ECE437/CS481 Operating Systems, Fall 2017

Programming Assignment #02 (3%)

Due: Thursday, 09/14/2017

1. **(30%) Use Linux utilities to explore the format of a linkable object file.**

Create a program in C (named as simple_math.c) as follows

```
#include<stdio.h> /* simple_math.c */
int  aa=2, bb;
int int_add(int, int), int_mul(int,int);
main() {
   int cc;
   bb = 3; cc = int_add(aa,bb);
   printf("Hello World! int_add(%d,%d)=%d\n", aa, bb, cc);
   printf("Hello World again! int_mul(%d,%d)=%d\n", aa, bb, int_mul(aa, bb));
}
```

a) Compile your simple_math.c into simple_math.o with "-c" option. Find the type of file simple_*math.o*, ELF's magic number and the number of section headers. ("gcc -c" and "readelf -a" may help).

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

c) Reading the symbol table ("nm" may help), 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?

2. **(20%) Explore the format of an executable object file.**

Complete two more programs in C (named as my_add.c and my_mul.c, respectively)

```
#include<stdio.h> /* my_add.c */
extern int aa,bb;
int int_add(int x, int y) {
   printf("\nL: int_add(%d,%d)), aa=%d, bb=%d\n",x,y,aa,bb);
   return(x+y);
  }
```

```
#include<stdio.h> /*my_mul.c*/
  int int_mul(int x, int y) {
     printf("\nL: int_mul(%d,%d))", x,y);
     return(x*y);
    }
```

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.

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.

3. **(25%) Build and link your own static math library.**

Instead of linking *math.o*, *my_add.o*, *my_mul.o* into *simplemath* directly, you are asked to build your own static math lib, *libmymath.a*, and then to link it with *math.o* to generate your executable *simplemath_a*. Show your steps with comments and discussion. You don't need to submit other files. Make sure your every step is error free (compile, link, and execute your final *simplemath_a*).

a) Compile your *my_add.c* and *my_mul.c* respectively. (**You may want to eliminate external definition of aa and bb in** ***my_add.c*****.)**

b) Build your own static math library. ("ar -rcs" may help).

c) Link your *math.o* with your newly built *libmymath.a* to generate an executable file, *simpleone_a*, to execute.

4. **(25%) Build and link your own shared math library.**

You are asked to build your own shared math lib, *libmymath.so*, and then to link it with *math.o* to generate your executable *simpleone_so*. (gcc 's option -fPIC, -shared, -l, -L may help). Show your steps with comments and discussion. You don't need to submit other files. Make sure your every step is error free (compile, link, and execute your final *simpleone_so*).

a) Compile your *my_add.c* into a Position Independent Code (PIC). Start your *libmymath.so* with *my_add* function only. (**You may want to eliminate external definition of aa and bb.)**

b) Compile your *my_mul.c* into a Position Independent Code (PIC) too. Incrementally build (relink) your *libmymath.so* with *my_mul* function being added.

c) Link your *math.o* with your newly built *libmymath.so* to generate an executable file, *simpleone_so*, to execute.

d) Check the shared library dependencies of the executable *simpleone_a* and *simpleone_so* respectively. What is the difference between them? ("ldd *executable*" may help.)

The man pages for *gcc, file, nm, readelf, ar, objdump, ldd* can be good resources to learn. You are encouraged to google many other resources as well. For Q3 and Q4, you may want to refer to:

- "Static , Shared Dynamic and Loadable Linux libraries" at http://www.yolinux.com/TUTORIALS/LibraryArchives-StaticAndDynamic.html and
- "Shared libraries with GCC on Linux" at http://www.cprogramming.com/tutorial/shared-libraries-linux-gcc.html where you can find step-by-step examples.

The following link may also help you to understand more of the static and shared libraries:

- http://www.linuxfromscratch.org/blfs/view/svn/introduction/libraries.html