ECEN 5623 - Real Time Embedded Systems

Time Lapse System

- ***Gautham K A***

Contents

Introduction:

Real time systems are defined as the systems producing correct response within the time specified. In the real world usually, a real time system service is often indicated by asynchronous interrupts. The system has to get relevant inputs from the system and do necessary processing and deliver to user within deadline defined for the service. Deadline is defined as the time relative to a request for service when the service has to be completed to realize full utility of the service.

Here we are designing a Time Lapse System which is a soft real time system. Time Lapse system is the one which does real time acquisition through camera and produce output to user with various services defined. This project consists of 1Hz and 10Hz image capture and save services. Time Lapse video is another term used for fast motion video. These are used in situations to study the processes that are very slow in nature.

This project is a simulation of a real world real-time system without major computation involved, that is without major execution time (full processor utilization) involved in the services that can produce effective user outputs which can help the humans ease their tasks.

System Functional and Overall Design:

Time Lapse system is a real time system with simple video editing application. Basically, we design time lapse system, as for a normal service to be perceived by humans as video is to have at least 24 frames per second. The digital Camera device that we are using has maximum frame rate of 30 fps but in usual lighting it can produce around 20 frames and hence we are capturing at lesser rates to compensate for I/O speed deficiency.

The system is designed to capture frames at 10 Hz. Since we are creating time lapse video of capturing one frame per second, of the 10 available, one frame is taken and is written to file system.

Services have to run in periodic nature. Here in order to enable this I am running a sequencer which releases resources for threads at regular appropriate intervals. This sequencer would be a highest priority service running at 20 Hz. This becomes our most important part of our real time project as to control the operation of threads in timely manner for correct real time response. Since using nano sleep which by far has the highest resolution for delay generation (sleeping mechanism) is also not efficient for long runs. So, mechanisms to adjust rates are to be done for it to release resources at right intervals.

1Hz Design:

Real time service requirement that we are designing here is to get a frame at each second of actual clock. If the system starts at 0 seconds after 2000 seconds (i.e. 33 minutes 20 sec) we need to have correct output which is frame from each second (2000 frames). Constraints that have to be taken care in design are jitter in sequencer scheduling and frame capturing and writing execution delays.

So, since I have to save only one frame for each second, I will start my saving thread at 700ms after release of capturing service. So, my virtual deadline for writing service would be the remaining 300ms out of 1000ms available. This gives me opportunity to select best frame any of the seven available frames.

10Hz Design:

In 10Hz case, my deadline to capture each frame is 100ms. And since saving the image takes more than 100ms (compression and other processing [wcet]), it is not made a real time service and it is a best effort service. So, my design involves two threads which does capture and save of image within the deadline of 100ms (real time services).

Another added functionality would be to do image write only for those images where the pixels value percentage difference is more than 1 percent with previous captured frame.

Functional Capabilities:

- System can write(save) images at each second of real time clock for about 30 minutes with real time effectiveness.
- System saves images with information like timestamp and processor information.
- System can capture images at 10 frames per second for each second of real time clock.
- System can save images at 10 Hz and write it to system as a best effort service at 640*480 resolution.
- System can save images at a compressed format with different levels of compression and the resultant image format would be jpg format.
- System can analyze the images captured and write only those to the system which have pixels value greater than 1 percent change.

Real time Requirements:

As discussed briefly in design overview the system has two services running as part of real time service implementation application.

Sequencer: High Priority thread which schedules other services at defined periodic times. This thread is made to run at 20 Hz. Services of lesser priority would get resource from this service to continue its operation.

1 Hz System

Frame Capture Service:

This service thread is made to run at 10hz. So, it captures 10 frames every second. Deadline for capturing 1 frame is 100ms. So, it is designed to capture frame every 100ms in one second period. This service gets a Mat Frame with resolution defined as 640 (width) and height (480). This service has a buffer of 20 Mat frames to store. Buffer is made to operate as a circular buffer. Each frame capture takes about 20ms in average.

