# Lab 8: ELF-Introduction with Code

In this lab, we will learn how to manipulate ELF files by writing a simplified version of the readelf program. We will parse the ELF file and extract useful information from it. In particular, we will access the data in the section header table, and in the symbol table. We will also learn to use the mmap system call.

## Important

This lab is written for 32bit machines. Some of the computers in the labs already run on a 64bit OS (use *uname -a* to see if the linux OS is 64bit or not). 32bit and 64bit machines have different instruction sets and different memory layouts. Make sure to include the *-m32* flag when you compile files, and to use the Elf32 data structures (and not the Elf64 ones).

In order to know if an executable file is compiled for 64bit or 32bit platform, you can use readelf, or the *file* command-line tool (for example: file /bin/ls).

## Useful Tips

In some cases it is advised to use hexedit and readelf for debugging purposes. In order to take advantage of these tools and make your tasks easier, you should:

- Print debugging messages: in particular the offsets of the various items, as you discover them from the headers.
- Use *hexedit* and *readelf* to compare the information you are looking for, especially if you run into unknown problems. *hexedit* is great if you know the exact location of the item you are looking for.
- Note that while the object files you will be processing will be linked using *ld*, and will, in most cases, use direct system calls in order to make the ELF file simpler, there is no reason why the programs you write need to use this interface. You are allowed to use the standard library when building your own C programs.
- In order to preserve your sanity, even if the code you MANIPULATE may be without stdlib, we advise that for your OWN CODE you DO use the C standard library!
- In order to keep sane in the following lab as well, **understand** what you are doing and **keep track** of that and of your code, as you will be using them in a future lab.

## Lab 8 Tasks

**Deliverables**
You should read and understand the reading material, and do task 0 before attending the lab. To be eligible for a full grade, you must complete tasks 1 and 2 during the regular lab. Task 3 may be done in a completion lab, if you run out of time.

You must use only the mmap system call to read/write data into your ELF files from this point onwards.

## Task 0

### Task 0a:

Download the following file: a.out. Answer the following questions (be prepared to explain your answers to the lab instructor):

1. Where is the entry point specified, and what is its value?
2. How many sections are there in a.out?
3. What is the size of the `.text` section?
4. Does the symbol `_start` occur in the file? If so, where is it mapped to in virtual memory?
5. Does the symbol main occur in the file? If so, where is it mapped to in virtual memory?
6. Where in the file does the code of function "`main`" start?

### Task 0b:

This task is about learning to use the mmap system call. Read about the mmap system call (man mmap).

Write a program that uses the mmap to examine the header of a 32bit ELF file (include and use the structures in elf.h). The program is first activated as:

```
myELF
```

The program then uses a menu with available operations, as follows:

```
Choose action:
0-Toggle Debug Mode
1-Examine ELF File
2-Print Section Names
3-Print Symbols
5-Quit
```

Note that the menu should use the same technique as in lab 2, i.e. an array of structures of available options. For all other functions implement **stubs**, i.e. a function that does nothing but print a line saying: "not implemented yet".

In this task, we will implement the following options:

**Toggle Debug Mode** turns the debug flag on (if it is currently off, which it is in the initial state) and prints "Debug flag now on". If the debug flag is on, this function prints "Debug flag now off" and turns the flag off. When the debug mode is on, you will print additional information that will help you debug your code.

**Examine ELF File** queries the user for an ELF file name to be used and examined henceforth.

**Quit** unmaps and closes any mapped or open files, and "exit normally".

All file input should be read using the mmap system call. You are NOT ALLOWED to use read, or fread.

> To make your life easier throughout the lab, map the entire file with one mmap call.

In Examine ELF File, after getting the file name, you should close any currently open file (indicated by global variable `Currentfd`) open the file for reading, and then print the following:

1. Bytes 1,2,3 of the magic number (in ASCII)
2. Entry point (in hexadecimal)

Check using *readelf* that your data is correct.

Once you verified your output, extend *examine* to print the following information from the header:

1. Bytes 1,2,3 of the magic number (in ASCII). Henceforth, you should check that the number is consistent with an ELF file, and refuse to continue if it is not.
2. The data encoding scheme of the object file.
3. Entry point (hexadecimal address).
4. The file offset in which the section header table resides.
5. The number of section header entries.
6. The size of each section header entry.
7. The file offset in which the program header table resides.
8. The number of program header entries.
9. The size of each program header entry.

The above information should be printed in the above exact order (Print it as nicely as *readelf* does). If invoked on an ELF file, examine should initialize a global file descriptor variable `Currentfd` for this file, and leave the file open. When invoked on a non-ELF file, or the file cannot be opened or mapped at all, you should print an error message, unmap the file (if already mapped) , close the file (if already open), and set `Currentfd` to -1 to indicate no valid file. You probably also should use a global map_start variable to indicate the memory location of the mapped file.

## Task 1 - Sections

Extend your myELF program from Task 0 to allow printing of all the Section names in an 32bit ELF file (like `readelf -S`). That is, implement the Print Section Names function, as follows.

Print Section Names should visit all section headers in the section header table, and for each one print its index, name, address, offset, size in bytes, and type number. Note that this is done for the file currently mapped, so if current fd is invalid, just print an error message and return.

The format should be:

```
[index] section_name section_address section_offset section_size  section_type
```

```
[index] section_name section_address section_offset section_size  section_type

[index] section_name section_address section_offset section_size  section_type

....
```

Verify your output is correct by comparing it to the output of *readelf*.

In **debug mode** you should also print the value of the important indices and offsets, such as shstrndx and the section name offsets.

You can test your code on the following file: a.out.

**Hints**
Global information about the ELF file is in the ELF header, including location and size of important tables. The size and name of the sections appear in the section header table. Recall that the actual name **strings** are stored in an appropriate **section** (.shstrtab for section names), and not in the section header!

## Task 2 - Symbols

Extend your myELF program from task 1 to support an option that displays information on all the symbol names in a 32bit ELF file.

The Print Symbols option should visit all the symbols in the current ELF file (if none, print an error message and return). For each symbol, print its index number, its name and the name of the section in which it is defined. (similar to `readelf -s`). Format should be:

```
[index] value section_index section_name symbol_name

[index] value section_index section_name symbol_name

[index] value section_index section_name symbol_name

...
```

Verify your output is correct by comparing it to the output of *readelf*.

In **debug mode** you should first print the size of each symbol table, the number of symbols therein. And similar to Task 1, you should print important indices and offsets, such as st_name, and st_shndx

You should finish everything up to here during the lab. The rest can be done in a completion lab, if you run out of time.

# Task 3

## task 3a

The goal of this task is to display the compiled code (in bytes) of the function *main*, in the abc executable.

In order to do that, you need to:

1. find the offset (file location) of the function main.
2. find the size of the function main.
3. use hexedit to find the content of that function.

Finding the needed information:

1. Find the entry for the function main in the symbol table of the ELF executable (`readelf -s`).
2. In that reference, you will find both the size of the function and the function's virtual address and section number.
3. In the section table of the executable, find the entry for the function's section (`readelf -S`).
4. Find both the section's virtual address (Addr) and the section's file offset (Off).
5. Use the above information to find the file offset of the function.

## Task 3b

What are the first two machine instructions in function main, stated in assembly language? I.e. you need to manually dis-assemble these first two instructions.

You can use the opcode information in the nasm manual

## Deliverables:

Tasks 1, and 2 must be completed during the regular lab. Task 3 may be done in a completion lab, but only if you run out of time during the regular lab. The deliverables must be submitted until the end of the lab session.

You must submit **a source file** for tasks 1, 2 along its makefile.