Assignment 1: ELF Files

Jake Gendreau

September 27, 2024

Contents

1 Results

1.1 Steps

1. Write the code.
2. Compile individual files with `gcc <filename.c> -c <filename.o>`. The `-c` flag indicates that it will not be linked yet.
3. Link the files with `gcc <file0.o> ... <fileN.o> -o combined` to link all of the files together and produce the final `ELF` file, called `combined`.
4. Use `readelf -s combined > SymbolTable.txt` to get the symbol table of the `ELF` file, `combined`, and write it to `SymbolTable.txt`.
5. Use `readelf -S combined > DataHeaders.txt` to write the data section headers of the `ELF` file to `DataHeaders.txt`.
6. Use `readelf -h combined > FileHeader.txt` to write the file header of the `ELF` file to `FileHeader.txt`.
7. Use `grep` in each of these files to find relevant fields, such as function names or data headers for the `ELF` file.

1.2 Part 1

With the original code written, and by following the steps above, I can determine the requirements for this assignment.

1.2.1 Size of each function

By using `cat SymbolTable.txt | grep <function name>`, we can find the size for each function in memory, since the second field is the size for the corresponding function in each line.

- `Maximum()` - The command above yields `0000000000011f9 95 FUNC GLOBAL DEFAULT 16 Maximum`, which means that `Maximum()` has a size of 95.
- `Minimum()` - The command above yields `0000000000001258 95 FUNC GLOBAL DEFAULT 16 Minimum`, which means that `Minimum()` has a size of 95.
- `Sum()` - The command above yields `00000000000011b8 65 FUNC GLOBAL DEFAULT 16 Sum`, which means that `Sum()` has a size of 65.
- `main()` - The command above yields `0000000000001149 111 FUNC GLOBAL DEFAULT 16 main`, which means that `main()` has a size of 111.

1.2.2 Location of each function in memory

By using `cat SymbolTable.txt | grep <function name>`, we can find the location for each function in memory, since the first field is the address for the corresponding function in each line.

- `Maximum()` - The command above yields `0000000000011f9 95 FUNC GLOBAL DEFAULT 16 Maximum`, which means that `Maximum()` is stored at `0x0000000000011f9`.
- `Minimum()` - The command above yields `0000000000001258 95 FUNC GLOBAL DEFAULT 16 Minimum`, which means that `Minimum()` is stored at `0x0000000000001258`.
- `Sum()` - The command above yields `00000000000011b8 65 FUNC GLOBAL DEFAULT 16 Sum`, which means that `Sum()` is stored at `0x00000000000011b8`.
- `main()` - The command above yields `0000000000001149 111 FUNC GLOBAL DEFAULT 16 main`, which means that `main()` is stored at `0x0000000000001149`.

1.2.3 Name and location of entry point for the whole program

By using `cat FileHeader.txt | grep Entry`, we can find the location of the entry point.

- `Entry point address` - Running the command above yields `Entry point address: 0x1060`.
- `Entry point name` - By plugging in the value that I found for the address of the entry point, I can find the associated function name. Running `cat SymbolTable.txt | grep 1060`, I get `0000000000001060 38 FUNC GLOBAL DEFAULT 16 _start`, which means that the name of the entry point is `_start`.

1.2.4 Locations and names from stdio.h

The only function that I used from `stdio.h` was `printf()`.

- To find the location of `printf`, I used `cat SymbolTable.txt | grep printf`, however, I found that the address was `0x000...00`. After doing some research, I learned that it's because `stdio.h` and its functions are dynamically linked at runtime.
- To figure out where the dynamic linking was happening, I used `readelf -r combined | grep printf` and got this result: `000000003fd0 000300000007 R_X86_64_JUMP_SLO 0000000000000000 printf@GLIBC_2.2.5 + 0`. The first field is called "Offset", and the parent section, `.rela.plt`, also has an offset at `0x610`. By adding the two, I got that `printf` lives at `0x45E0`, once the program starts to run.

1.2.5 Locations and sizes of data sections

By using `cat DataHeaders.txt | grep data`, I get all of the data locations and sections. According to the internet, the only two that I need to be concerned about are `.data` and `.rodata`. The Data headers start at offset `0x3780`, so I will add that to the final addresses.

