CS 361: Computer Systems **Lab Session 3 Worksheet**
Fall 2019

Dynamic Libraries: Setting up your environment

1. Before we begin, log in to systems1.cs.uic.edu (if systems1 is down, use systems2, systems3, etc.), and check out the github repository which contains the practice code for this lab session (https://github.com/uicsystems/LabSession3-PracticeCode.git).
2. Included in the lab session files is the compiled binary executable "uselib". Try running it with "./uselib". You should get an error that says

   ```
   ./uselib: error while loading shared libraries: libsess3.so: cannot open shared
   object file: No such file or directory
   ```
3. This error is telling you that your runtime environment cannot find the shared library (libsess3.so) that the uselib executable depends on. In order to tell it where to look for this library, you will set the LD LIBRARY PATH environment variable. To do this, run the command

   ```
   export LD_LIBRARY_PATH=<your current directory>
   ```

   with your current directory changed appropriately.
4. If you rerun "./uselib" it should now run.
5. Answer the first gradescope question.

Using Dynamic Libraries

1. Look at the source code for the shared library (sess3.h and sess3.c), and the source code for uselib (uselib.c). Run "make" to create both the shared library and the program that uses it, and run uselib to see what it does.
2. Create a new C file, tryshared.c, that uses the "sayHi" method from sess3.h.
3. Add new lines to the Makefile to compile tryshared.c. You should be able to use the Makefile entry for uselib.c as a guide. Try compiling just your new program, without recompiling libsess3.so.
4. Run your new program to see that it successfully uses the shared library code.
5. Now, in sess3.c, change the "sayHi" method so that it prints "361 rocks" instead of "Hi 361." Recompile ONLY the shared library by typing "make libsess3.so".
6. Run uselib and tryshared. Has what they print changed? Remember, you have not recompiled them.
7. Answer the second question on gradescope.

Creating Your Own Dynamic Libraries

1. Now, write code to create your own shared library, and a program that uses it. Use sess3.h and sess3.c as examples for your shared library, but you can write functions in your library that do whatever you want. Write code like uselib.c that uses the functions in your shared library. You can name both your shared library and your program anything you like.

2. Start adding code to the Makefile to compile the shared library you wrote. First, you will need to compile your code into a position independent object file, using the -fpic flag. (For libsess3.so, this is the line "gcc -c -Wall -Werror -fpic sess3.c".)

3. Next, add a line to compile the object file you just created into a shared library with the -shared flag. (For libsess3.so, this is the line "gcc -shared -o libsess3.so sess3.o"). The name of your shared library file should begin with "lib" and end with ".so".

4. Add lines to the Makefile to compile the code that uses your shared library. In order to do this, you will need to tell gcc where to you look for your library, and which library to look for.

   (a) Where to look for libraries: "-L**path**" tells gcc where to look for libraries. For libsess3.so, we use "-L$(PWD)", using the PWD environment variable to tell gcc to look in the current directory.

   (b) Which library to use: We use "-lsess3" to tell gcc to use "libsess3.so". gcc assumes that all libraries begin with "lib", and end with ".so" or ".a", so you just need to specify the rest of the name, starting with "-l".

5. Answer the remaining gradescope questions.

Solving Part 4 of Homework 2

1. To get binary 4 to run, you will need to create your own shared library, to replace the one it expects. This shared library will need to contain a specific function, with a specific name.

2. The error you get when you first try to run 4 should let you know the name of the shared library you need to create. Once you create a library with this name and make sure binary 4 knows how to find it (by setting the environment variables described in the first section), you should be able to get binary 4 running a little longer.

3. Now you need to figure out what functions binary 4 is trying to use in the shared library. If only you had some sort of tool that told you what library calls a binary was making . . .

4. Once you're calling a function with the right now, you need to figure out what 4 expects that function to do. Fortunately, 4 has some helpful assert statements that should tell you if the function is returning what it expects.