Lab 5 Process Loading and Management

Michael Coulter and Jake Klovenski

A) Objectives (1/2 page maximum)

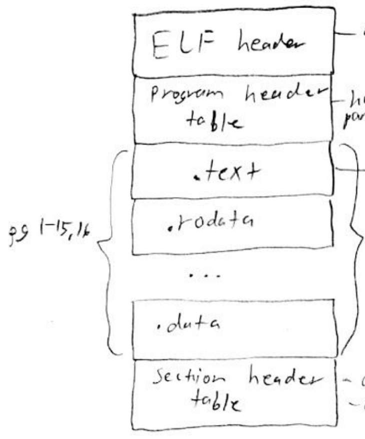
The objective of this lab is to implement user processes that are dynamically loaded into the system. This is accomplished by loading elf files onto the SD card, loading them into a heap on TM4C from the SD card and executing them from the heap. We will need to allow the user program call OS functions. To enable this we will use the SVC to dynamically link threads to OS calls.

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;dynamic linking and relocation process

ELF (Executable and Linkable Format)



- defines address size, file's organization

- how to create a process image (execute a program), determines what parts of file get mapped to memory

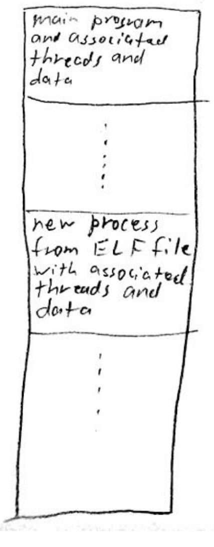
→ .text holds the executable instructions of the program

sections - contain special sections that are used to hold program and control info,
- exec. files are created

- describes files sections, helps locate all file's sections pg 1-9,10
- entry for every section
  - entries have info such as section name, size, etc.

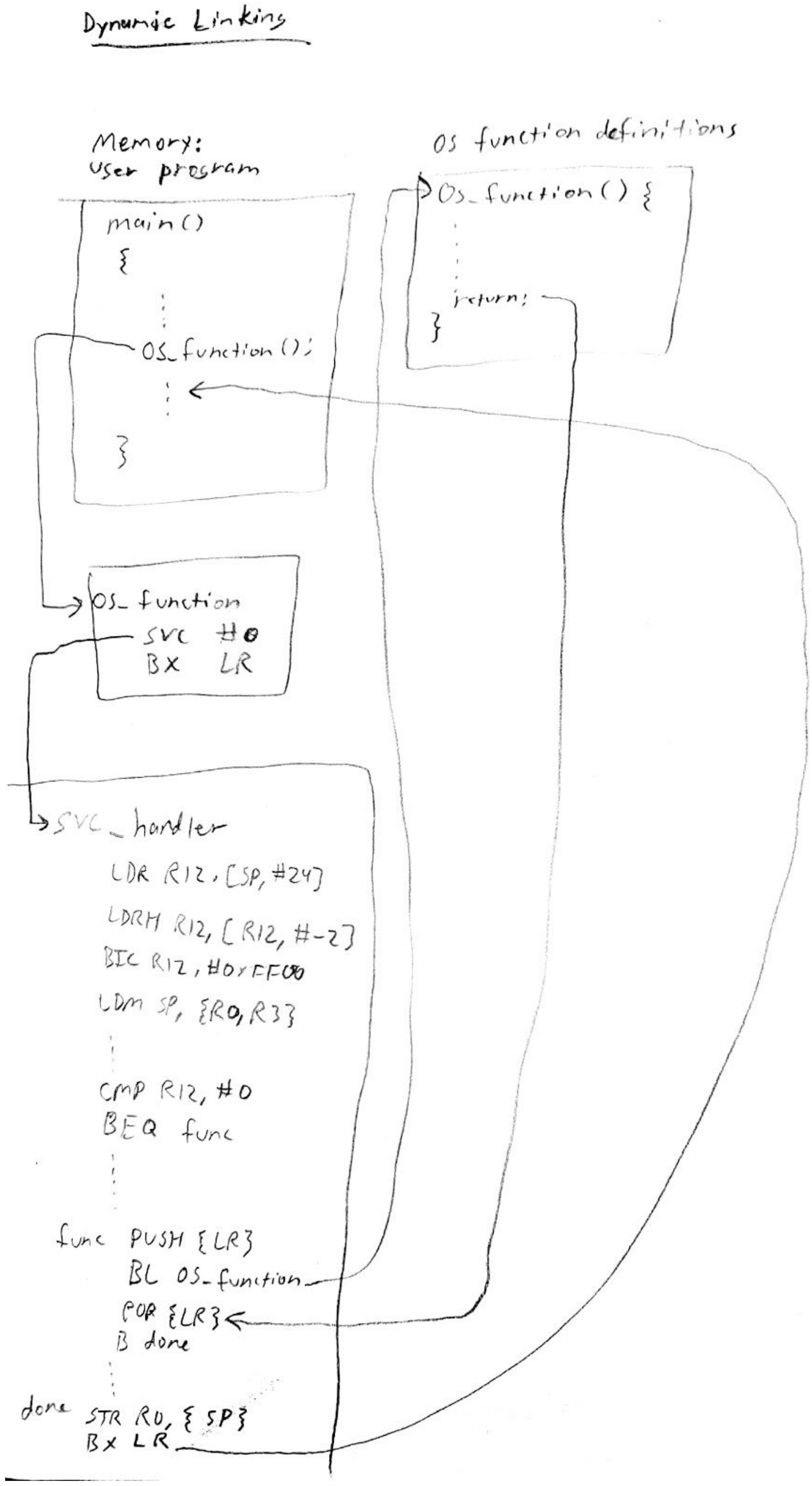
Launch Pad Memory

heap


main program and associated threads and data
new process from ELF file with associated threads and data

