GDB LESSON

Overview

This lab provides a simple introduction to the use of the GDB utility to debug a C program.

Performing the lab

The lab is started from the labtainer working directory on your Linux host (e.g., VM). From there, issue the command:

```
labtainer gdblesson
```

The resulting virtual terminal will include a bash shell.

Tasks (perform them as described)

1. **List all the files in the directory**.
Use
```
less sampleMath.c
```
to view the program. After looking the program over contemplate what the result of the program will be. Once you have done this, type "q" to exit the display. Next, type the gcc command into the shell as follows:

```
gcc -g sampleMath.c -o sampleMath
```

2. This will compile `"samplemath.c"` and create an executable version of the program called `"sampleMath".` To run this new program type:

```
./sampleMath
```
Note: the error.

Rather than looking at the C program alone, the program GDB will help us analyze what is throwing an exception. Type "`gdb sampleMath`" into the command line.

3. Once the program begins type "`r`" or "`run`" to start running the program. You will see that there is an "arithmetic expression" at the end of the program. To better understand the source of the problem, list the contents of the program with the list command. To repeat a command without having to re-type anything, just hit the Return/Enter key; doing this for the list command will display the next ten lines of code. Look at the code and figure out what line the "while-loop" begins on. Type "`break #`"where the "#" is the corresponding line number.

Use the “`run`” command again. Use the “`p`” or “`print`” command to display the value of any variable, just type its name after the “`p`” or “`print`” command. Type “`s`” or “`step`” to continue running one line through the program. Periodically print the value of the number until you see it reach zero. Once you realize what the issue with this code is type “`q`” to quit GDB. Then edit the source code of the program, e.g., with “`nano`” or “`vi`”, change the “while loop” condition to compare with the correct variable. Finally, execute the program.

4. compile “`sampleMath2.c`” into “`sampleMath2`”using gcc. Execute the program by typing “`./sampleMath2`”; notice the message it gives you, then pass it as an argument in that range by typing it after the name of the program separated by a space. Clearly the program is not running as it is intended, given that the values are different than expected. Once again, GDB is a useful tool for isolating what is going wrong with a program. Use the “`gdb`” command to open up “`sampleMath2`”

5. Run the program once using “`r #`” wherein the symbol “`#`” is the value of what parameter is being fed into the program for testing. Set a breaking point somewhere in the code and step through it. Print variables and try to figure out what value is causing the incorrect results. When you find out what needs to be changed, modify the code accordingly and input 342 into it.

Stop the Labtainer

When the lab is completed, or you’d like to stop working for a while, run

“`stoplab`”

from the host Labtainer working directory. You can always restart the Labtainer to continue your work. When the Labtainer is stopped, a zip file is created and copied to a location displayed by the `stoplab` command. When the lab is completes send that zip file to your instructor.