Porting FreeRTOS to MikroC for STM32 M3

This project is my first attempt of porting FreeRTOSV7.1.1 to CORTEX STM32F107 MikroC Version 2.5. Please feel free to experiment and let me know about your opinion.

- Unzip archive in root folder C:\. If you prefer any other folder, you can unzip it there, but you have to adjust paths by Shift-Ctrl-P.
- Double click on RTOS.mcpar.
- Led_Blinking 2.0 source should appear. Press Ctrl+F11 to build executable RTOS.hex and to program the device.
- LEDs should start blinking immediately after programming is finished.

About Led_Blinking 2.0

Although the visual effect is identical as in Led_Blinking demo from Mikroelektronika, operation is slightly different.

This example is implemented as a task which blocks itself for the certain period of time. During that time, the idle task is executed since it has lower priority. After the time period is expired, our task is unblocked and put in ready queue with higher priority. This immediately preempts the idle task, and our example is run again. Note that priority of our task is only one step above lowest (idle) priority.

However, if ordinary Dealy_ms() is used, Led_Blinking 2.0 becomes Led_Blinking 1.0, i.e. our example is executed spending most of the time in the busy loop, without yielding processor to the lower priority task. Luckily, it is an idle task, but if it was not, this could lead to the starvation of the lower priority task.

Idle task is created immediately after starting the scheduler and, in fact, it can execute some useful (background) code, provided it never blocks. There is facility for this in FreeRTOS which can be enabled by setting `configUSE_IDLE_HOOK = 1.`

Necessary modification in FreeRTOS core files

Due to the limitation of Mikroelektronika C compiler, I was forced to modify two core files: queue.c and tasks.c. All modifications are marked by “S.M.” (my initials). Off course, with a little help of Mikroelektronika, this can be avoided, but for now, this is only the proof that it is feasible.

Compiler limitations in this design are inability to handle `(void) variable` casting and some problems with multiple macro expansions. First problem was solved by commenting out such variables (they do nothing anyway), and the second, by manual replacing (“expanding”) macros.

Assembler was also very restrictive, but it did the job.