Panayioti Kitsos

ECE 437 Operations Systems PA02

1)

a) Find type of simple_math.o, ELF's magic number, and the section headers.

Type of file = REL (Relocatable file)
ELF's Magic Number = 7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
Number of Section Headers = 13

FIGURE 1: simple_math.o ELF Header

```
[pkitsos@vesta hw2]$ readelf -a simple_math.o
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              REL (Relocatable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x0
  Start of program headers:          0 (bytes into file)
  Start of section headers:          1288 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           0 (bytes)
  Number of program headers:         0
  Size of section headers:           64 (bytes)
  Number of section headers:         13
  Section header string table index: 10
```

b) List sections matched to the Linux process memory map shown in the class, plus one more section by your choice with simple explanation

FIGURE 2: Table Matching Linux Process Memory Map

Section Number	Section Name	Linux Memory Map
1	.text	.text
2	.rela.text	.text
3	.data	.data
4	.bss	.bss
6	.comment	.text
7	.note.GNU-stack	.stack

.text: This maps to the text segment, where program executables are stored.

.data: This maps to the data segment, where initialized data is stored.

.bss: This maps to the bss segment, where uninitialized data is stored.

.note.GNU-stack: This maps to the stack in memory.

.rodata: This section holds read only data that contributes to non-writable segments in the process image.

.symtab: This section holds the symbol table. If the file has a loadable segment including the symbol table, the sections attributes include SHF_ALLOC.

FIGURE 3: simple_math.o Section Header Table

```
Section Headers:
  [Nr] Name              Type             Address           Offset
       Size              EntSize          Flags  Link  Info  Align
  [ 0]                   NULL             0000000000000000  00000000
       0000000000000000  0000000000000000           0     0     0
  [ 1] .text             PROGBITS         0000000000000000  00000040
       0000000000000080  0000000000000000  AX       0     0     1
  [ 2] .rela.text        RELA             0000000000000000  00000388
       0000000000000168  0000000000000018   I      11     1     8
  [ 3] .data             PROGBITS         0000000000000000  000000c0
       0000000000000004  0000000000000000  WA       0     0     4
  [ 4] .bss              NOBITS           0000000000000000  000000c4
       0000000000000000  0000000000000000  WA       0     0     1
  [ 5] .rodata           PROGBITS         0000000000000000  000000c8
       0000000000000052  0000000000000000   A       0     0     8
  [ 6] .comment          PROGBITS         0000000000000000  0000011a
       000000000000002e  0000000000000001  MS       0     0     1
  [ 7] .note.GNU-stack   PROGBITS         0000000000000000  00000148
       0000000000000000  0000000000000000           0     0     1
  [ 8] .eh_frame         PROGBITS         0000000000000000  00000148
       0000000000000038  0000000000000000   A       0     0     8
  [ 9] .rela.eh_frame    RELA             0000000000000000  000004f0
       0000000000000018  0000000000000018   I      11     8     8
  [10] .shstrtab         STRTAB           0000000000000000  00000180
       0000000000000061  0000000000000000           0     0     1
  [11] .symtab           SYMTAB           0000000000000000  000001e8
       0000000000000168  0000000000000018          12     9     8
  [12] .strtab           STRTAB           0000000000000000  00000350
       0000000000000031  0000000000000000           0     0     1
```

c) Reading the symbol table, for integer variables, aa, bb, cc, and procedures (either defined or invoked in simple_math.c) which one is included in the symbol table, which one is not? Why or why not?

Aa: [Variable in table] The variable aa was defined a global variable and assigned an index value of 3.
Bb: [Variable in table] The variable bb was defined a common variable.
Cc: [Variable not in table] The variable cc is not in the symbol table because it is declared in the main function therefore it can only be used in the main function.
Main: [Function in table] Main function was defined and assigned an index value of 1. It is in the table because the linker needs to find it to resolve references.
Printf: [Function in table] Printf was undefined because it uses undefined functions. It is in the table because the linker needs to find it to resolve references.
Int_add: [Function in table] The int_add function is in the table but in undefined because the linker needs to find them to resolve references.
Int_mult: [Function in table] The int_mul function is in the table but it's undefined because the linker needs to find them to resolve references.

If a symbols references something out of the file, or the symbols scopes are outside of the file, then the symbol must be on the symbol table because the linker needs to know that it needs to find it otherwise it won't. This is precisely why cc isn't in the symbol table because its scope lies only within the main function.