Frame Write Service:

This service is where we write images on to memory of the system. For 1 Hz functionality we write one image out of 7 captured frames available for it to select from. This service has deadline of 1 second. This service is made less priority than Frame capture service which has higher frequency according to RM theory. This service takes about 15ms in average. It also embeds timestamp and uname information of the board being used.

10Hz System:

Frame Capture Service:

Same Service mechanism is used for this operation as utilized for 1Hz system.

Frame Copy Service:

This service is a real time service which does the copying of Mat frame captured into another Mat frame buffer for further writing images on to system. This service has deadline of 100ms. So, for each second, we will have buffer increments of size 10. Execution time for this service in average is 4ms

Frame Write service:

This is best effort service and no deadlines are defined.

The Sequencer posts resources for service usage at every 100ms for frame capture and frame copy services. Since Priority for both services are same according to RM theory with same frequency we use synchronization scheme further for correct output operation as frame copy service depends on frame capture service.

Each of these services include deadlines that can accommodate jitter for execution delays and sequencer sleep drift.

Functional Design and Overview:

Hardware Design:

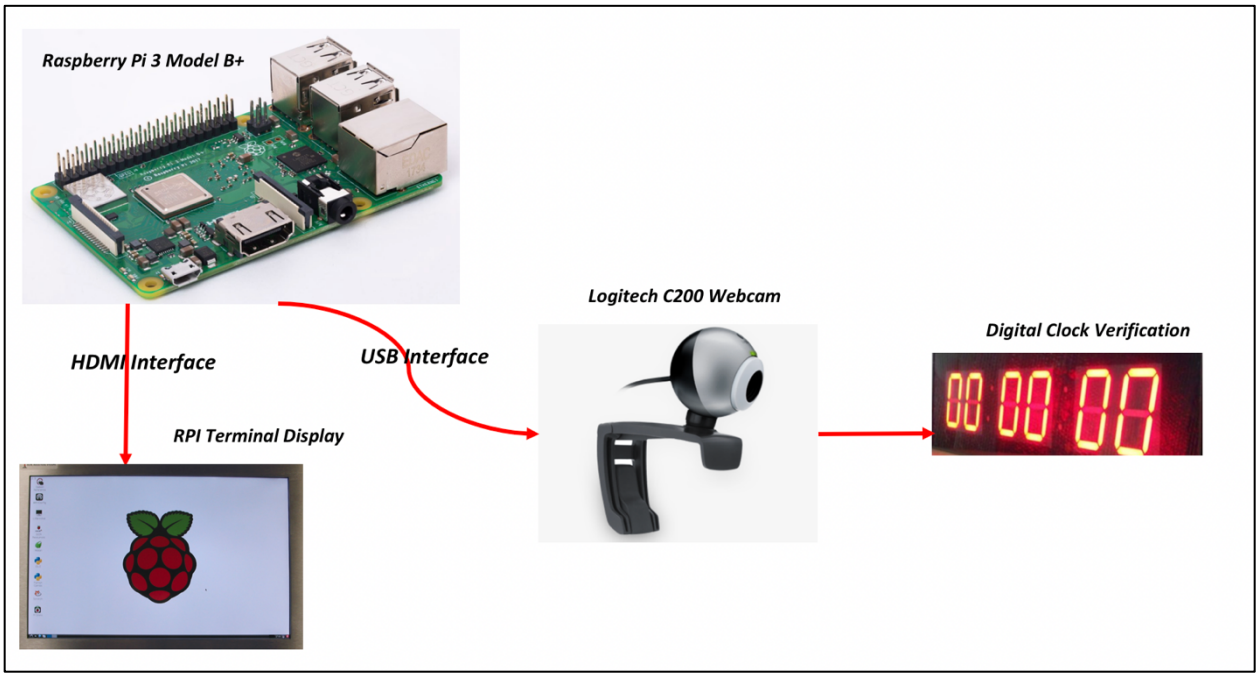


Figure: Hardware Block Diagram

We have very simple hardware block diagram. Raspberry Pi 3 model B+ is used for this embedded real time system, software powered by raspbian OS based on debian (a standard for RPI). Hardware Specifications of the board are: Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC running at 1.4GHz. It has support for full HDMI which we use with a monitor for user interface.

