****

**Universidad Autónoma de Guadalajara**

Biomedical Engineering

Peripherals and Interfaces

Jesús Arnoldo Zerecero Nuñez, 2885993

Edwin Francisco Zamora Oros, 2874382

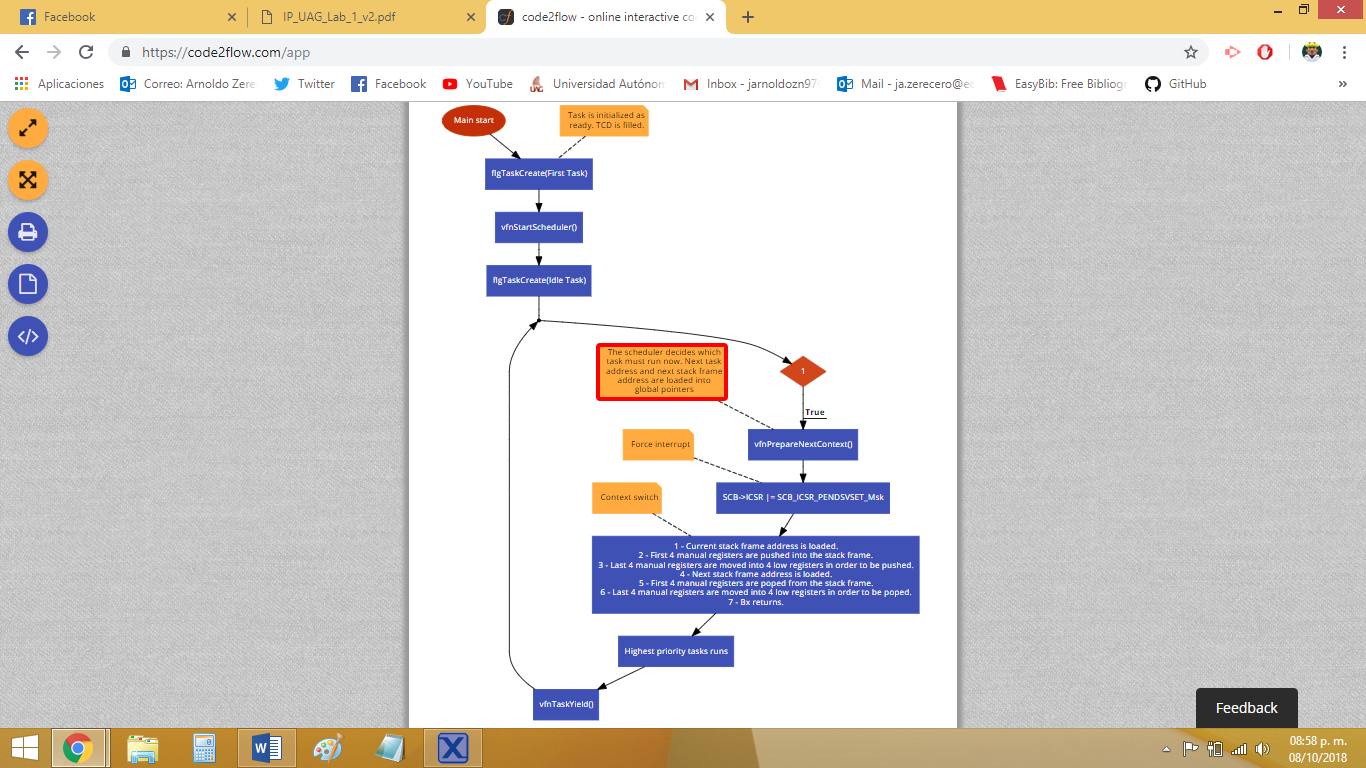
*“Lab 1: Scheduler (part 1)”*

8/10/18

**Introduction:**

In this first practice, the very basic APIs of a priority-based scheduler were created and implemented in a simple 4 task environment. In addition, context switching was implemented, to switch between these 4 tasks according to their priority. These functions where implemented:

* A function to create a task and set is as ready.
* A function to create a task and set is as blocked.
* A function to make a task relinquish the CPU (trigger a context switch).
* A function to set a task as ready.
  + A function to place a task into the ready queue.
  + A function to remove a task from the ready queue.
* A function to start the scheduler.
* A function to decide which task is next (based on priority).
* An exception for control switching.

**Flowchart:**

**Development (issues):**

* **Understading ARM Cortex M0 Arquitecture for coding in assembly.**
  + This was the major problem at first. Context switching must be done using assembly language. Every major problem we found was derived from our inexpertise in this language.
  + Every basic aspect of the Cortex M0 had to be learned before even thinking about context switching.
    - PSP & MSP management.
    - Exception returns.
    - xPSR behavior.
    - How the CPU manages the core registers by itself.
    - How pointers are addressed in memory.
    - And much more!
* **Masking the ICSR in order to force a PendSV interruption.**
  + We had lots of trouble figuring out why sometimes the line of code which masks ICSR to force a context switch (SCB -> ICSR |= SCB\_ICSR\_PENDSVSET\_Msk) would sometimes get skipped over.
  + We struggled to understand that this is an asynchronous interruption. So, it is not called immediately. Therefore, it can’t be called while debugging or if it’s inside an infinite loop.

**Conclusion:**

**Arnoldo:**

The scheduler is a software component which manages and orchestrates the tasks of an operative system in appropriate order. It’s the heart of the kernel, in charge of sharing CPU time and resources between every task. Its implementation is not as hard as it seems to be, yet advanced knowledge of the architecture you’re working in is essential if appropriate scheduling is desired. There is much to know about a scheduler and all of its components, and building one appropriately takes effort, as every possible fault must be pinpointed to avoid screwing up the system it’s built for (there are a LOT of possible bad things that could happen).

**Edwin:**

In this first practice while doing the scheduler we were able to give us an idea of ​​how a RTOS works, and how all the apis works too. With this we will facilitate the following practice as it is a continuation of this one.

We had several problems with the ASM code when making the context switch but in the end we were able to solve it. In addition, we complicated ourselves more and did the initialization of each stack in ASM when it is much simpler in C.