- `.data` - using the command above, the line with `.data` yields `[25] .data PROGBITS 0000000000004000 00003000`. Since the `DataHeaders.txt` file uses two lines , by reading lines 25 and 26, I get that `.data` has size `0x024` and has the address of `0x7780`.
- `.rodata` - Using the same method as above, I get that `.rodata` has size `0x01f` and has address `0x5780`.

1.3 Part 2

With the original code written, compiling with the -O flag, and by following the steps above, I can determine the requirements for this assignment.

1.3.1 Size of each function

By using `cat SymbolTable.txt | grep <function name>`, we can find the size for each function in memory, since the second field is the size for the corresponding function in each line.

- `Maximum()` - The command above yields `00000000000011e4 55 FUNC GLOBAL DEFAULT 16 Maximum`, which means that `Maximum()` has a size of 55.
- `Minimum()` - The command above yields `000000000000121b 55 FUNC GLOBAL DEFAULT 16 Minimum`, which means that `Minimum()` has a size of 55.
- `Sum()` - The command above yields `00000000000011c1 35 FUNC GLOBAL DEFAULT 16 Sum`, which means that `Sum()` has a size of 35.
- `main()` - The command above yields `0000000000001149 120 FUNC GLOBAL DEFAULT 16 main`, which means that `main()` has a size of 120.

1.3.2 Location of each function in memory

By using `cat SymbolTable.txt | grep <function name>`, we can find the location for each function in memory, since the first field is the address for the corresponding function in each line.

- `Maximum()` - The command above yields `00000000000011e4 55 FUNC GLOBAL DEFAULT 16 Maximum`, which means that `Maximum()` is stored at `0x00000000000011e4`.
- `Minimum()` - The command above yields `000000000000121b 55 FUNC GLOBAL DEFAULT 16 Minimum`, which means that `Minimum()` is stored at `0x000000000000121b`.

- `Sum()` - The command above yields `00000000000011c1 35 FUNC GLOBAL DEFAULT 16 Sum`, which means that `Sum()` is stored at `0x00000000000011c1`.
- `main()` - The command above yields `0000000000001149 120 FUNC GLOBAL DEFAULT 16 main`, which means that `main()` is stored at `0x0000000000001149`.

1.3.3 Name and location of entry point for the whole program

By using `cat FileHeader.txt | grep Entry`, we can find the location of the entry point.

- `Entry point address` - Running the command above yields `Entry point address: 0x1060`.
- `Entry point name` - By plugging in the value that I found for the address of the entry point, I can find the associated function name. Running `cat SymbolTable.txt | grep 1060`, I get `0000000000001060 38 FUNC GLOBAL DEFAULT 16 _start`, which means that the name of the entry point is `_start`.

1.3.4 Locations and names from stdio.h

The only function that I used from `stdio.h` was `printf()`.

- To find the location of `printf`, I used `cat SymbolTable.txt | grep printf`, however, I found that the address was `0x000...00`. After doing some research, I learned that it's because `stdio.h` and its functions are dynamically linked at runtime.
- To figure out where the dynamic linking was happening, I used `readelf -r combined | grep printf` and got this result: `000000003fd0 000300000007 R_X86_64_JUMP_SLO 0000000000000000 printf@GLIBC_2.2.5 + 0`. The first field is called "Offset", and the parent section, `.rela.plt`, also has an offset at `0x610`. By adding the two, I got that `printf` lives at `0x45e0`, once the program starts to run.

1.3.5 Locations and sizes of data sections

By using `cat DataHeaders.txt | grep data`, I get all of the data locations and sections. According to the internet, the only two that I need to be concerned about are `.data` and `.rodata`. The Data headers start at offset `0x3788`, so I will add that to the final addresses.

- `.data` - using the command above, the line with `.data` yields `[25] .data PROGBITS 0000000000004000 00003000`. Since the `DataHeaders.txt` file uses two lines , by reading lines 25 and 26, I get that `.data` has size `0x024` and has the address of `0x7788`.
- `.rodata` - Using the same method as above, I get that `.rodata` has size `0x01f` and has address `0x5788`.

1.4 Part 3

I have now modified the code to pass the array around rather than having it global, and by following the steps above, I can determine the requirements for this assignment.

