EE 6314 Advanced Embedded Microcontroller System Design
Project 1 – Design of a Real-time Operating System
Spring 2018

1 Overview

The goal of this project is write an RTOS solutions for an M4F controller that implements that implements a preemptive RTOS solution with support for semaphores, yielding, sleep, priority scheduling, priority inheritance, and a shell interface.
.
A simple framework for building the RTOS is included in the rtos.c file.

2 Requirements

Scheduler:
Each time the scheduler is called, it will look at all ready processes and will choose the next process. Modify the scheduler to add prioritization to 8 levels.

Note: The method used to prioritize the relative importance of the tasks is implicit in the assignment of the prioritization when createThread() is called.

Kernel Functions:
Add a function yield() that will yield execution back to the kernel that will store the return address and save the context necessary for the resuming the process later.

Add a function sleep(time_ms) and supporting kernel code that will store the return address and save the context necessary for the resuming the process later. The process is then delayed until a kernel determines that a period of time_ms has expired. Once the time has expired, the process that called sleep(time_ms) will be marked as ready so that the scheduler can resume the process later.

Add a function wait(semaphore) that causes a process to block until a resource or resources is available. The waiting process will be recorded in the semaphore process queue.

Add a function post(semaphore) and supporting kernel code as discussed in the lectures.

Modify the function createThread() to store the process name and initialize the process stack as needed.

Add a function destroyThread() that removes a process from the TCB and cleans up all semaphore entries for the process that is deleted.

In implementing the above kernel functions, code the function systickIsr() to handle the sleep timing and kernel functions.. The code to switch task should reside in the pendSvcIsr() function..

Add a shell process that hosts a command line interface to the PC. The command-line interface should support the following commands (borrowing from UNIX):

ps: The PID id, process name, and % of CPU time should be stored at a minimum.

ipcs: At a minimum, the semaphore usage should be displayed.

kill <PID>:This command allows a task to be killed, by referencing the process ID.

reboot: The command restarted the processor.

pidof <Process_Name> returns the PID of a process

<Process_Name> & starts a process running in the background if not already running. Only one instance of a named process is allowed.

3 Hints

You should start with the rtos.c code provided in class and modify the program as appropriate to test operation. No code from any other source other than the current semester's discussions, including last semester's solutions and prior student projects, may be included. If you reference a small snippet of code from a book, you must clearly reference the work and page number.

It is recommended that you start a cooperative design with the yield() function only and determine the mechanism required to record the context for the current process, call the scheduler, and then restore the context of a process that was selected by the scheduler.

Once this is complete, code the sleep() function. Also add any interrupt code needed to handle the pending timer(s).

Now the wait() and post() functions can be added, which will also reuse the context saving code above. The post() function needs to be carefully examined when called from an interrupt.

Next add the UART shell processing to your example. It is recommended that you avoid the C string libraries when possible.

Next, add preemption to the timer interrupt and make changes in the kernel functions as needed to be compatible with the extra registers stored by the interrupt routine.

4 Deadlines and Teams

The project is an individual project. All work should be your own. No material from prior semesters should be used. The shell interface solution should use code based on steps 2-4 of your EE5314 project.

All work should be completely original and should not contain code from any other source, except the rtos.c file framework. Do not rename any of the existing functions or change priorities as this can affect the grading process, which requires these names be preserved for extraction.

When submitting your project, you should submit only one C file with the name indicated in the rtos.c file. Any other files will be ignored. This may be poor programming style, admittedly, but this is a speed up the code review. Specifically, do not include the interrupt vector table section (#pragma DATA_SECTION(g_pfnVectors, ".intvecs")) in these files.

If any part of your project (even 3 lines of code) is determined to not be unique, your grade will be impacted.

Please include your name clearly at the top of your program (rtos.c). Please document your program well to maximize credit.

Projects are due at the time indicated in the class syllabus with an oral defense at that time. No late projects will be accepted.

Have fun!