It has got 4 USB 2.0 ports out of which three of them are utilized for keyboard, mouse and camera.

Camera has a feature to focus the image capture. We use digital clock or analog clock for verification of system performance by capturing image frames of running clock appropriately.

RPI is powered by 5v/2.5A.

The Cortex-A53 processor has one to four cores, each with an L1 memory system and a single shared L2 cache (1-4x Symmetrical Multiprocessing (SMP) within a single processor cluster, and multiple coherent SMP processor clusters through AMBA 4 technology). It has memory of around 20GB for efficient image processing and storage applications.

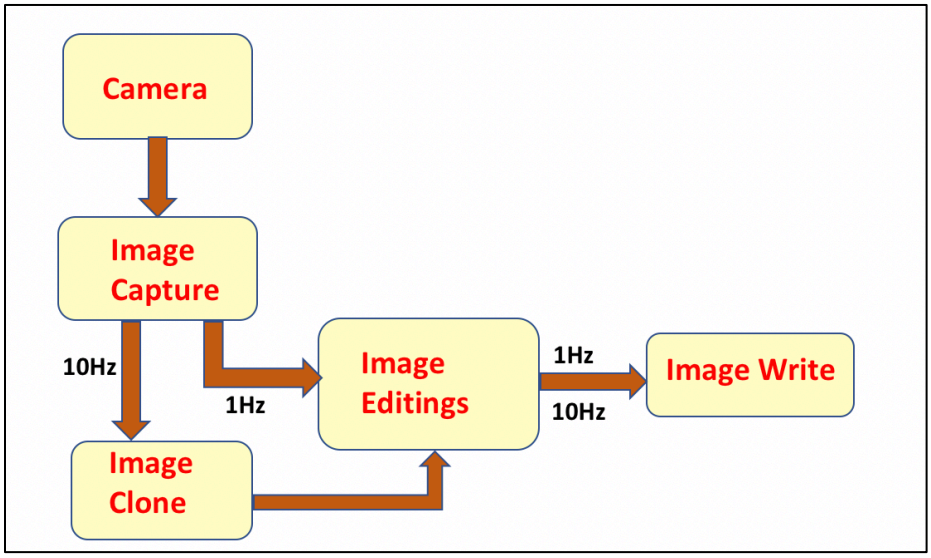
Software Design:

For any real-time system, software plays a crucial role in effective and responsive real time response operation. There are many scenarios where a good software design with limited hardware resources can exceed in performance, which has been the trend in recent days. For our simple hardware system with enough resources simple and reliable software design is made.

For the camera features access and minimal image processing, OpenCV 3 is being used.

For the efficient synchronization posix - pthreads are being used with necessary synchronization mechanisms. Threads are created with priority assigned according to RM theory and are set affinity to cores for efficient performance. Real time services are made to run on single core to extract real time performance for this application and further building on it.

