Lab Exercise 1 – Invariant LCM Schedules

Observations and Analysis

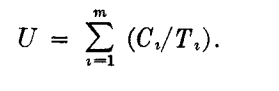
1.

Timing diagram of RM fixed policy schedule is as below :



*Fig 1.1 Timing Diagram for one LCM (15 cycles)*

The utilization for processor is defined by fraction of processor time spent in execution of task set [1] and mathematically as following:

(i)

where:

U is Utilization Factor

Ci – Worst case Execution time of a Task (run time) for Task i

C1, C2 and C3 for above Task 1 ,2 and 3

Ti – Request Time period (act as Deadline/Release time for RMS) for Task i

T1, T2 and T3 for above Task 1 ,2 and 3

Ci/Ti is fraction of processor time spent in executing Task

Therefore, calculating Ci/Ti for each task

For Task1: C1/T1 = 1/3.

For Task2: C2/T2 = 2/5.

For Task3: C3/T3 =3/15 = 1/5.

Utilization, U= 1/3+ 2/5 + 1/5 = **0.933** (as observed in above Timing diagram) --------(ii)

RM (Rate Monotonic fixed policy) states that higher priority is assigned to a Task having most frequently requested and lowest priority is assigned to task least frequently requested. The timing diagram is plotted by this definition.

Further Rate Monotonic Scheduling (RMS) Least Upper Bound(LUB) [2] is defined as follow:

(iii)

where m: is the number of services.

Using [iii] for three services U less than (2^ (1/3)-1) = .779763.

As **.933 > .7797,** indicates that it is not feasible.

But as we can see from Fig 1, Task 1 ,2 and 3 are plotted for a LCM of 15 ms and none of the services miss the deadline and is LCM invariant (does not change when harmonically repeated), it is unlikely to miss a deadline concluding to be safe using RMS.

According to [1] this Least upper bound test is a sufficient condition but **not a necessary** condition for feasibility. RMS is a pessimistic feasibility test and thus if the U is well above the required value, still it can be feasible.

**In conclusion, the above services with priority assigned according to RM are feasible and safe.**

2.

Ans

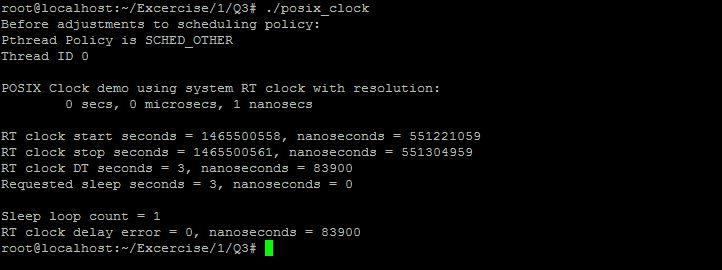
**Summary**

While descending Apollo 11 at moon CPU suffered from an overload condition ,which inturn could have resulted in aborting the first landing of moon . When Apollo 11 was trying to land on lunar ,an alarm 1202 started buzzing indicating that the CPU is does not have the resources to offer and allow the systems very well required for safe landing are not reliable .

3.

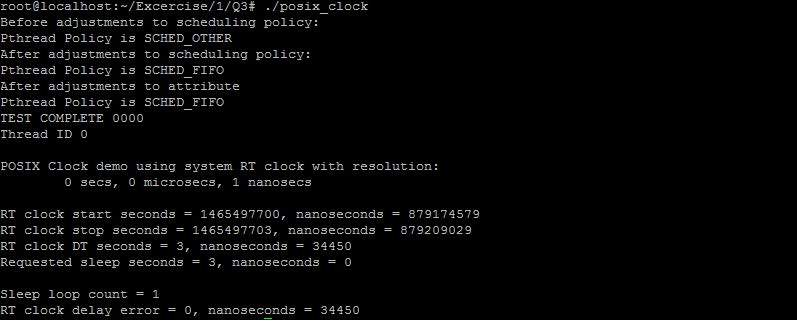
Ans

(i) Running RT\_Clock posix\_clock.c code on Ubuntu hosted on Altera DE1-SoC with scheduled FIFO\_OTHER as scheduling policy.



*Fig 3.1 Screen shot of execution of RT Clock Code using SCHED\_OTHER*

(i) Running RT\_Clock posix\_clock.c code on Ubuntu hosted on Altera DE1-SoC with scheduled FIFO as set scheduling policy



*Fig 3.2 Screen shot of execution of RT Clock Code using SCHED\_FIFO and creating a thread*

**Description of code**

The code at basic point is creating a single thread(process) and assigning scheduling policy FIFO. The main aim of the code us to check the consistency and accuracy of Real time clock of Linux.

