LAB 1

CS 361: Systems Programming / Spring 2023

Description

In this lab session, you will explore how ELF files work for static libraries, including utilities `readelf` and `objdump`. Please read this document carefully and follow the instructions on the last section to complete this lab session. When you answered all the questions, please show your work to the TA.

Object files

You can build object files with the C compiler `gcc` by using the flag `-c`. For instance, the following line creates an object file `lab1.o` from the source file `file1.c`:

```
gcc -c file1.c -o file1.o
```

In this way, you can create many such object files for your program. You can then link all object files together and generate an executable file using the `gcc` command. For instance, consider that you have 3 object files (`file1.o`, `file2.o`, and `file3.o`). You can link all together to generate the executable `lab1` with command:

```
gcc file1.o file2.o file3.o -o lab1
```

Inspecting object files and executables

Once you have an object file, you can use commands `readelf` and `objdump` to inspect their contents. Their detailed documentation can be found on:

- `readelf`: https://man7.org/linux/man-pages/man1/readelf.1.html
- `objdump`: https://linux.die.net/man/1/objdump

readelf

The `readelf` utility reads an object code file & prints out various information about it, including symbol names, their type, their binding etc. You can inspect the output of `readelf` on object files, and observe the differences in the symbol table information for different types of symbols. To do so, use the command

```
readelf -sW <object file>
```

The output of `readelf` has the following columns:

- **Num**: The symbol number
- **Value**: The value (address) of the symbol
- **Size**: The size of the symbol
- **Type**: The type of the symbol
- **Bind**: How symbols are bound (e.g., GLOBAL, WEAK, LOCAL)
- **Vis**, **Ndx**: Misc information
- **Name**: The name of the symbol

If you have access to the source of an object file, you can match symbols by name. If you do not have access to the source, you can understand what each symbol maps to in the source by inspecting the Type and Bind columns.

objdump

`objdump` is yet another utility for displaying object file information. We can use the disassemble option of `objdump` to export the assembly code corresponding to the object file. This allows us to read the assembly file and make informed guesses on the functionality, signature of different functions defined in the source code. To disassemble an object file, use the command:

```
objdump -d <object file>
```

Importing functions and global variables

When building C code, you can declare functions just by their signature, without a body. For instance, the following line declares function `defined_in_another_file`:

```
void defined_in_another_file();
```

Such functions can also take arguments and return values:

```
int defined_in_another_file(char * arg);
```

You can also declare global variables that are exported by other object files using the keyword `extern`. For instance, the following line declares a global variable defined in another file:

```
extern int this_variable_is_defined_elsewhere;
```

Once you declare functions and variables as such, you can use them as normal functions and variables in the rest of your C program.

Guide

1. Accept the invitation for Lab 1 on Github classroom: https://classroom.github.com/a/rDnp9CVu
2. Import the Github repository created to your machine using vscode, as explained in Assignment 0
3. Make sure that you can launch a terminal inside vscode via menus: Terminal > New Terminal
4. The repository has three files: `Makefile`, `lab1.c` and `lib.c`. Modify the provided `Makefile` to compile each into an object file, and then to link both object files into an executable called `lab1` when you type the command `make all` on a terminal. Answer Question 1.
5. Inspect the contents of source file `lib.c` and answer Question 2.
6. Inspect each object file with the `readelf` command and answer Questions 3 and 4.
7. Modify file `lab1.c` so that it calls one function defined inside file `lib.c`, and answer Question 5.
8. Modify file `lab1.c` so that it reads the value of one global variable defined inside file `lib.c` after calling the function, and answer Question 6.
9. Show your work to the TA, commit and push your changes to the repo created in Step 2.

Questions

You can write the answers to your questions to file `ans.txt` in your repository, it then makes it easy to show your work to the TA.

1. What changes did you perform to `Makefile` to build each object file and link all into a single executable when typing `make all`?
2. What global variables and functions are declared inside file `lib.c` ?
3. In the output of `readelf`, what is the Type and Bind of global variables?
4. In the output of `readelf`, what is the Type and Bind of functions?

5. How did you modify file `lab1.c` to execute one function defined inside `lib.c`? What is the output of that function?
6. How did you modify file `lab1.c` to read one global variable defined inside file `lib.c` after calling the function? What is the value of that variable?

Extra / Optional

1. Use command `objdump` on the object file you generated from `lab1.c` and on the executable file. Compare the assembly generated to call functions and read global variables defined in `lab1.c`. Can you explain the differences?

Grading

Show your UIC card to the TA when you enter the lab, or type your UIN on the chat when joining remotely. Stay in the session until you show your work, or until the TA announces that the lab is over.

- You have to remain present for the whole lab to get attendance, which you can then use to resubmit Assignment 1.
- You can leave early after showing your work to the TA (answers to all questions). In this case, you will get a 5% bonus in Assignment 1.