Lab 2: Program Profiling on Linux

Welcome to the 2nd lab of the VE473 Advanced Embedded System course! Today we have two goals. First, we will get familiar with the compilation, linking, and loading processes. By the end, you should be comfortable with the differences between static and dynamically linked programs, including binary size and execution time, and better understand how the runtime linker works. Optionally, you will learn how to create your own dynamic library. Second, we will use several important utilities like GDB for analyzing and debugging programs. Additionally, it will guide you through the use of several other (possibly new) utilities, such aspmap, nm, readelf, and ldd.

Project Description and Exercise

First, we need a program to compile, link, and debug. In this lab, we use the five programs in the lab_2 folder.

The lab codes are tested with GCC 8.3. To make everybody in the class is using the same compiler version please issue following command:

module add gcc-8.3.0

Please also add this command as a line in your ~/.bash_profile file, which will then run the command whenever you open a new terminal.

Second, Now, you will compile two different binaries for this program, using no compiler optimizations. Issue the following commands:

gcc arr_search.c -o arr_search_dynamic -O0 -lm

gcc arr_search.c -o arr_search_static -O0 -static -lm

Now, issue the command

ls -lh

This will tell you the size of each compiled binary.

Third, the time utility allows you to measure a program's execution time.

Use this utility to measure the execution time of your dynamically linked program first. Please use a number of iterations that allows it to run for more than 0.1s. We recommend a command like the following:

time ./arr_search_dynamic 100000

Once you have an appropriate number of iterations, write down this value, as well as the "real" value reported by the time utility.

Now, measure the execution time, with the same number of iterations, for your arr_search_static binary.

Fourth, The ldd utility, which stands for "List Dynamic Dependencies," can be used to print the shared objects orlibraries required by a provided binary. You can read about its syntax and usage with man 1 ldd You'll now use ldd to compare your two binaries. Issue the commands:

ldd arr_search_static

ldd arr_search_dynamic

Fifth, The nm utility, short for "names," prints the symbol table (name list) of the provided binary. You can read about its syntax and usage with man 1 nm Now, use nm to compare your two binaries. Issue the commands:

nm arr_search_static nm arr_search_dynamic

This will list the respective line, word, and character counts of the output.

Sixth, The Gnu Debugger (GDB) is a utility that should be in every Linux programmer's toolkit. Every programmer makes mistakes, and debugging is just as important as writing code. You should be familiar with GDB maybe from VE422, but if not (or if you need a refresher), we will examine some of its functionality. we will use breakpoints and watchpoints. Breakpoints allow us to interrupt execution of the program when a specific instruction address is reached, while watchpoints allow us to interrupt execution when a certain memory address is modified. Breakpoints and watchpoints are both extremely powerful, and in this lab we are just scratching the surface of what can be done with them.

First, you will need to recompile the original `arr_search.c` with extra debugging information:

gcc arr_search.c -o arr_search_debugging -O0 -g -lm

Next, have GDB load the compiled binary:

gdb ./arr_search_debugging

Seventh, Now, we will add a breakpoint to our program. This will cause program execution to pause when the breakpoint is reached. Add a breakpoint to the function call library_calls with the command:

b library_calls

You can have GDB run your program with the command r. Any values provided after r are passed as arguments to your program. Run your program for two iterations by issuing the command:

r 2

Eighth, At this point, GDB should have stopped at the breakpoint you specified. The c command allows you to continue to the next breakpoint. Issue this command

c

Ninth, The n command allows you to proceed to the next line of code. Issue this command so that the malloc call occurs and the values pointer variable points to a region of dynamically allocated memory. The p command allows you to print the value of a variable. Use this to get the address that is stored in the values variable by issuing the following command:

p values

Tenth, Now, you will set a watchpoint on the values array. Whenever this location is modified, GDB will pause execution. Set a watchpoint at this address using the command:
*watch *(float *) <address>*
For example, if the address of values is 0x602010, use the command:
*watch *(float *) 0x602010*
Now, we will also set a watchpoint on the next element of the values array. You can do this with the command:

*watch *(float *) (values + 1)*

Once you have set these watchpoints, press `c` to continue execution. Continue pressing `c` each time GDB interrupts program execution until the program execution exits.

Submission to each exercise:

0. List the names of the people who worked together on this studio.
1. No submission for the first exercise.
2. For the second exercise, please list the size of each one, and explain briefly why they are (or are not) different.
3. For the third exercise, please list:
   The number of iterations used
   The "real" execution time for the arr_search_dynamic binary
   The "real" execution time for the arr_search_static binary
   Why you think these execution times are (or are not) different
4. For the four exercise, please list the output of each command, and explain why they are (or are not) different.
5. Compare the outputs, and as the answer to this exercise, please tell us what you noticed.
6. Compare the outputs, and as the answer to this exercise, please tell us what you noticed.
7. Please list the output from issuing the command.
8. Please describe what happened after issuing command c.
9. As the answer to this exercise, please show the output from this command.
10. Please list what values the program assigned to the watched addresses.

Advanced Topic (not included in the lab grade):

1. Try to understand and play with other utilities, like readelf, pmap, and so on.
2. Generate a memory leakage with “Segmentation fault”. Then try to use GDB to locate this error.