If RUN\_RT\_THREAD is **undefined** a process/function delay\_test is called which checks without creating a thread and main is by default executed in SCHED\_OTHER policy. The Delay error of RT\_CLOCK is observed to be nearly around **83900 ns** as seen in Fig3.1**.**

If RUN\_RT\_THREAD is **defined** a main\_thread is created and scheduling parameter attribute is set to SCHED\_FIFO policy (which is FIFO scheduling policy). The delay test function call is assigned to this thread and Delay error of RT\_CLOCK is observed to be nearly around **34450 ns** as seen in Fig3.2.

**Observations**

(i)An error of multiple of ten thousand of nano seconds can be seen real time clock in Ubuntu (executed on Altera DE1 SoC) which may have **catastrophic** effects when accumulated over time for real hard time systems. As ten thousands of nanoseconds is micro seconds of deviation from actual deadline in real time which would add up to milli seconds at later point.

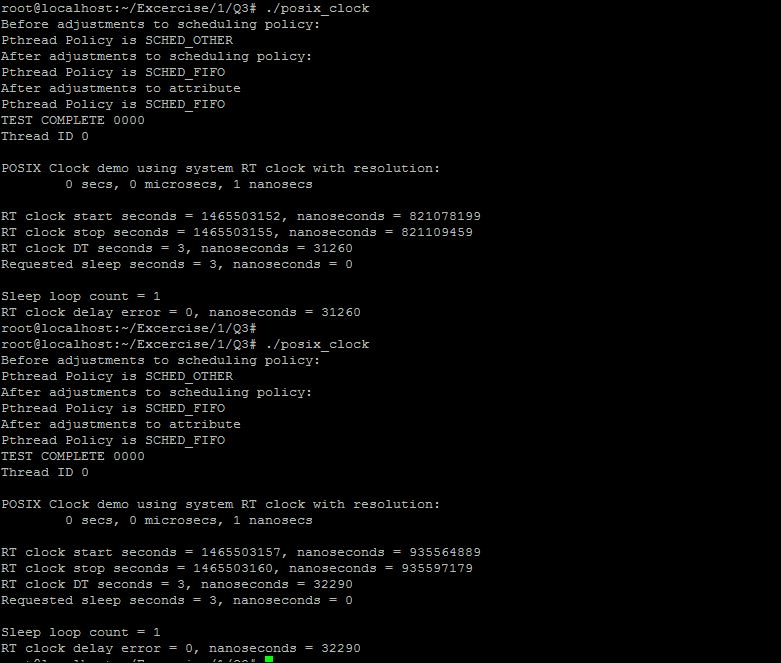
(ii)One interesting observation is error is varying by approx. 40000 ns when scheduling policy has been assigned. The RT Clock is more accurate with SCHED\_POLICY. I can’t analyze the reason for the same.

(iii)Importance of

1. Interrupt handler Latency[2] – It can be defined as the time from which an interrupt is asserted and Program counter is vectored to an interrupt handler . It is time between interrupt occurrence and execution of first ISR instruction. This latency can depend on time for saving the PC in stack, having registers (saving a context), hardware delay, other interrupt (ISR) execution etc. Lower latency leads to more deterministic results of real time hard deadlines.
2. Low Context Switch – It is time taken to a run new thread and save the context (PC and registers) of presently running thread which is preempted by new thread, so when it is dispatched again it can be start running from previous preempted point. It is significant to this machine code instructions (Context overhead) to meet deadlines accurately and deterministically. Keeping overhead value less helps to reduce drift in execution time
3. Service Timers – The accuracy of Stable timers provides perfect timings thus allowing no deviation from course of thread and right time of preemption. This further reduces drift and jitter. Further, more the timers are unstable, it would lead incorrect and miss of deadlines. Even if there are issues with timers and are deterministic (known and constant time of drift) it can be corrected while designing the system.

(iv)Accuracy of RT Clock Code –

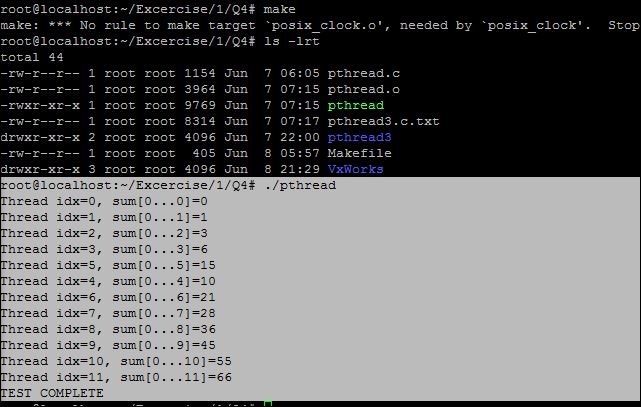
As Can be seen from Fig 3.3 and observation (ii) there is variation in execution of time thus changing values of delay in thousands of nano seconds. It uses nanosleep function call to suspend thread and clock\_gettime which may be causing variations. The inaccuracy in timer services cause jitter in timing leading to inaccuracy in RT Clock analysis code. The context switch time variation also would be leading to this issues . Therefore, the Real Time Clock tested on this system is less accurate in my belief as above considerations are not incorporated.



