**77.98%**
Frame storage	10Hz	T3	100	C3	10	U3	0.1	**Utot**	**31.1%**

Similarly, for the 10Hz frame rate, the services are proven to be schedulable as the CPU utilization is lower than that of the RM LUB [2].

Verification efforts:

- To check if the tasks are meeting deadlines, system performances are evaluated in terms of latency and response-time jitter [3]. This is done with a cyclictest to accumulate latencies from several time samplings.
- Each service duration will be recorded to check if it is running for the estimated WCET time or if it is not meeting its deadlines. With this, each service jitter and latency can be calculated. This way, the predictability of the services' responses can be verified.
- All the timestamps will be stored in the syslog file (ie. tracing of events with syslog) so as to not delay the program execution with printf statements.
- Using interval timer, the sequencer will be scheduled so provide more accurate timing analysis as nanosleep does not provide the most accurate delay.
- Each captured frame will have the accurate timestamp received through clock_gettime() so the time delay in between each frames can be verified.
- The captured frames will be converted into an mp4 video using ffmpeg for a true time-lapse sequence.
- Option to enable or disenable features provided by the program at run-time.

d) Each individual should turn in a paragraph on their role in the project and an outline of what they intend to contribute (specific feature, design, documentation, testing, analysis, coding, drivers, debugging, etc.) and team schedule. For groups of 2 or 3, it is paramount that you specify individual roles and contributions. For those of you working alone, just provide a basic outline of your schedule to complete the project.

Schedule outline:

Design and coding:

1. Implementation of services with CPU affinities set.
2. Implementation of ring buffer to provide IO decoupling.
3. Implementation of global clock for synchronisation.
4. Implementation of frame transformation as an on/off feature.
5. Syslog tracing for every service.
6. Implementation of socket communication for client and server machine over ethernet.

Testing:

1. Checking the system response ie. jitter and latency and making sure the response is predictable and that it does not accumulate delay over time.
2. Checking the captured frames are glitch-free.
3. Communication over socket functionality

Documentation:

1. Final report submission
2. Encoding the captured frames into an mp4 file
3. Jitter and latency data in comprehensible, clear formal.

Papers and citations: The following papers were referenced for the project proposal:

[1] S. Siewert, "IO latency hiding in pipelined architectures," 2005 IEEE Region 5 and IEEE Denver Section Technical, Professional and Student Development Workshop, Boulder, CO, USA, 2005, pp. 39-45, doi: 10.1109/TPSD.2005.1614345.

[2] C.L LIU and JAMES W. LAYLAND. “Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment” in Journal of the ACM, vol. 20, no.1, pp,46-61, Jan. 1973

[3] Reghenzani, Federico & Massari, Giuseppe & Fornaciari, William. (2019). The real-time linux kernel: A survey on Preempt_RT. ACM Computing Surveys. 52. 1-36. 10.1145/3297714.

Report References:

1. http://www.lege.com/unix-nt/
2. https://medium.com/dataseries/when-smart-ships-divide-by-zer0-uss-yorktown-4e53837f75b2
3. https://www.jerrypournelle.com/reports/jerryp/yorktown.html#smart1
4. https://groups.google.com/g/comp.os.ms-windows.nt.advocacy/c/EhJ7ZOPNOXY
5. https://people.freebsd.org/~andreas/unix-vs-nt/index.html
6. https://ethics.csc.ncsu.edu/risks/reliability/pentium/study.php
7. https://en.wikipedia.org/wiki/Pentium_FDIV_bug