ECEN 5623, Real-Time Systems:

Exercise #3 – Threading/Tasking and Real-Time Synchronization

DUE: As Indicated on Canvas and in class

Please thoroughly read Chapters 6, 7 & 8 in the text.

Please see example code provided - http://mercury.pr.erau.edu/~siewerts/cec450/code/

Please complete this lab on an Altera DE1-SoC, Raspberry Pi or Jetson and provide evidence that you did your work with screenshots.

Exercise #3 Requirements:

1) [10 points] Read Sha, Rajkumar, et al paper, "Priority Inheritance Protocols: An Approach to Real-Time Synchronization" and summarize 3 main key points the paper makes. Read Dr. Siewert's summary paper on the topic as well. Finally, read the positions of Linux Torvalds as described by Jonathan Corbet and Ingo Molnar and Thomas Gleixner on this topic as well. Take a position on this topic yourself and write at least one well supported paragraph or more to defend your position based on what we have learned in class. Does the PI-futex (Futex, Futexes are Tricky) that is described by Ingo Molnar provide safe and accurate protection from un-bounded priority inversion as described in the paper? If not, what is different about it?

2) [25 points] Review the terminology and describe clearly what it means to write "thread safe" functions that are "re-entrant". There are generally three main ways to do this: 1) pure functions that use only stack and have no global memory, 2) functions which use thread indexed global data, and 3) functions which use shared memory global data, but synchronize access to it using a MUTEX semaphore critical section wrapper. Describe each method and how you would code it and how it would impact real-time threads/tasks. Now, using a MUTEX, provide an example using RT-Linux Pthreads that does a thread safe update of a complex state with a timestamp (pthread_mutex_lock). Your code should include two threads and one should update a timespec structure contained in a structure that includes a double precision attitude state of {X,Y,Z acceleration and Roll, Pitch, Yaw rates at Sample_Time} (just make up values for the navigational state and see http://linux.die.net/man/3/clock_gettime for how to get a precision timestamp). The second thread should read the times-stamped state without the possibility of data corruption (partial update).

3) [15 points] Download http://mercury.pr.erau.edu/~siewerts/cec450/code/example-sync/ and describe both the issues of deadlock and unbounded priority inversion and the root cause for both in the example code. Fix the deadlock so that it does not occur by using a random back-off scheme to resolve. For the unbounded inversion, is there a real fix in Linux – if not, why not? What about a patch for the Linux kernel? For example, Linux Kernel.org recommends the RT_PREEMPT Patch, but would this really help? Read about the patch and describe why think it would or would not help with unbounded priority inversion. Based on inversion, does it make sense to simply switch to an RTOS and not use Linux at all for both HRT and SRT services?

4) [15 points] Review heap_mq.c and posix_mq.c. First, re-write the VxWorks code so that it uses RT-Linux Pthreads (FIFO) instead of VxWorks tasks, and then write a brief paragraph describing how these two message queue applications are similar and how they are different. You may find the following Linux POSIX demo code useful, but make sure you not only read and port the code, but that you build it, load it, and execute it to make sure you understand how both applications work and prove that you got the POSIX message queue features working in Linux on your Altera DE1-SoC or Jetson. Message queues are often suggested as a way to avoid MUTEX priority inversion. Would you agree that this really circumvents the problem of unbounded inversion? If so why, if not, why not?

5) [35 points] Watchdog timers, timeouts and timer services – First, read this overview of the Linux Watchdog Daemon and describe how it might be used if software caused an indefinite deadlock. Next, to explore timeouts, use your code from #2 and create a thread that waits on a MUTEX semaphore for up to 10 seconds and then un-blocks and prints out "No new data available at <time>" and then loops back to wait for a data update again. Use a variant of the pthread_mutex_lock called pthread_mutex_timedlock to solve this programming problem.

Grading Rubric

[10 points] Priority inheritance paper review and unbounded priority inversion issues:

[4 points] Three Priority inheritance key points made in paper

_____

[3 points] Why the Linux position makes sense or not_____

[3 points] Reasoning on whether FUTEX fixes_____

[25 points] Thread safety with global data and issues for real-time services:

[5 pts] Description of methods _____

[5 pts] Impact of each to real-time services_____

[15 pts] Correct shared state update code_____

[15 points] Example synchronization code using POSIX threads:

[5 pts] Demonstration and description of deadlock with threads

_____

[5 pts] Demonstration and description of priority inversion with threads

_____

[5 pts] Description of RT_PREEMPT_PATCH and assessment of whether Linux can be made real-time safe _____

[15 points] Example synchronization code using POSIX message queues and threads:

[10 pts] Demonstration of message queues on Altera DE1-SoC adapted from examples

_____

[5 pts] Description of how message queues would or would not solve issues associated with global memory sharing

_____

[35 points] Watchdog timers (for the system) and timeouts (for API calls):

[10 pts] Description of how the Linux WD timer can help with recovery from a total loss of software sanity (E.g. system deadlock)

_____

[25 pts] Adaptation of code from #2 MUTEX sharing to handle timeouts for shared state

_____

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. 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 D2L 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 the TAs can build your solution on Ubuntu VB-Linux, a Jetson or a Altera DE1-SoC. Please zip or tar.gz your solution with your first and last name embedded in the directory name.***