- exec_elf function parses ELF file, dynamically allocates memory for the code and data segments on the heap and loads both segments into the allocated space on the heap.
- OS_AddProcess will launch this new program as a process (separate memory space) (a process can have threads that run in a shared memory space)

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



```
main()
{
   :
   OS_function();
   :
}
```

```
OS_function() {
   :
   return;
}
```

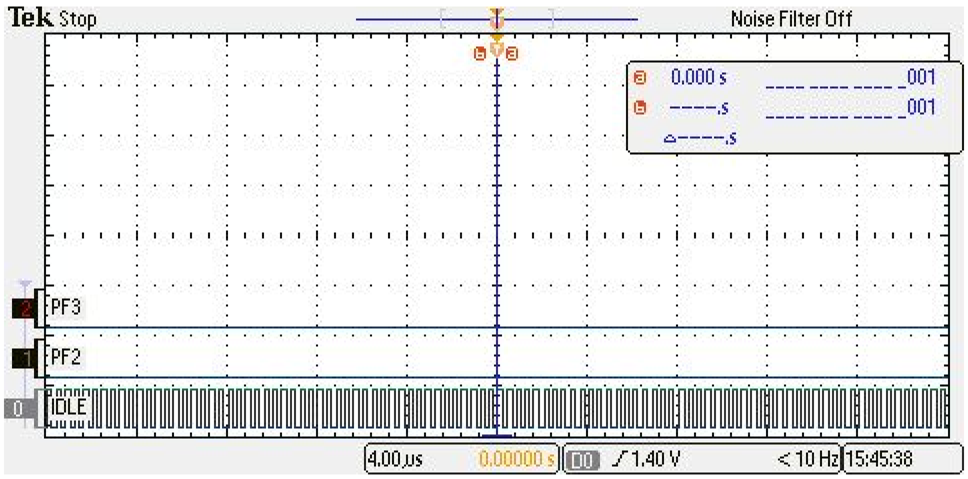
```
OS_function
   SVC  #0
   BX   LR
```

```
SVC_handler
      LDR R12, [SP, #24]
      LDRH R12, [R12, #-2]
      BIC R12, #0xFF00
      LDM SP, {R0, R3}
      :
      CMP R12, #0
      BEQ  func
      :
func  PUSH {LR}
      BL OS_function
      POP {LR}
      B done
      :
done  STR R0, { SP }
      BX LR
```

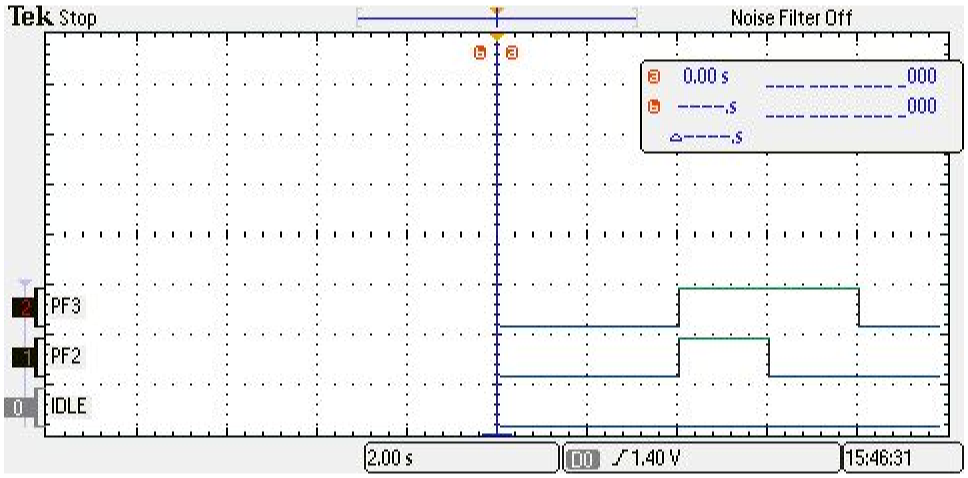
Shows dynamic linking process (part 1) and SVC_Handler.

3) High level software system (the new interpreter commands)
   a) We added a new command to the interpreter that loads and launches a user process from the SD card using the supplied exec_elf function.

D) Measurement Data
1) Logic analyzer profile of idle task execution



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.



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?
   a) The heap is a Knuth Heap implementation. Each section allocated has a section of meta-data at the beginning and end of the allocated section of the heap that indicates the size of the section. This implementation has external fragmentation because each block is exactly the size requested by the program. No internal fragmentation occurs.

2) How many simultaneously active processes can your system support? What factors limit this number, and how could it be increased?
    a) Our program has an artificial limit set by us of 8 processes. A better implementation would have the amount of processes dependant on the size of the program and available memory in the stack. If the amount of processes were dependent on the heap size the max amount of processes would be based on the size of the processes and the available stack size of the new process trying to be added.