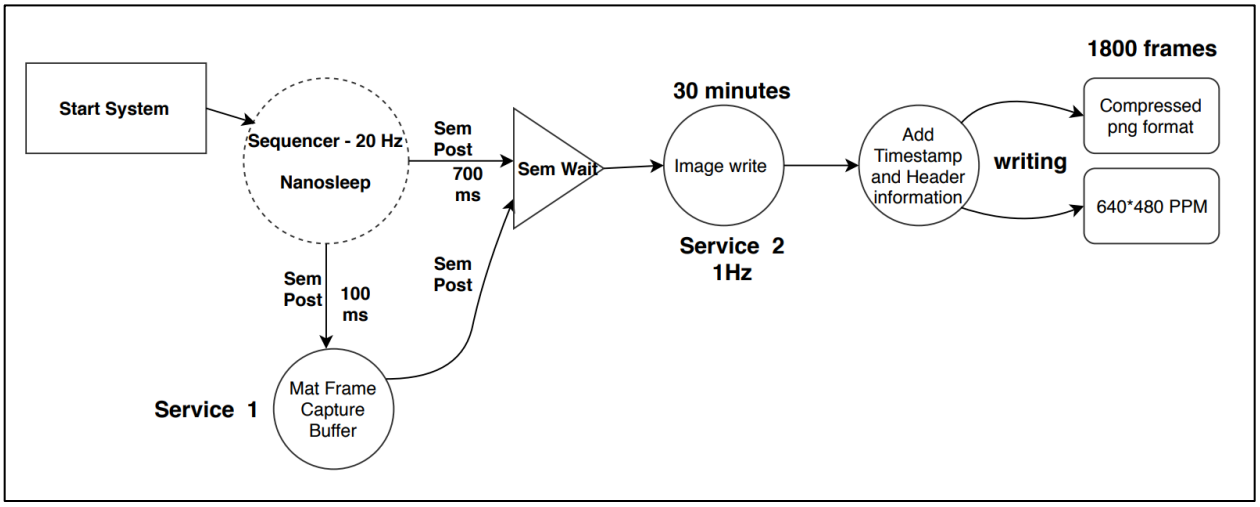
As explained previously, system has basically 4 different threads. Sequencer thread is created with highest priority and is run at 20 Hz. It involves releasing semaphores at regular intervals for execution services which is synchronized to system clock timing using nano-sleep feature. Since, system requires frame capture to run first and acquire frame, it has high priority which releases resource after execution to copy or write thread to continue operation on the captured image frame. If the service is running at low frequency it is made to run at lower priority. Here is the simple System Flow chart of the software system:



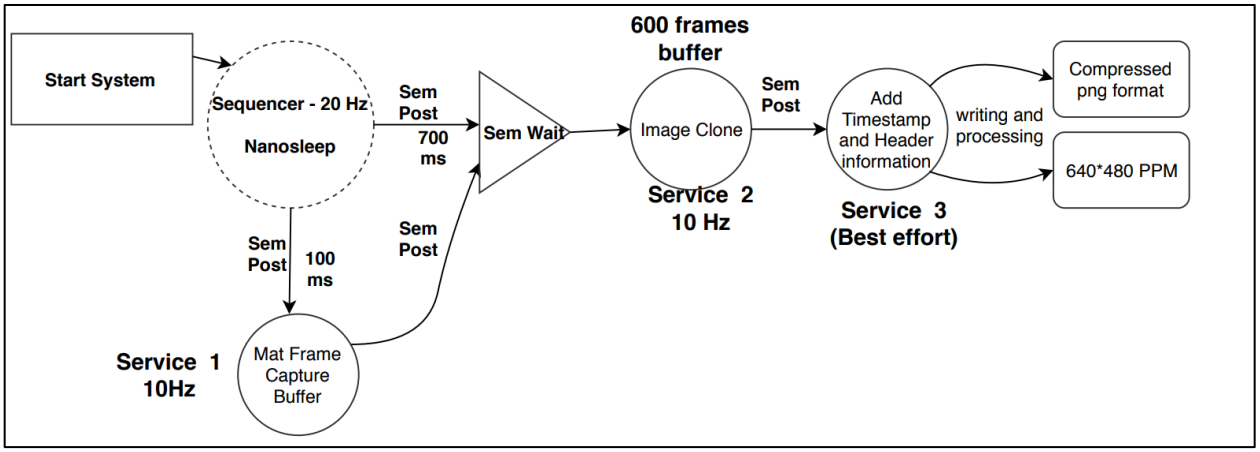
State Diagram:

This diagram shows all the various states system can be during thread execution:

1Hz:



10Hz



Real Time Analysis and Design:

Real time service sets were run and profiling on each service and sequencer thread is done. Each service execution time is calculated many times with different lighting conditions and at different instances. By this process, WCET for each service is calculated

This analysis was done to see that the fixed deadlines were not crossed. Deadline for 10 Hz program was to capture each image within 100ms. However, without compression and any image processing, I observed that capturing and writing happen within 100ms. Hence to accommodate compression and image difference analysis I have created a save thread which buffers frames for writing in another service thread which is made to be a best effort service without deadline.

