AGH	*Akademia Górniczo-Hutnicza w Krakowie*		
Faculty: ESA	Academic Year 2019/2020	Year of study	Fields of science:
Student name:	**Ioannis Manousaridis**		
Course:	**Real Time Operating Systems in Control Applications**		
Exercise nr:	**07**		
Exercise subject:	**FreeRTOS real time operating system - basics**		
Lecturer: dr inż. Grzegorz Wróbel			
Date: 00.00.2020			Grade:

1 Exercise purpose

The aim of the exercise is to get familiar with the visual studio express, learn the basics of FreeRTOS real-time operating system architecture and the principles of creating and running applications in its environment in windows 10 operating system.

2 Assumptions / Theory

An Embedded Operating System like FreeRTOS is nothing but software that provides multitasking facilities. FreeRTOS allows to run multiple tasks and has a simple scheduler to switch between tasks. In this lab, the focus is on tasks' functions.

In FreeRTOS, the user has precise control of when tasks will use the CPU. The rules are simple:

- Task with highest priority will run first, and never give up the CPU until it sleeps.
- If 2 or more tasks with the same priority do not give up the CPU (they don't sleep), then FreeRTOS will share the CPU between them (time slice).

3 Description of implementation

The task.h library is used for dealing with tasks in FreeRTOS. In the begining two different tasks are created, vStartTask_1 and vStartTask_2. Both of them have a similar concept. They print a text after a certain amount of time passes and then they terminate. Task 1 will print on screen every 1000 clock ticks and will terminate after 30 repeations and task 2 will print on screen every 2000 clock ticks and will terminat after 20 repeations. The

termination-deletion of the tasks is possible by using the handlers which are initialized as global variables.

Afterward, the main function is being executed. The The main function in FreeRTOS based project is nothing but a function that creates the tasks. FreeRTOS lets multi-task based on the different tasks and their priority. In this case, both tasks are created with the same priority and they are allocated with the minimum amount of memory in the stack memory.

Then the vTaskStartScheduler functions is being called. In general, Typically, before the scheduler has been started, main() (or a function called by main()) will be executing. After the scheduler has been started, only tasks and interrupts will ever execute. Starting the scheduler causes the highest priority task that was created while the scheduler was in the Initialization state to enter the Running state.

The scheduler creates also the IDLE task when it starts. If the heap memory overflows during the execution of the stacks, the scheduler will terminate and an appropriate message will be printed on the screen letting know the user about the insufficient memory in the heap.

```
// Scheme of the program

Task_1(){
        //DO SOMETHING HERE
}

Task_2(){
        //DO SOMETHING HERE
}

int main()
{
    Create_task1()
    Create_task2()

vTaskStartScheduler();

print("The scheduler failed. Insufficient FreeRTOS heap memory");
    Return 0;
}
```

4 Conclusions

Getting familiar with the priniciples of FreeRTOS and the tasks' function does not require a lot of time. Basic knowlegde of C language is a requirement though. Although this lab is simple, it's a good example to start thinking about the memory that can be caused in a rtos and make the user responsible for the future and teach how to manage the memory issues.