FIGURE 4: simple_math.o Symbol Table

```
Symbol table '.symtab' contains 15 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
     0: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 0000000000000000     0 FILE    LOCAL  DEFAULT  ABS simple_math.c
     2: 0000000000000000     0 SECTION LOCAL  DEFAULT    1
     3: 0000000000000000     0 SECTION LOCAL  DEFAULT    3
     4: 0000000000000000     0 SECTION LOCAL  DEFAULT    4
     5: 0000000000000000     0 SECTION LOCAL  DEFAULT    5
     6: 0000000000000000     0 SECTION LOCAL  DEFAULT    7
     7: 0000000000000000     0 SECTION LOCAL  DEFAULT    8
     8: 0000000000000000     0 SECTION LOCAL  DEFAULT    6
     9: 0000000000000000     4 OBJECT  GLOBAL DEFAULT    3 aa
    10: 0000000000000004     4 OBJECT  GLOBAL DEFAULT  COM bb
    11: 0000000000000000   128 FUNC    GLOBAL DEFAULT    1 main
    12: 0000000000000000     0 NOTYPE  GLOBAL DEFAULT  UND int_add
    13: 0000000000000000     0 NOTYPE  GLOBAL DEFAULT  UND printf
    14: 0000000000000000     0 NOTYPE  GLOBAL DEFAULT  UND int_mul

No version information found in this file.
[pkitsos@vesta hw2]$
```

2)

a) **Link simple_math.o, my_add.o, my_mul.o to generate an executable file simplemath. Examine file simplemath to study variables and routines (namely, aa, bb, int_add, int_mul, main) by listing their names, their types, and their sections.**

FIGURE 5: Table Examining simplemath Executable

Name	**Type**	**Section**
aa	OBJECT	.data (D)
bb	OBJECT	.bss (B)
main	FUNCTION	.text (T)
int_add	FUNCTION	.text (T)
int_mul	FUNCTION	.text (T)

FIGURE 6: nm simplemath Table

```
[pkitsos@comet hw2]$ ls
Makefile   my_add.c   my_add.o  my_mul.c~  simplemath     simple_math.c~
Makefile~  my_add.c~  my_mul.c  my_mul.o   simple_math.c  simple_math.o
[pkitsos@comet hw2]$ nm simplemath
0000000000601034 D aa
000000000060103c B bb
0000000000601038 B __bss_start
0000000000601038 b completed.6345
0000000000601030 D __data_start
0000000000601030 W data_start
0000000000400470 t deregister_tm_clones
00000000004004e0 t __do_global_dtors_aux
0000000000600e18 t __do_global_dtors_aux_fini_array_entry
0000000000400698 R __dso_handle
0000000000600e28 d _DYNAMIC
0000000000601038 D _edata
0000000000601040 B _end
0000000000400684 T _fini
0000000000400500 t frame_dummy
0000000000600e10 t __frame_dummy_init_array_entry
00000000004008b0 r __FRAME_END__
0000000000601000 d _GLOBAL_OFFSET_TABLE_
                 w __gmon_start__
00000000004003e0 T _init
0000000000600e18 t __init_array_end
0000000000600e10 t __init_array_start
00000000004005ad T int_add
00000000004005d4 T int_mul
0000000000400690 R _IO_stdin_used
                 w _ITM_deregisterTMCloneTable
                 w _ITM_registerTMCloneTable
0000000000600e20 d __JCR_END__
0000000000600e20 d __JCR_LIST__
                 w _Jv_RegisterClasses
0000000000400680 T __libc_csu_fini
0000000000400610 T __libc_csu_init
                 U __libc_start_main@@GLIBC_2.2.5
000000000040052d T main
                 U printf@@GLIBC_2.2.5
00000000004004a0 t register_tm_clones
0000000000400440 T _start
0000000000601038 D __TMC_END__
[pkitsos@comet hw2]$
```

b) Compare simplemath and simple_math.o to discuss differences between executable and linkable object files, in terms of program headers, section headers, and symbol tables.