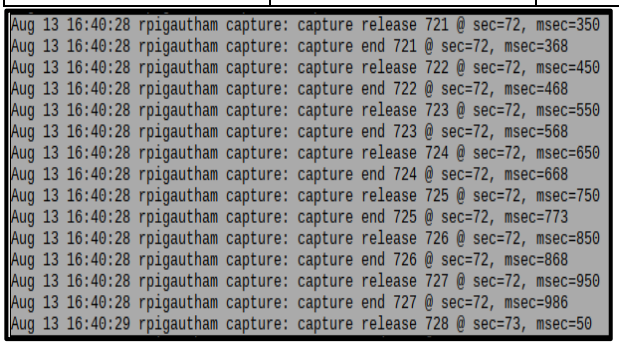
Deadline for 1 Hz process is 1 second for frame capture and write process.

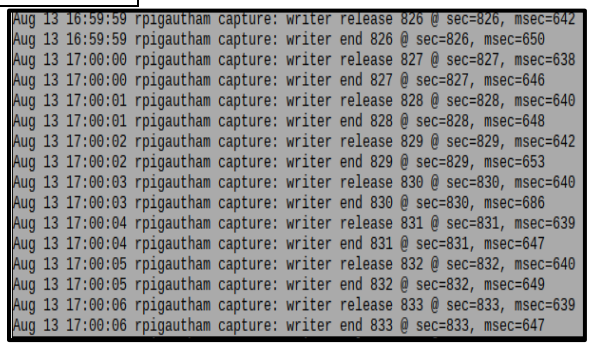
Worst case execution times are calculated for long run of the system and taking values at different times, average value is found to be within deadline and hence we can say that system is performing as expected in real time correct manner.

Ci and Worst-case Execution Times:

1 Hz System:

	WCET	Average Ci
Service 1	**36ms**	**25.05ms**
Service 2	**46ms**	**13.29ms**





Sample Snaps of service execution times on Syslog window.

Service 1 is the capture service running at 10 Hz and deadline is 100ms. From the WCET and Average execution times we see that there is no execution or response jitter which can affect behavior of 1Hz system to be a real time response system.

10 Hz System:

	WCET	Average Ci
Service 1	**22ms**	**19.33ms**
Service 2	**5ms**	**4ms**

```
Aug 13 17:27:28 rpigautham capture: writer release 2019 @ sec=241, msec=317
Aug 13 17:27:28 rpigautham capture: capture release 2410 @ sec=241, msec=330
Aug 13 17:27:28 rpigautham capture: capture end 2410 @ sec=241, msec=349
Aug 13 17:27:28 rpigautham capture: save release 2410 @ sec=241, msec=349
Aug 13 17:27:28 rpigautham capture: save end 2410 @ sec=241, msec=352
Aug 13 17:27:28 rpigautham capture: writer end 2019 @ sec=241, msec=398
Aug 13 17:27:28 rpigautham capture: writer release 2020 @ sec=241, msec=398
Aug 13 17:27:28 rpigautham capture: capture release 2411 @ sec=241, msec=429
Aug 13 17:27:28 rpigautham capture: capture end 2411 @ sec=241, msec=448
Aug 13 17:27:28 rpigautham capture: save release 2411 @ sec=241, msec=448
Aug 13 17:27:28 rpigautham capture: save end 2411 @ sec=241, msec=452
Aug 13 17:27:28 rpigautham capture: writer end 2020 @ sec=241, msec=487
Aug 13 17:27:28 rpigautham capture: writer release 2021 @ sec=241, msec=487
Aug 13 17:27:28 rpigautham capture: capture release 2412 @ sec=241, msec=530
Aug 13 17:27:28 rpigautham capture: capture end 2412 @ sec=241, msec=549
Aug 13 17:27:28 rpigautham capture: save release 2412 @ sec=241, msec=549
Aug 13 17:27:28 rpigautham capture: save end 2412 @ sec=241, msec=552
Aug 13 17:27:28 rpigautham capture: writer end 2021 @ sec=241, msec=586
Aug 13 17:27:28 rpigautham capture: writer release 2022 @ sec=241, msec=587
Aug 13 17:27:28 rpigautham capture: capture release 2413 @ sec=241, msec=630
```