1.4.1 Size of each function

By using `cat SymbolTable.txt | grep <function name>`, we can find the size for each function in memory, since the second field is the size for the corresponding function in each line.

- `Maximum()` - The command above yields `0000000000001285 100 FUNC GLOBAL DEFAULT 16 Maximum`, which means that `Maximum()` has a size of 100.
- `Minimum()` - The command above yields `00000000000012e9 100 FUNC GLOBAL DEFAULT 16 Minimum`, which means that `Minimum()` has a size of 100.
- `Sum()` - The command above yields `000000000000123e 71 FUNC GLOBAL DEFAULT 16 Sum`, which means that `Sum()` has a size of 71.
- `main()` - The command above yields `0000000000001169 213 FUNC GLOBAL DEFAULT 16 main`, which means that `main()` has a size of 213.

1.4.2 Location of each function in memory

By using `cat SymbolTable.txt | grep <function name>`, we can find the location for each function in memory, since the first field is the address for the corresponding function in each line.

- `Maximum()` - The command above yields `0000000000001285 100 FUNC GLOBAL DEFAULT 16 Maximum`, which means that `Maximum()` is stored at `0x0000000000001285`.
- `Minimum()` - The command above yields `00000000000012e9 100 FUNC GLOBAL DEFAULT 16 Minimum`, which means that `Minimum()` is stored at `0x00000000000012e9`.
- `Sum()` - The command above yields `000000000000123e 71 FUNC GLOBAL DEFAULT 16 Sum`, which means that `Sum()` is stored at `0x000000000000123e`.
- `main()` - The command above yields `0000000000001169 213 FUNC GLOBAL DEFAULT 16 main`, which means that `main()` is stored at `0x0000000000001169`.

1.4.3 Name and location of entry point for the whole program

By using `cat FileHeader.txt | grep Entry`, we can find the location of the entry point.

- `Entry point address` - Running the command above yields `Entry point address: 0x1080`.
- `Entry point name` - By plugging in the value that I found for the address of the entry point, I can find the associated function name. Running `cat SymbolTable.txt | grep 1080`, I get `0000000000001080 38 FUNC GLOBAL DEFAULT 16 _start`, which means that the name of the entry point is `_start`.

1.4.4 Locations and names from stdio.h

The only function that I used from `stdio.h` was `printf()`.

- To find the location of `printf`, I used `cat SymbolTable.txt | grep printf`, however, I found that the address was `0x000...00`. After doing some research, I learned that it's because `stdio.h` and its functions are dynamically linked at runtime.
- To figure out where the dynamic linking was happening, I used `readelf -r combined | grep printf` and got this result: `000000003fd0 000300000007 R_X86_64_JUMP_SLO 0000000000000000 printf@GLIBC_2.2.5 + 0`. The first field is called "Offset", and the parent section, `.rela.plt`, also has an offset at `0x610`. By adding the two, I got that `printf` lives at `0x45e0`, once the program starts to run.

1.4.5 Locations and sizes of data sections

By using `cat DataHeaders.txt | grep data`, I get all of the data locations and sections. According to the internet, the only two that I need to be concerned about are `.data` and `.rodata`. The Data headers start at offset `0x3788`, so I will add that to the final addresses.

- `.data` - using the command above, the line with `.data` yields `[25] .data PROGBITS 0000000000004000 00003000`. Since the `DataHeaders.txt` file uses two lines , by reading lines 25 and 26, I get that `.data` has size `0x010` and has the address of `0x7788`.
- `.rodata` - Using the same method as above, I get that `.rodata` has size `0x01f` and has address `0x5788`.

2 Assignment Log

2.1 Hours

Prior to starting this assignment, I expect it will take my about 4 hours to complete.

- Monday, September 23 7:00 PM - 8:00 PM: Figured out readelf commands and wrote base code
- Thursday, September 26 3:00 PM - 5:00 PM and 9:30 PM - 11:30 PM: Did the program writeup.

It took a little longer than I anticipated, but that's alright. I also took a break in the middle to go to the jam session at Bucer's, so that was a nice break.