Program headers: The linkable object file simple_math.o does not have any program headers whereas, the simplemath executable file does. According to the ELF Header table (below), the executable simplemath has 56 bytes of program headers spanning 9 program headers.
Section headers: Both the simple_math.o object file and the simplemath executable have section headers. I did notice however, that the executable file contains quite a few more section headers. For instance, the simplemath executable shows 36 section headers versus 23 found in the simple_math.o object file. The obvious section headers that are contained in the executable and not the object file are: .init, .got, and .dynamic. Since headers are for dynamic linking information, global offset, and code to initialize code before the main function, it appears that the executable file utilizes section headers that set up values and initialize memory before the code starts executing. On the other hand, the object file has headers that only setup memory offset of where those things will happen.
Symbol headers: The simplemath executable contains a dynamic symbol table, but not the object file. This dynamic symbol table appears to link function that are called within the stdio.h header file. The regular symbol table in the simplemath executable also contains twice as many entries than the simple_math object file. The executable contains 77 regular system headers whereas the object file contains a mere 29. Looking at both tables, we can see that the system headers don't start until entry 40 in the executables table. The executable file contains system entries to setup IO, program frames, header files, external functions, and the global offset variable and the symbol table for the object file doesn't start until three quarters the way through. The symbol table only contains entries for things declared specifically inside the object files in which they were linked from.

3) **Build and link your own static math library**

To build my own static math library, I created a makefile to make things easier and more streamlined. To begin, I run make all, which compiles my_add.c, my_mul.c, and simple_math.c using the flag –c to create object files with the respective names. From here, I run make lib_a. This essentially builds and populates the static library libmymath.a with the object files my_add.o and my_mul.o. This was done using the ar command with –rcs where the r is for replacing new files, c for creating the archive, and s for adding the index to the archive. To show that the library has been populated, I used the ar command with the –t flag to get a table listing the object files in the library. Next, I compiled simple_math.c and the static library libmath.a into the executable file simpleone_a using the –o flag then finish by linking the library path with the respective library flag –lmymath. After all this is complete, I have a static math library built and linked to the executable file simpleone_a that runs error free. See figure 8 in number 4 below for the makefile used.

<u>FIGURE 7: Error Free simpleone_a Executable</u>

```
[pkitsos@triton hw2]$ ls
Makefile   my_add.c   my_mul.c   simple_math.c
Makefile~  my_add.c~  my_mul.c~  simple_math.c~
[pkitsos@triton hw2]$ make all
gcc -std=gnu99 -g -O -c simple_math.c -fPIC -I -L
simple_math.c:4:1: warning: return type defaults to 'int' [enabled by default]
 main() {
 ^
gcc -std=gnu99 -g -O -c my_add.c -fPIC -I -L
gcc -std=gnu99 -g -O -c my_mul.c -fPIC -I -L
[pkitsos@triton hw2]$ ls
Makefile   my_add.c   my_add.o  my_mul.c~  simple_math.c   simple_math.o
Makefile~  my_add.c~  my_mul.c  my_mul.o   simple_math.c~
[pkitsos@triton hw2]$ make lib_a
ar -rcs libmymath.a my*.o
[pkitsos@triton hw2]$ ar -t libmymath.a
my_add.o
my_mul.o
[pkitsos@triton hw2]$ make simpleone_a
gcc -o simpleone_a simple_math.c libmymath.a
gcc -o simpleone_a simple_math.c -L/nfs/user/p/pkitsos/Fall2017/ece437/hw2 -lmym
ath
[pkitsos@triton hw2]$ ./simpleone_a

L: int_add(3, 3)), aa = 4195840, bb = 1303826048
Hello World! int_add(2,3)=5

L: int_mul(2, 3)Hello World again! int_mul(2,3)=6
[pkitsos@triton hw2]$
```

4) **Build and link your own shared math library**
To build and link my shared math library, I used a Makefile to make the process more streamlined after incrementally relinking. To start, I run make all which compiles my_add.c, my_mul.c, and simple_math.c into object files into Position Independent Code (PIC) using the flags –c –fPIC –I –L where the –c flag creates object file, -fPIC –I -L options to create PIC and add directory to the list of searched directories. Then, I run make lib_so to build a dynamic shared math library using the –shared flag to indicate I am building a shared library using the object files and the –o flag. Last, I run make simpleone_so to link and generate the executable file simpleone_so using –o to create the executable with the shared library that I created libmymath.so. To link I then move the shared library into /usr/lib/ and run ldconfig to map the shared library name to the location of the corresponding shared library file. After all this is done, I now have the simpleone_so executable that compiles and runs error free. Note: The UNM Linux server

doesn't allow me to have super user permissions so I had to resort to a Linux virtual machine to actual get it working.

