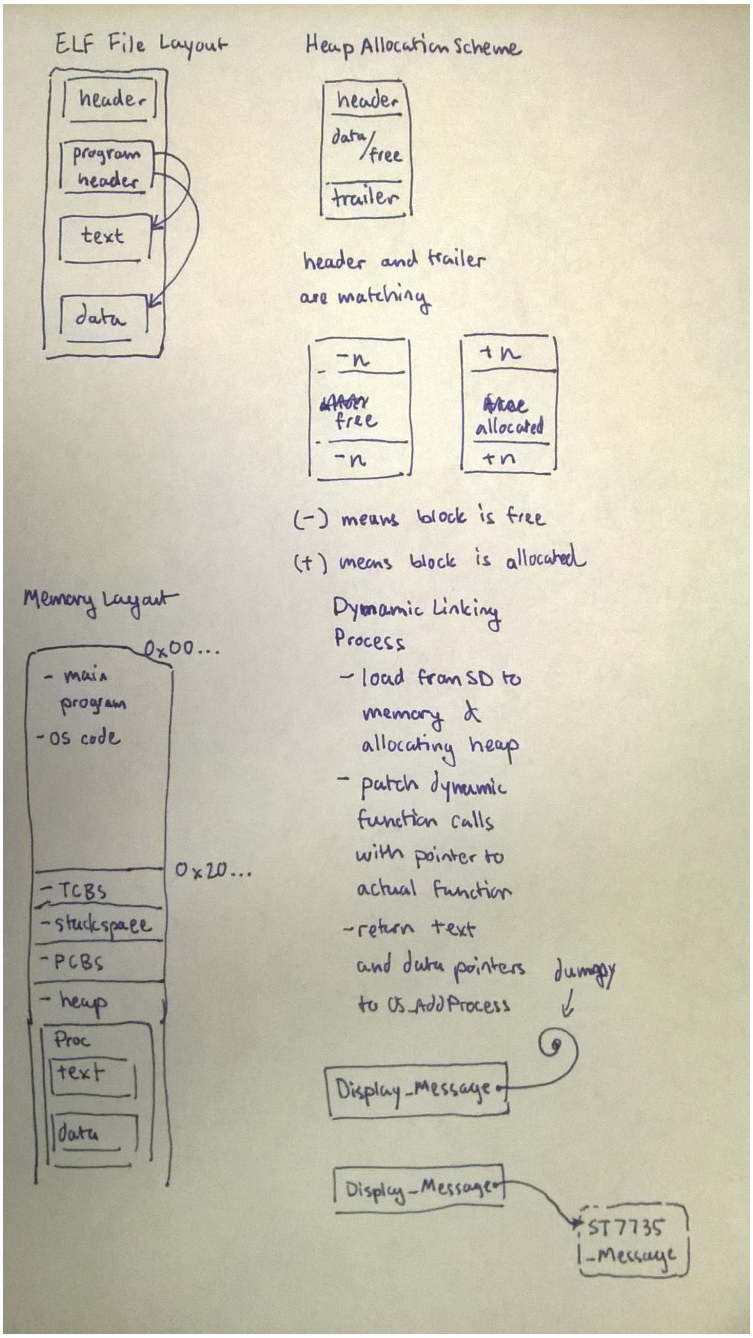
A) Objectives (1/2 page maximum)

The goals of this lab is to dynamically load code from the SD card and execute the compiled software.
To achieve this, we will need to:
0) Compile a process to load onto an SD card (creating the executable)
1) Allocate memory for our external program (memory manager using a heap)
2) Read in data (integrate fat file system)
3) Dynamically link function calls (integrate loader to patch dynamic symbols)
4) Implement static function calls (implement SVC handler)
5) Create structure for managing processes (implement PCB struct)
6) Run interpreter to start processes
The process of this lab is very similar to compiling programs to run on desktops running operating systems such as Windows.