2.2 Notes

- When using the -O flag, the sizes for the functions were considerably smaller, almost half in some cases, when compared to no optimization. That was only with level 1 optimizations as well.
- The program data in `.data` decreased when I passed the array to the functions rather than having it be global. I'm not entirely sure why this is, but it was interesting. The functions were also slightly larger compared to their counterparts, particularly main though.
- The entry point for the program also changed when I changed the functions to pass the array, rather than having it global. Again, not sure why.
- I'm not sure if I did the `stdio.h` part right, but everything online said that it was dynamically linked, so I did the best that I could.

2.3 Challenges

- I thought that the -O flag for optimization was the same as the -o flag for output, and got confused by that. Then I got confused because I thought it was a -0 for a minute, then I thought that it would go in the same place as the -o flag. Eventually I figured out that it was a capital O and that it went after the input files.
- Understanding `readelf` and all of its flags took a while, and I still don't feel incredibly solid on it, but I got what I needed to down.
- Figuring out the compilation process using -o files as intermediaries was confusing, but again, the internet helped me out there.
- I thought that `extern` functions would be significantly harder than they were.

3 Source Files

My formatting didn't want to put it on this page and I didn't feel like debugging it at the time because it's getting late...
Source files on following pages

Listing 1: MainPBR.c

```c
/*
Jake Gendreau

MainPBR.c
Main file implemented with pass by
reference rather than global variable.

9/26/2024
*/

#include <stdio.h>

extern int Sum(int* data, int size);
extern int Maximum(int* data, int size);
extern int Minimum(int* data, int size);

int main()
{
    int data[] = {10, 20, 30, 40, 40};
    int size = 5;

    printf("Sum: %i\n", Sum(data, size));
    printf("Max: %i\n", Maximum(data, size));
    printf("Min: %i\n", Minimum(data, size));
}
```

Listing 2: FindMinPBR.c

```c
/*
Jake Gendreau

FindMinPBR.c
FindMin of array passed by reference

9/26/2024
*/

#include <stdio.h>

int Minimum(int* data, int size)
{
    int min = data[0];

    for(int i = 1; i < 5; i++)
    {
        if(min > data[i])
        {
            min = data[i];
        }
    }
    return min;
}
```

Listing 3: FindMaxPBR.c

```c
/*
Jake Gendreau

FindMaxPBR.c
FindMax of array passed by reference

9/26/2024
*/

#include <stdio.h>

int Maximum(int* data, int size)
{
    int max = data[0];

    for(int i = 1; i < 5; i++)
    {
        if(max < data[i])
        {
            max = data[i];
        }
    }

    return max;
}
```

Listing 4: SumPBR.c

```c
/*
Jake Gendreau

SumPBR.c
Sum an array passed by reference

9/26/2024
*/

#include <stdio.h>

int Sum(int* data, int size)
{
    int sum = 0;
    for(int i = 0; i < 5; i++)
    {
        sum += data[i];
    }

    return sum;
}
```

Listing 5: Main.c

```
/*
Jake Gendreau

Main.c
Main file to test readelf

9/23/2024
*/

#include <stdio.h>

extern int SumAndAverage();
extern int Maximum();
extern int Minimum();

int data[] = {10, 20, 30, 40, 40};

int main()
{
    Sum();
    Maximum();
    Minimum();
}
```

Listing 6: FindMin.c

```c
/*
Jake Gendreau

FindMin.c
FindMin of array

9/23/2024
*/

#include <stdio.h>

extern int data[];

int Minimum()
{
    int min = data[0];

    for(int i = 1; i < 5; i++)
    {
        if(min > data[i])
        {
            min = data[i];
        }
    }

    printf("Min: %i\n", min);

    return min;
}
```

Listing 7: FindMax.c

```
/*
Jake Gendreau

FindMax.c
FindMax of array

9/23/2024
*/

#include <stdio.h>

extern int data[];

int Maximum()
{
    int max = data[0];

    for(int i = 1; i < 5; i++)
    {
        if(max < data[i])
        {
            max = data[i];
        }
    }

    printf("Max: %i\n", max);

    return max;
}
```

Listing 8: Sum.c

```c
/*
Jake Gendreau

Sum.c
Sum an array

9/23/2024
*/

#include <stdio.h>

int Sum(int* data, int size)
{
    int sum = 0;
    for(int i = 0; i < 5; i++)
    {
        sum += data[i];
    }

    return sum;
}
```