*Fig 3.3 Screen shot of execution of RT Clock Code using SCHED\_FIFO and variations in delay error*

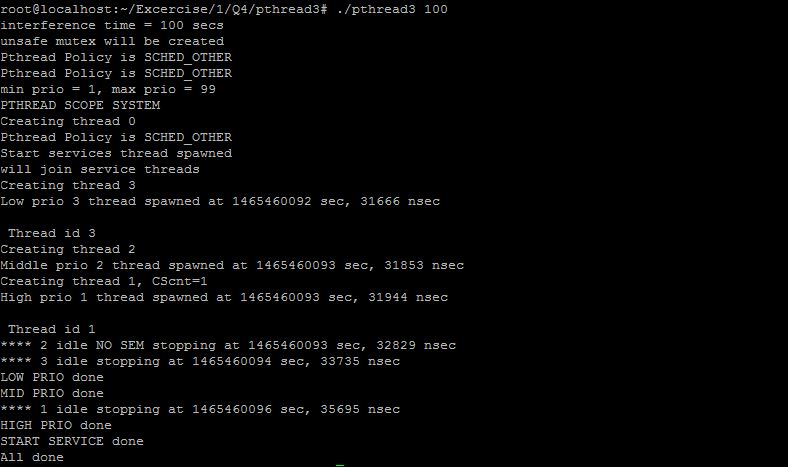
4.

(i) Running Simple Thread pthread.c code on Ubuntu hosted on Altera DE1-SoC



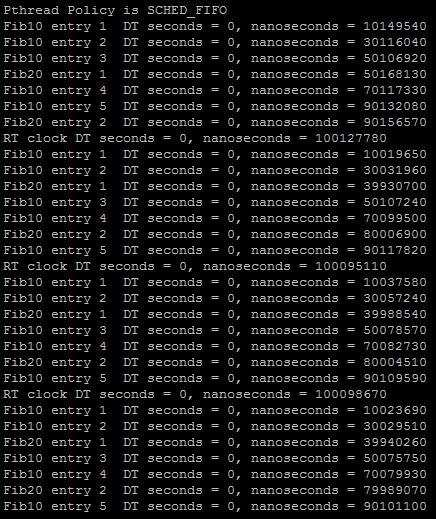
*Fig 4.1 Screen shot of execution of Simple Pthread*

(ii) Running Sync Thread (pthread3.c) code on Ubuntu hosted on Altera DE1-SoC



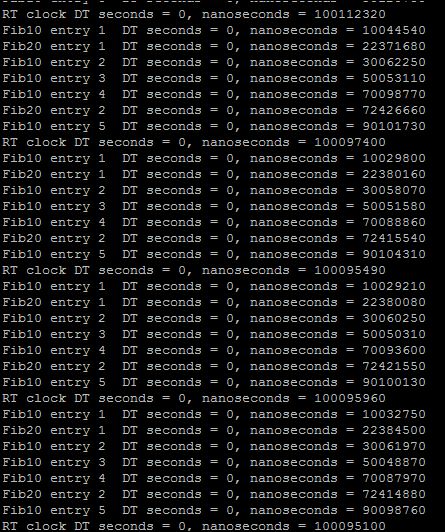
*Fig 4.2 Screen shot of execution of Sync Pthread*

(ii) Running Binary Semaphore Linux code (binSemLinux.c) ported from VXwroks LCM variant implementation on Ubuntu hosted on Altera DE1 SoC. Please find the code and makefile attached with submission.



*Fig 4.3 Screen shot of execution of Binary Semaphore Pthread (Linux) using affinity(Set one core )*****

*Fig 4.4 Screen shot of execution of Binary Semaphore Pthread (VxWorks) source from Question*



*Fig 4.3 Screen shot of execution of Binary Semaphore Pthread (Linux) using set using multiple core*

**References**

[1] Lui and Layland Paper <http://ecee.colorado.edu/~ecen5623/ecen/rtpapers/archive/PAPERS_READ_IN_CLASS/liu_layland.pdf>

[2] Real Time Embedded Components and Systems by Sam Siewert and John Pratt