ECEE 5623, Real-Time Systems:

Exercise #1 – Invariant LCM Schedules

DUE: As Indicated on Canvas

Please thoroughly read Chapters 1 & 2 in RTECS with Linux and RTOS

Please see example code provided - Linux, and if interested, FreeRTOS, VxWorks, and Zephyr

This lab is written **to be completed with embedded Linux running on the R-Pi3b (order or borrow and use NOOB image). It is possible to also use a Jetson Nano (Getting started)**, Jetson TK1/TX1/TX2 (JetPack install) or Beagle board(s), and while the course is Linux focused, it is also possible to complete it using FreeRTOS, VxWorks or Zephyr as an option. Note that you will either have to adapt the example code or use equivalent examples from the links above for RT-Clock, use of tasks in place of pthreads, and the appropriate board to boot FreeRTOS, VxWorks, or Zephyr as described on the course main web page. **The standard final project requires use of a camera, which is simple with embedded Linux using the UVC driver**, so for other options such as Zephyr or VxWorks, this may be an issue, unless a suitable camera interface can be found, and will require proposal of an alternative creative project, or switching back to embedded Linux.

Exercise #1 Requirements:

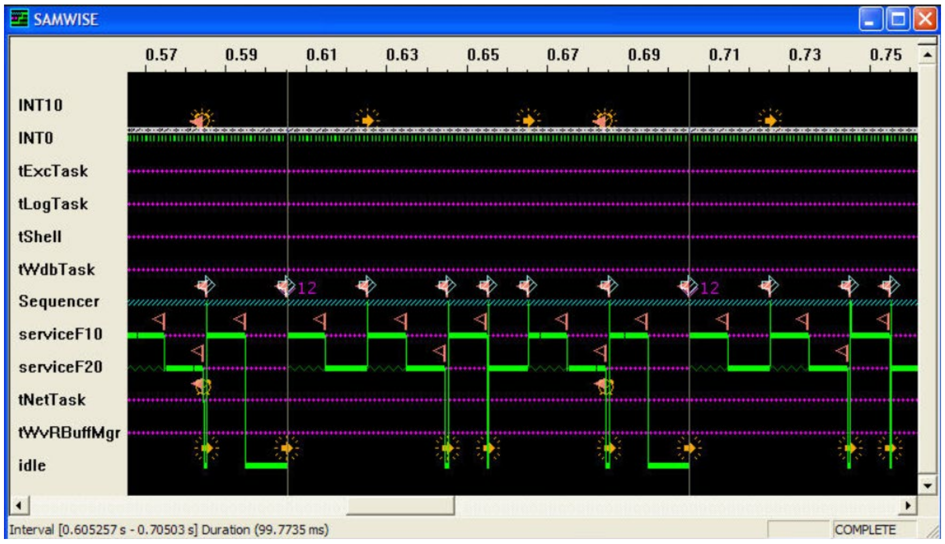
1) [15 points] The Rate Monotonic Policy states that services which share a CPU core should multiplex it (with context switches that preempt and dispatch tasks) based on priority, where highest priority is assigned to the most frequently requested service and lowest priority is assigned to the least frequently requested AND total shared CPU core utilization must preserve some margin (not be fully utilized or overloaded). Draw a timing diagram for three services S_1, S_2, and S_3 with $T_1=3$, $C_1=1$, $T_2=5$, $C_2=2$, $T_3=15$, $C_3=3$ where all times are in milliseconds. [Note that you can find examples of timing diagrams in Lecture and here – note that we have not yet covered dynamic priorities, just RM fixed policy described here, so ignore EDF and LLF for now]. Label your diagram carefully and describe whether you think the schedule is feasible (mathematically repeatable as an invariant indefinitely) and safe (unlikely to ever miss a deadline). What is the total CPU utilization by the three services?

2) [15 points] Read through the Apollo 11 Lunar lander computer overload story as reported in RTECS Notes, based on this NASA account, and the descriptions of the 1201/1202 events described by chief software engineer Margaret Hamilton as recounted by Dylan Matthews. Summarize the story. What was the root cause of the overload and why did it violate Rate Monotonic policy? Now, read Liu and Layland's paper which describes Rate Monotonic

policy and the Least Upper Bound – they derive an equation which advises margin of approximately 30% of the total CPU as the number of services sharing a single CPU core increases. Plot this Least Upper bound as a function of number of services and describe 3 key assumptions they make and document 3 or more aspects of their fixed priority LUB derivation that you don't understand. Would RM analysis have prevented the Apollo 11 1201/1202 errors and potential mission abort? Why or why not?

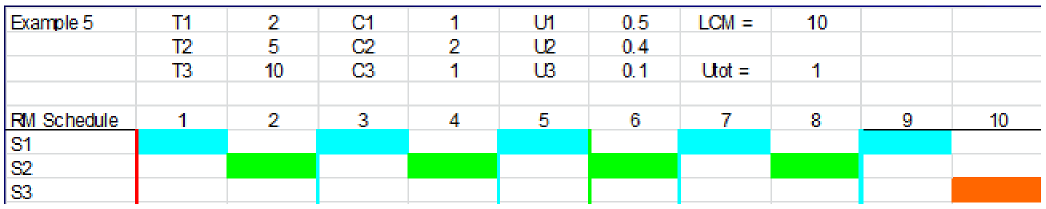
3) [20 points] Download **RT-Clock** and build it on an R-Pi3b+ or Jetson and execute the code. Describe what it's doing and make sure you understand clock_gettime and how to use it to time code execution (print or log timestamps between two points in your code). Most RTOS vendors brag about three things: 1) Low Interrupt handler latency, 2) Low Context switch time and 3) Stable timer services where interval timer interrupts, timeouts, and knowledge of relative time has low jitter and drift. Why are each important? Do you believe the accuracy provided by the example RT-Clock code?

