Introduction

An alternative to sequentially running tasks is to pre-empt a task to run a new (more important) task. To be able to pre-empt a task, a context switch has to be implemented. Gladly, this has already been done by the instructor!

Assignments week 4 – Pre-emptive scheduler

This week you will learn how to:

- understand the context switch;
- implement privilege modes;
- create a system interface using supervisor calls;
- implement a priority based scheduler.

4.1 The teacher has built a simple preemptive OS that is still missing some important features. In these assignments you'll implement some extra features and gain a bigger understanding in how an OS operates. Next week we'll start using a fully developed RTOS with all necessary features for a production environment.

A Download the project VersdOS.zip.

B Import this project into STM32CubeIDE with the menu-option File > Import..., General > Projects from Folder or Archive, Next, Archive, select the file `VersdOS.zip`, Open and Finish. Build and debug the project. The LEDs should blink.

C Browse through the code and especially make sure you understand the scheduling process and the context switch.

D Measure the periods at which the LEDs toggle using a logic analyzer and explain why this is not 2 × but 8 × the `blocking_delay` time.

4.2 The OS currently uses a *blocking* delay. While usually this would form a problem by delaying everything, this pre-empting OS still allows other tasks to get a chance to run. It is however very wasteful with CPU-cycles and thus energy. The goal of this assignment is to implement a non-blocking delay.

A Implement a non-blocking delay so that a task can request the OS to be kept out of the scheduling loop for a certain number of system ticks. The scheduler should remain preemptive and perform round-robin on all the *available*(not delayed) tasks. Once the requested number of system ticks have passed, the scheduler should include the task in the selection process. You may use the `taskYield()` function to let the OS know a task is ready to be switched out.

B Measure the periods of the LEDs using a logic analyzer and explain the results.

4.3 The current OS doesn't utilize the privilege modes that the ARM core offers.

A Try changing the SysTick period to 2 ms within a task. A task can change the SysTick period to any value it wants, but you can imagine this isn't desired behavior.

B Implement unprivileged mode for the tasks. When succeeded the previous test should fail and be caught in an infinite ISR fault-handler.

4.4 Tasks are no longer able to access the SysTick peripheral. It is time to implement a system call using the SVC interrupt mechanism. The `taskYield()` function would also classify as a system call. Write a system call function that will allow a task to change the SysTick period to a value between 1 and 10 ms. The system call should utilize the SVC mechanism by passing a different number than the `taskYield()` call. Make sure both calls keep working! Within the SVC interrupt service routine you will have to find the used SVC instruction using the PC value and look at the LSB to determine the system call number. You can also utilize the tasks' stack to find the passed SysTick period parameter. When this parameter is invalid the system call should simply return.

4.5 The current scheduler implements a round-robin algorithm. In a real-time situation it would be necessary to give tasks different priorities in order to make important deadlines.

A In reality tasks have more important jobs to do than just toggling a LED. This means tasks will take time. Use the original blocking delay function to give tasks a realistic waiting time in a random amount of milliseconds. In order to verify timing requirements with a logic analyzer, you can toggle the LED at the end of the busy waiting period.

B Implement a priority based scheduling algorithm. When adding tasks you should be able to pass the priority. When there are multiple tasks available with the same priority a round-robin algorithm should be applied. Test this functionality for multiple tasks with and without non-blocking delays.

C The round-robin period thus far equaled the SysTick period. Implement the ability to have a task-dependent round-robin period. E.g. if taskA and taskB have equal priorities, TaskA will run 2 SysTicks, taskB will need 5 SysTicks. This period should also be something passed at task creation. Verify this functionality with a logic analyzer.