LINGI 2315 - Homework 4 - My RTOS App 2 **P1**

Name	Delcoigne Ben	Noma	38771700

Description of the first configuration, the problem and the implementation of the Starvation mechanism with relevant screenshots of your code

I created a task that just loops and prints out things in the console. The problem is that, if this task has a higher priority and never “sleeps”, no other task can run. Resulting in only “inf loop” being written in the console. (this obviously occurs only when the same core is shared for all tasks)

```
/* ------------------------------
void Task_Tutorial(void){
    for(;;){
        MTXLOCK_STDIO();
        printf("Inf loop");
        MTXUNLOCK_STDIO();
    }
}
```

(this task is set at a priority 0 while all others are set at 1)

In order to solve that, we must starve the task. This is explained in mabassi ’s user guide page 66: we must define a starving time (and priority) to each task. Here i allocate each task 5 os tick times before looking at the other tasks if anything is to run:

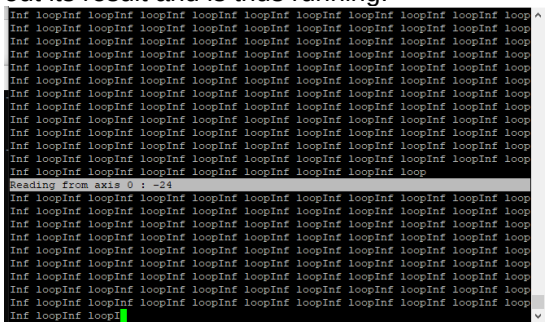
```
/* ------------------------------------------------ */
/* Application set-up                               */
    Task = TSKcreate("Task Tutorial", 0, 8192, &Task_Tutorial, 0)

    if(OX_STARVE_RUN_MAX != 0){
        TSKsetStrvRunMax(Task,5* OS_TICK_PER_SEC);
    }
    if(OX_STARVE_PRIO != 0){
        TSKsetStrvPrio(Task, 1);
    }

    TSKsetCore(Task, 1);                         /* Create new
    TSKresume(Task);                             /* when BMP (
```

This is done for every task

The result is good: we see in the console that another task (gsensor reading from previous homework) is able to print out its result and is thus running:



This shows that even though the infinite loop has infinite priority, once is has run for 5 os tick times, other tasks get a chance to run (thus main task is starved).

This demonstrates the problem being solved in my opinon.

LINGI 2315 - Homework 4 - My RTOS App 2 **P2**

Name	Delcoigne Ben	Noma	38771700

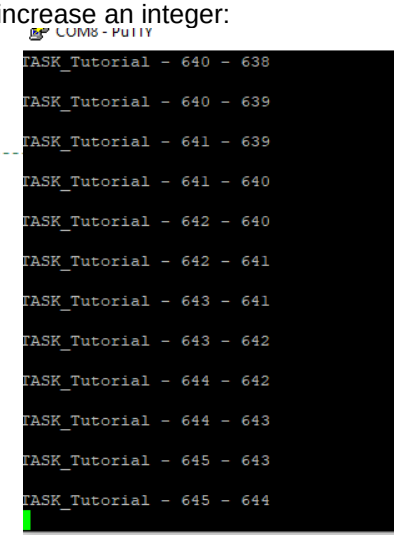
Description of the second configuration, the problem and the implementation of the Round Robin mechanism with relevant screenshots of your code

As opposed to the starvation problem where one task of higher priority takes the ressources of all other tasks, we might have a problem when ressources are shared evenly with same priority tasks. Indeed, tasks of same priority get an even portion of time (round robin) to run.

To demonstrate this problem, we must find a way to measure how long a task is being run compared to another task.

In order to do that, I implemented counters in two tasks that just increase an integer:

```
/* ---------------------------------------------------------------------------
int int1=0;
void Task_Tutorial(void){
    for(;;){
        int1 +=1;
        MTXLOCK_STDIO();
        printf("\nTASK_Tutorial - %d - %d \n", int1, int2);
        MTXUNLOCK_STDIO();
    }
}
int int2=0;
void Task_Tutorial2(void){
    for(;;){
        int2 +=1;
        MTXLOCK_STDIO();
        printf("\nTASK_Tutorial - %d - %d \n", int1, int2);
        MTXUNLOCK_STDIO();
        }
    }
```



We notice equal numbers (approx), which means each task has the same priority. We will change the “fraction of time” they recieve in round robin using: (note I had to remove the mutexes otherwise it doesn’t work since one task waits the other

```
Application set-up                                  */
  Task = TSKcreate("Task Tutorial", 0, 8192, &Task_Tutorial, 0);

  /*if(OX_STARVE_RUN_MAX != 0){
      TSKsetStrvRunMax(Task,5* OS_TICK_PER_SEC);
  }
  if(OX_STARVE_PRIO != 0){
      TSKsetStrvPrio(Task, 1);
  }*/
  if(OX_ROUND_ROBIN !=0){
      TSKsetRR(Task, 1*OS_TICK_PER_SEC);
  }

  TSKsetCore(Task, 1);                          /* Create new task, will always run on core #1  */
  TSKresume(Task);                              /* when BMP (G_OS_MP_TYPE == 4 or 5)            */
```

When doing so, with different round-robin timers (ratio of 2-1), we notice a difference between the counters which indeed shows both tasks get different time shares now.