Dependencies:
When checking the dependencies, I found that the three common sections had varying addresses. Apart from that, the other big difference was the executable simpleone_so contained the shared library dependency libmymath.so unlike the simpleone_a executable. *libmymath.so => /usr/lib/libmymath.so (0x00007f4f09213000)*

FIGURE 8: Dependencies

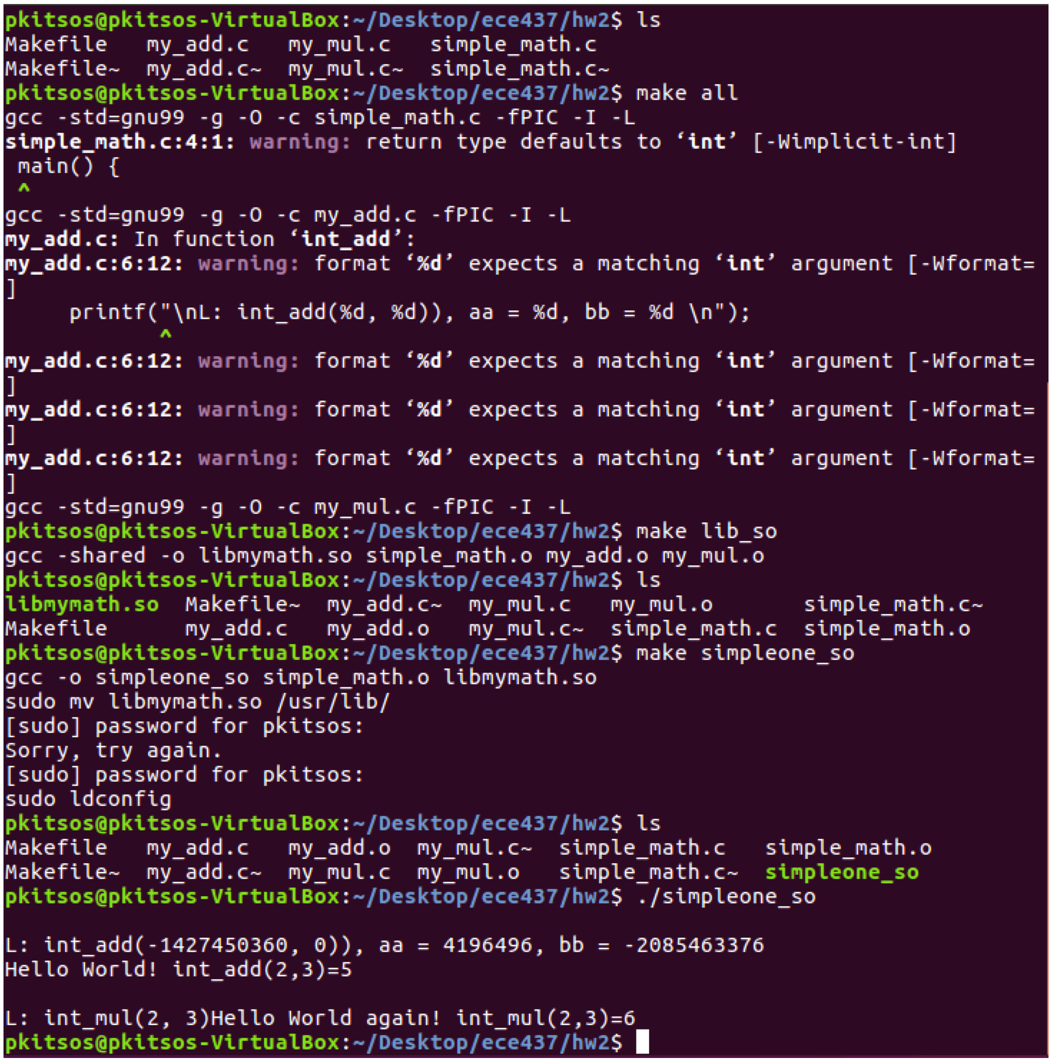
linux-vdso.so.1	**libc.so.6**	**/lib64/ld-linux-x86-64.s0.2**
0x00007ffebe1d8000	0x00007fcdc29e4000	0x00005643b9d36000
0x00007ffe56f57000	0x00007f4f08e49000	0x000055f72ada8000