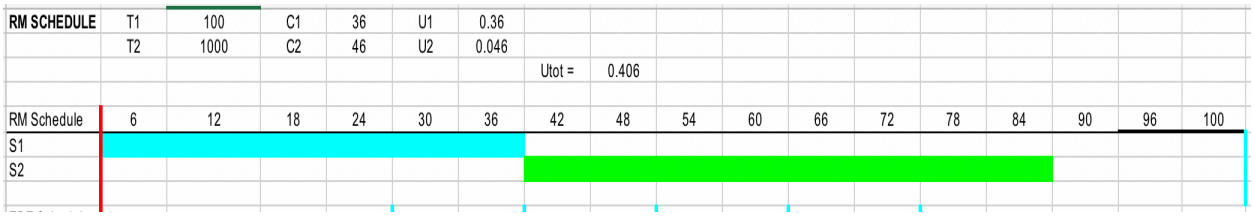
Even though capture frame service is implemented similarly for both systems we observe higher WCET in 1 Hz system as videocapture read is being performed which checks for blank frame. This adds to execution delay. Since 10 Hz system needs to run faster as both capturing and saving needs to be done in deadline of 100ms, it is not added.

If camera operations are performed in bad light, frame capture service does not operate reliably This has to be taken for correct system response.

Another important analysis to be done for correct operation of this service is that capture frame service has to start at every 100ms interval from the system start. If there is jitter accumulated due to sequencer operation (nano-sleep implementation) we will not be able to synchronize with external clock which is the main parameter of this project. We can see that 5ms of jitter accumulation with the system running for about 30 minutes.

RM Scheduling:

Hand-drawn RM Schedule:



The above image shows RM Schedule for the system. We can calculate the system utilization.

$$U = \sum_{i=1}^{m} Ci/Ti \leq m\,(2^{1/m} - 1)$$

C1/T1 + C2/T2 = 0.36 + 0.0406 = 0.406

According RM lub, Utilization for two services is given by, m (2^(1/m) -1) where m is number of services. Therefore U = 2(2^0.5 -1) = 0.82.

So, we can see that services are schedulable with great margin present for further processing and proper response from the system.

For 10 Hz system utilization is found to be 0.27, C1/T1 + C2/T2 = 0.22 + 0.05

So, even this service is schedulable.

Cheddar Analysis

<u>For 1Hz System:</u>

Two real time services are present.

Service 1: Frame Capture Service: C1 = 36, T1=100

Service 2: Frame write service: C2=46, T2=1000

Cheddar Simulation for above inputs is done to get positive feasibility results as shown in snap below

```
Scheduling feasibility, Processor cpu1 :
1) Feasibility test based on the processor utilization factor :

- The hyperperiod is 1000 (see [18], page 5).
- 594 units of time are unused in the hyperperiod.
- Processor utilization factor with deadline is 0.40600 (see [1], page 6).
- Processor utilization factor with period is 0.40600 (see [1], page 6).
- In the preemptive case, with RM, the task set is schedulable because the processor utilization factor 0.40600 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).
   T2 => 82
   T1 => 36
- All task deadlines will be met : the task set is schedulable.



Scheduling simulation, Processor cpu1 :
- Number of context switches :  2
- Number of preemptions :  0

- Task response time computed from simulation :
   T1 => 36/worst
   T2 => 82/worst
- No deadline missed in the computed scheduling : the task set is schedulable if you computed the scheduling on the feasibility interval.
```

For 10 Hz System:

Two real time services are present.