4) [30 points] This is a challenging problem that requires your to learn quite a bit about pthreads in Linux and to implement a schedule that is predictable. Download, build and run code in http://ecee.colorado.edu/~ecen5623/ecen/ex/Linux/simplethread/ and based on the example for creation of 2 threads provided by incdecthread/pthread.c, as well as testdigest.c with use of SCHED_FIFO and sem_post and sem_wait as well as reading of POSIX manual pages as needed - describe how you would attempt to implement Linux code to replicate the LCM invariant schedule implemented in the VxWorks RTOS which produces the schedule measured using event analysis shown below:



Interval [0.605257 s - 0.70503 s] Duration (99.7735 ms)

The observed timing above fits our theory for RM policy on a priority preemptive scheduling system as shown by the timing diagram below:



Example 5	T1	2	C1	1	U1	0.5	LCM =	10		
	T2	5	C2	2	U2	0.4				
	T3	10	C3	1	U3	0.1	Utot =	1		
RM Schedule	1	2	3	4	5	6	7	8	9	10
S1	🟦		🟦		🟦		🟦		🟦	
S2		🟩		🟩		🟩		🟩		
S3										🟧

You description should outline how you would implement code equivalent to the VxWorks synthetic load generation and schedule emulator. Code the Fib10 and Fib20 synthetic load generation and work to adjust iterations to see if you can at least produce a reliable 10 millisecond and 20 millisecond load on ECES Linux or a Jetson system (Jetson is preferred and should result in more reliable results). Describe whether your able to achieve predictable reliable results in terms of the C (CPU time) values alone and how you would sequence execution.

Hints – You will find the LLNL (Lawrence Livermore National Labs) pages on pthreads to be quite helpful.

If you really get stuck, a detailed solution and analysis can be found here, but if you use, be sure to cite and make sure you understand it and can describe well. If you use this resource, not how similar or dissimilar it is to the original VxWorks code and how predictable it is by comparison.

[20 points] Overall, provide a well-documented professional report of your findings, output, and tests so that it is easy for a colleague (or instructor) to understand what you've done. Include any C/C++ source code you write (or modify) and Makefiles needed to build your code and make sure your code is well commented, documented and follows coding style guidelines. I will look at your report first, so it must be well written and clearly address each problem providing clear and concise responses to receive credit.

Note: Linux manual pages can be found for all system calls (e.g. fork()) on the web at http://linux.die.net/man/ - e.g. http://linux.die.net/man/2/fork

In this class, you'll be expected to consult the Linux manual pages and to do some reading and research on your own, so practice this in this first lab and try to answer as many of your own questions as possible, but do come to office hours and ask for help if you get stuck.

Upload all code and your report completed using MS Word or as a PDF to Canvas and include all source code (ideally example output should be integrated into the report directly, but if not, clearly label in the report and by filename if test and example output is not pasted directly into the report). ***Your code must include a Makefile so I can build your solution on an embedded Linux system (R-Pi 3b+ or Jetson). Please zip or tar.gz your solution with your first and last name embedded in the directory name and/or provide a GitHub public or private repository link. Note that I may ask you or SA graders may ask you to walk-through and explain your code. Any code that you present as your own that is "re-used" and not cited with the original source is plagiarism. So, be sure to cite code you did not author and be sure you can explain it***

in good detail if you do re-use, you must provide a proper citation and prove that you understand the code you are using.

Grading Rubric

[15 points] Rate Monotonic Analysis and Timing Diagrams:

[5 points] Correct diagram ____

[5 points] Feasibility and safety issues articulated _____________________________

[5 points] Utility and method description ___________________________________

[15 points] Shared CPU system overload:

[3 pts] Apollo 11 reading and summary ______________________________________

[3 pts] Root cause analysis and description ___________________________________

[3 pts] RM LUB plot and description of margin _______________________________

[6 pts] Arguments for and against RM policy and analysis prevention of Apollo 11 overload scenario (at least 2 main arguments and points) _________________________

[20 points] POSIX Real-Time Clock:

[5 pts] Download, build, run as is ____

[5 pts] Description of code as is ____

[5 pts] What is value of each RTOS bragging point? _____________________________

[5 pts] Determination and argument for or against accuracy of RT clock on system tested

____

[30 points] Pthread download, build, and analysis of features:

[5 pts] Download, build, run simple thread code__________________________________

[5 pts] Download, build, run example-sync code _________________________________

[10 pts] Description of key RTOS / Linux OS porting requirements _________________

(3 pts) Threading vs. tasking ____

(3 pts) Semaphores, wait and sync ______________________________________

(4 pts) Synthetic workload generation ___________________________________

[5 pts] Synthetic workload analysis and adjustment on test system __________________

[5 pts] Overall good description of challenges and test/prototype work _______________

. [20 points] Quality of reporting and code quality and originality:

[10 pts] Professional quality of reporting, testing and analysis (0…6 is below average, 7 is average, 8 is good, 9 excellent, and 10 is best overall.)____

[10 pts] Code quality including style, commenting, originality, proper citation for re-used code, modified code, etc.____