FIGURE 9: Makefile

```
################################################################################
#
# ECE 437 Operating System PA02 Makefile
# Professor: Dr. Shu
# Created by: Panayioti Kitsos
# Date: September 13, 2017
#
# Problem 2:
#       $ make all
#       $ make simplemath
#
# Problem 3:
#       $ make all
#       $ make lib_a
#       $ make simpleone_a
#       $ ./simpleone_a
#
# Problem 3:
#       $ make all
#       $ make lib_so
#       $ make simpleone_so
#       $ ./simpleone_so
#
################################################################################

CC = gcc
CFLAGS = -std=gnu99 -g -O -c
CSECFL = -fPIC -I -L
CFLAG3 = -shared
OBJ = simple_math.o my_add.o my_mul.o
LD = ld -r
RM = /bin/rm -f
A = libmymath.a
SO = libmymath.so
LIBS = -lmymath
```

```
################################  Make All  ###################################

# Compiles all files to PIC
all: simple_math my_add  my_mul

# The following build object files
simple_math: simple_math.c
        $(CC) $(CFLAGS) $@.c $(CSECFL)

my_add: my_add.c
        $(CC) $(CFLAGS) $@.c $(CSECFL)

my_mul: my_mul.c
        $(CC) $(CFLAGS) $@.c $(CSECFL)

##########################  Simplemath Executable  #############################

# Compiles simplemath executable file
simplemath: $(OBJ)
        $(CC) $(OBJ) -o $@

###########################  Static Math Library  ##############################

# Builds and populates static library (libmymath.a) with object files
lib_a: $(OBJ)
        ar -rcs $(A) my*.o

# Compile/link static library and create simpleone_a executable
simpleone_a: $(OBJ)
        $(CC) -o $@ simple_math.c $(A)
        $(CC) -o $@ simple_math.c -L/nfs/user/p/pkitsos/Fall2017/ece437/hw2 $(LI
BS)

#########################  Shared Math Library  ################################

# Create dynamic shared math library (libmymath.so)
lib_so: $(OBJ)
        $(CC) $(CFLAG3) -o $(SO) $(OBJ)

# Link and generate executable simpleone_so
simpleone_so: $(OBJ)
        $(CC) -o $@ simple_math.o $(SO)
        sudo mv $(SO) /usr/lib/
        sudo ldconfig

#############################  Clean  ##########################################

clean:
        $(RM) *.o simplem* simpleo* lib* $(SO)
```

FIGURE 9: Error Free simpleone_so Executable



```
pkitsos@pkitsos-VirtualBox:~/Desktop/ece437/hw2$ ls
Makefile   my_add.c   my_mul.c   simple_math.c
Makefile~  my_add.c~  my_mul.c~  simple_math.c~
pkitsos@pkitsos-VirtualBox:~/Desktop/ece437/hw2$ make all
gcc -std=gnu99 -g -O -c simple_math.c -fPIC -I -L
simple_math.c:4:1: warning: return type defaults to ‘int’ [-Wimplicit-int]
 main() {
 ^
gcc -std=gnu99 -g -O -c my_add.c -fPIC -I -L
my_add.c: In function ‘int_add’:
my_add.c:6:12: warning: format ‘%d’ expects a matching ‘int’ argument [-Wformat=
]
     printf("\nL: int_add(%d, %d)), aa = %d, bb = %d \n");
            ^
my_add.c:6:12: warning: format ‘%d’ expects a matching ‘int’ argument [-Wformat=
]
my_add.c:6:12: warning: format ‘%d’ expects a matching ‘int’ argument [-Wformat=
]
my_add.c:6:12: warning: format ‘%d’ expects a matching ‘int’ argument [-Wformat=
]
gcc -std=gnu99 -g -O -c my_mul.c -fPIC -I -L
pkitsos@pkitsos-VirtualBox:~/Desktop/ece437/hw2$ make lib_so
gcc -shared -o libmymath.so simple_math.o my_add.o my_mul.o
pkitsos@pkitsos-VirtualBox:~/Desktop/ece437/hw2$ ls
libmymath.so  Makefile~  my_add.c~  my_mul.c   my_mul.o       simple_math.c~
Makefile      my_add.c   my_add.o   my_mul.c~  simple_math.c  simple_math.o
pkitsos@pkitsos-VirtualBox:~/Desktop/ece437/hw2$ make simpleone_so
gcc -o simpleone_so simple_math.o libmymath.so
sudo mv libmymath.so /usr/lib/
[sudo] password for pkitsos:
Sorry, try again.
[sudo] password for pkitsos:
sudo ldconfig
pkitsos@pkitsos-VirtualBox:~/Desktop/ece437/hw2$ ls
Makefile   my_add.c   my_add.o  my_mul.c~  simple_math.c   simple_math.o
Makefile~  my_add.c~  my_mul.c  my_mul.o   simple_math.c~  simpleone_so
pkitsos@pkitsos-VirtualBox:~/Desktop/ece437/hw2$ ./simpleone_so

L: int_add(-1427450360, 0)), aa = 4196496, bb = -2085463376
Hello World! int_add(2,3)=5

L: int_mul(2, 3)Hello World again! int_mul(2,3)=6
pkitsos@pkitsos-VirtualBox:~/Desktop/ece437/hw2$
```