Service 1: Frame Capture Service: C1 = 22, T1=100

Service 2: Frame copy(clone) Service: C2=5, T2=100

Cheddar Simulation for above inputs is done to get positive feasibility results as shown in snap below.

```
Scheduling simulation, Processor cpu1 :
- Number of context switches :  1
- Number of preemptions :  0

- Task response time computed from simulation :
   T1 => 27/worst
   T2 => 5/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 100 (see [18], page 5).
- 73 units of time are unused in the hyperperiod.
- Processor utilization factor with deadline is 0.27000 (see [1], page 6).
- Processor utilization factor with period is 0.27000 (see [1], page 6).
- In the preemptive case, with RM, the task set is schedulable because the processor utilization factor 0.27000 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).
   T1 => 27
   T2 => 5
- All task deadlines will be met : the task set is schedulable.
```

Scheduling point and Completion point Tests

Since the RATE MONOTONIC test is a necessary test for feasibility check, necessary and sufficient tests are also performed on these inputs. Scheduling point and completion point tests shows that the schedule is feasible. Below is the snap of the code output (code from exercise is reused).

```
gautham@ubuntu:~/RTES/ex_2/qn_3$ ./feasibility_tests
******** Completion Test Feasibility Example
Ex-5 U=0.41 (C1=36, C2=46,; T1=100, T2=1000, T=D): FEASIBLE
Ex-6 U=0.27 (C1=22, C2=5,; T1=100, T2=100, T=D): FEASIBLE
******** Scheduling Point Feasibility Example
Ex-5 U=0.41 (C1=36, C2=46,; T1=100, T2=1000, T=D): FEASIBLE
Ex-6 U=0.27 (C1=22, C2=5,; T1=100, T2=100, T=D): FEASIBLE
```

Jitter Analysis:

We have two types of jitter which is of concern in a real time system like this.

Execution Jitter and Response jitter which are defined and analyzed in detail below:

Jitter in Sequencer Execution which releases the resources for services. So, this jitter will add to the services execution and cause synchronization issues even though the processes run within deadlines. This happens due to inaccuracy in sleep function in sequencer thread. I have designed compensation mechanism in my system which reduces this jitter by providing sleep time accordingly. With this process I could see worst case of 6ms of jitter for 1Hz system running for about 30 minutes. For 10 Hz system maximum jitter due to this reason was similar as same design is implemented.

Execution Jitter: When the service is dispatched, and the number of execution cycles required to complete the cycle on each release.

1Hz System:

Service 1[Frame Capture]: So, from the calculation of WCET, we can observe that WCET was found to be higher than average C (execution time) which indicates the presence of execution jitter. In real time systems with high utilization this would be a concern as it would lead to defining unnecessary higher deadlines. Since this system has high margin, occasional execution jitter does not affect the real time correctness as we observe each second is captured in different frames. Also, for digital camera if proper light is not made available it takes long time and response is not even (IO Bound).

The average execution jitter is found to be around 2.95ms.

When compression process is added to the system only service two would be affected.

Service 2[Frame Write]: This service writes the images on to the system. So, when added compression into writing the execution changes. Below are the times found for this service. This is because compression takes some execution time.

Jitter without compression: average value is 2.1ms.

Jitter with compression: average value is 6ms

10 Hz System:

Service 1[Frame Capture]:

Since the Frame capture service is implemented similarly in both scenarios (1Hz and 10Hz). The execution jitter is found to be same as above. However, the actual execution time value is different as mentioned in the design because reading the captured frame is not implemented in this system which is a redundant process. The execution value is reduced by around 15ms.

Service2[Frame Copy]:

Here second real time service is where we copy the frame in to larger buffer.

Average execution jitter is found to be: 0.66ms.

Since compression is not done in this thread we will not have different value if compression feature is included.

Response jitter:

This is variation in completion of service and synchronization with an external clock of physical process.

1Hz System:

For the 1 Hz system no delay in response is found. It is running before deadline. Since writing completes the real time operation of 1Hz design, writing thread was woken at around 700ms and on average its execution time is 13.29ms and with the worst case we can see that the system would run 250ms faster.