B) Hardware Design (none)

C) Software Design (documentation and code)

1) Pictures illustrating the loader operation, showing: ELF file layout of compiled user program on disk; heap allocation scheme;
memory layout of machine after loading the program; and dynamic linking and relocation process



2) Operating system extensions (C and assembly), including SVC_Handler

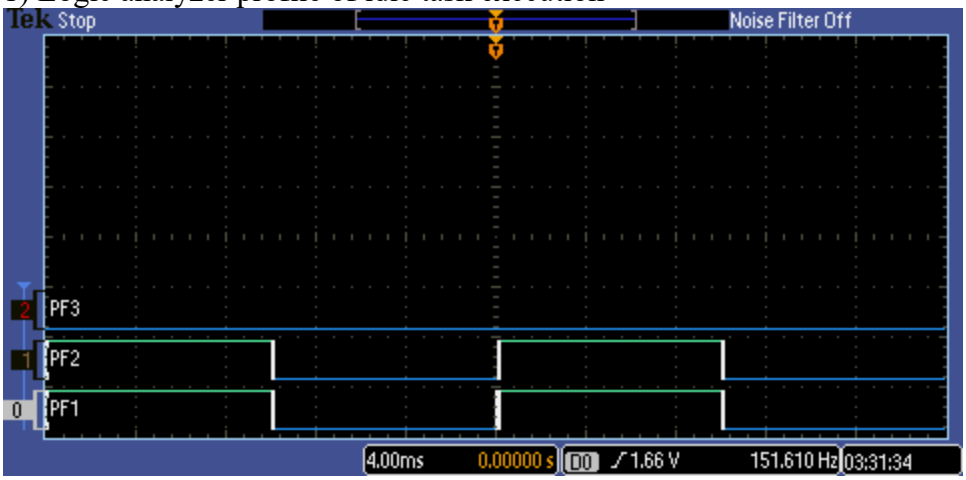
We added a PCB structure which only has ID, text pointer, data pointer, and number threads. When number of threads reach down to 0, we will deallocate whatever data and text points to.
We also fixed OS_AddThread to set the correct data base pointer (R9).
The SVC handler simply gets the call number by looking up the PC and accessing PC-4. Then simply executes the corresponding system call.
Our code can be found in the repo.

3) High level software system (the new interpreter commands)

Our interpreter simply calls exec_elf() with the corresponding file name.

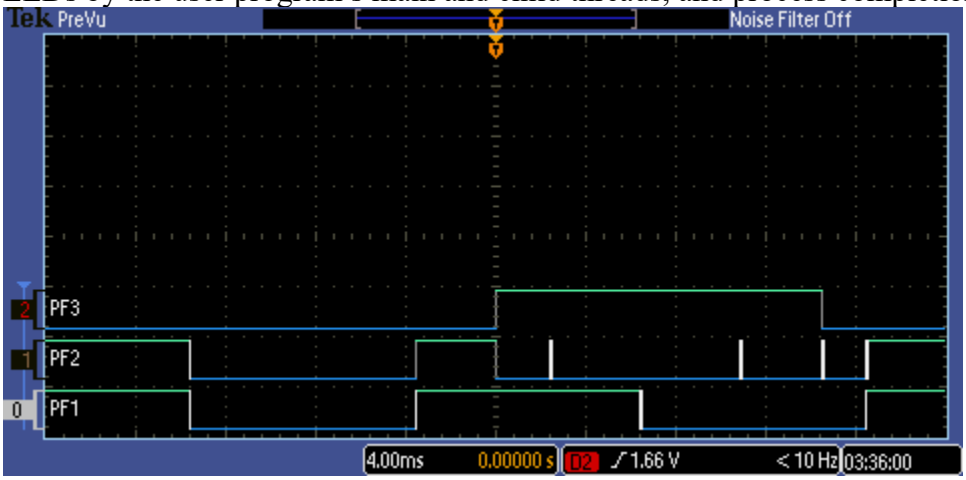
D) Measurement Data

1) Logic analyzer profile of idle task execution



PF1 is SysTick and PF2 is the idle task toggling.

2) Logic analyzer profile of user program execution: process creation, SVC traps, toggling of PF2 and PF3 LEDs by the user program's main and child threads, and process completion.



We can see when the program is loaded, PF2 toggles. Then immediately, the main thread propagates a child thread to toggle PF3. Then both calls OS_Sleep so the idle task toggles PF2 twice. Finally, both process threads toggle PF2 and PF3 at the end.

E) Analysis and Discussion (1 page maximum). In particular, answer these questions

1) Briefly explain the dynamic memory allocation algorithm in your heap manager. Does this implementation have internal or external fragmentation?

We used the given heap implementation. The implementation has both internal and external fragmentation. Internal fragmentation is caused by alignment so not much memory is lost. External fragmentation grows gradually as the free space gets split into multiple small pieces as well as memory required for headers and tailers.

2) How many simultaneously active processes can your system support? What factors limit this number, and how could it be increased?

20; We are limited by the maximum number of threads our system can handle, and by the PCB list's static allocation. We can increase this number by statically allocating more memory for our TCBs and PCBs.