Here I have presented the average time of how fast the service is completed

Service 1 [Deadline 100ms]: 74.95ms

Service 2 [Deadline 1000ms] Without Compression: 286ms

Service 2 [Deadline 1000ms] With Compression: 271ms

10 Hz System:

Service 1 [Deadline 100ms]: 81ms

Service 2 [Deadline 100ms]: 96ms

Proof-of-Concept with Example Output and Issues faced:

Important task in this system is the high priority sequencer. This thread has to run with real time correctness. Since this service is implemented with nano-sleep which sometimes is not accurate in its operation we need to manage the jitter caused. This operation has to be verified for perfection. If first service resource release happens at 230ms next one should happen at 330ms for this system. This was verified to work with almost no jitter. But as mentioned I could see jitter which was around 4-5ms for the system running for about 33minutes. This can be an issue if the system runs for very long time and rigorous methods needs to be implemented for safety critical systems. Since execution times was calculated to be within deadlines and no major jitter is found that could cause the available margin to exceed, the system can be said to work in real time manner.

Times are profiled using syslog which will not add to jitter.

System is tested various times to see that the operation is reliable, verifying the operation using digital clock with minutes, seconds and milliseconds precision.

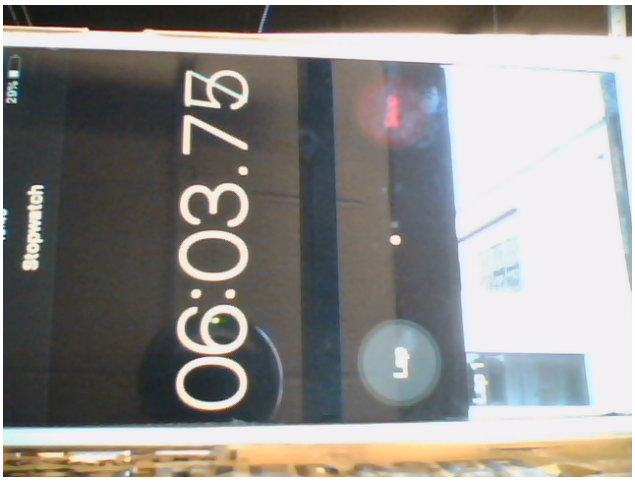
Issues: Initially the capture of 1800 frames in 30 minutes was happening without any jitter but frame capture and selecting the frame was an issue due to accumulation of jitter and release times of capture service was not even throughout. Hence, synchronization with external clock was not proper. Compensation for this jitter was done in the high priority sequencer thread and the system started behaving as expected. But due to lighting issues many times debugging was getting difficult as problem was not pointed out to be a light issue.

Below is the snap of the example output.

[Raw frames are uploaded, along with the video and code]

1 Hz:

10 Hz:



Conclusion

Real time system design is not as simple as any other system. Throughout the course we have learned the techniques and requirements for designing real time systems. Reading research papers and real time system failure scenarios gave a huge exposure to the industrial applications and way of thought required and used.

Since the requirements for this design was provided with regard to deadline and services, it did not create initial trouble to get minimum requirement part of the system running.

This system is made to run real time services on a single core. Frame capture and save in 1hz and frame capture and copy in 10hz are the real time services running on same core. Other threads are made to run on other cores that raspberry pi provides. High priority sequencer is made to run on different core. In 10 Hz system writing images to memory is carried out on different core as a best effort service.

1hz system requires us to capture 1800 frames in 30 minutes with each frame having different second capture of external clock.

10hz system requires us to capture 6000 frames in 10 minutes and for each second 10 frames, one each in 100ms is to be captured.

Compression of frames is added to this system which reduces the size and the required quality can be chosen in the compression parameters of the write process.

Difference analysis feature is added to this system for 10 Hz service which writes only images in which captured frames have difference of 1 % or higher difference value in pixels with previously captured frame.

Acknowledgements:

I would like to thank Real-time Embedded Systems Professor Sam Siewert for his valuable resources, lectures and help in analyzing the system to point out the issues and areas that could be improved. I am thankful for him for providing opportunity to work on designing a real-time embedded system and learn greatly from challenges faced.

References:

https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/

https://developer.arm.com/products/processors/cortex-a/cortex-a53

https://docs.opencv.org/3.4.1/d6/d6d/tutorial_mat_the_basic_image_container.html

https://docs.opencv.org/3.4.1/

http://ecee.colorado.edu/~ecen5623/index_summer.html

http://man7.org/linux/man-pages

Real Time Embedded Components and Systems with Linux and RTOS – Sam Siewert and John Pratt (book)