COMP 2560 Fall 2020

Lab 2

Deadline: Oct. 4, 11:59 PM

The main purpose of this lab is to

a) use the “script” and “scriptreplay” commands to record and replay a terminal session,

b) learn the basics of debugging using gdb, and

c) debug a program.

Part 1: The script and scriptreplay commands

We showed you in lab 1 how to record your whole computer screen using Powerpoint, a free online recorder, or quicktime (on a Mac). In fact, the Unix/Linux system itself has two commands that could be used to record your terminal session and replay it. Note it only records your active terminal window, not the whole screen of your machine.

The “script” and “scriptreplay” are very useful Unix/Linux commands to record and replay a terminal session. The recorded terminal session is saved in a text file and optionally together with an associated timeline file so that you could replay your terminal session later for demonstration and other purposes. **Most importantly in this course, the recorded terminal session will sometimes be required to be submitted. Therefore, it is very important you know how to use these two commands.**

Watch this Youtube video for typical use of the “script” and “scriptreplay” commands in Linux.

https://www.youtube.com/watch?v=tweyWNr6X18

This video shows a few ways to use the script command. Pay attention to the way script/scriptreplay command is used from 2:44s to 4:40s. That is what you will be using them for the semester if asked.

You could also check out the manual page for “script” and “scriptreplay” by typing

man script

man scriptreplay

at your shell prompt if you want to know more about these two commands.

Part 2: GDB - The tool to debug your program

1) GDB is a commonly used very useful program to debug almost any of your program under Linux/Unix. There are many excellent resources online explaining how to use gdb. To get you started quickly, I have chosen a very short tutorial that covers the essentials of gdb. Please follow the link below

https://condor.depaul.edu/glancast/373class/docs/gdb.html#Running_the_Program_being_Debugged

to read the whole page.

The document was written for debugging C++ programs. All the commands introduced in the document still apply to debugging C programs. The only two changes that you need are:

a) In the document, in "Preparation", step 1, it shows the command to compile a C++ program is `g++ -g printch.cpp -o printch`

Note in the above command, "printch.cpp" is the name of the C++ source code, and "-o printch" specifies the name of resulting executable file. If you do not specify the name of the output executable file, then it is named "a.out" by default.

Since we are coding C programs, you need to change the above command to

```
gcc -g sourcecode  -o executable_name
```

or simply

```
gcc -g sourcecode
```

(in this case, a.out will be the name of the resulting executable file by default.)

b) in the document, the command to run the gdb is as follows:

```
gdb -xdb -tui printch
```

Note "printch" is the name of the executable file.
On our CS Linux server, the "-xdb" option is not supported, use just

```
gdb --tui printch
```

2) Now, download the source code “copy.c” from “Weekly Lessons/Week2…..” on the course site in BB. This program will copy a file and its usage is as

```
Shell prompt>  ./copy source dest
```

(assuming the executable files is named copy)

For example,

```
Shell prompt>  ./copy copy.c   copy.bak
```

will make a copy of the file “copy.c” and the copy is called “copy.bak”

The program “copy.c” could be successfully compiled, but when run, it does not do what it is supposed to do but output a message “writing problem : Bad file descriptor”. According to this message and the source code of the program, the cause should be the file descriptor(s) is corrupted. Use GDB to launch a debugging session to inspect relevant variables to confirm the suspicion.

More specifically, perform the following steps and use the script command with timing option to record your debugging session.

a) Type the following to launch gdb to execute ./copy at the shell prompt:

   ```
   gdb  -tui  ./copy
   ```

b) Set up a breakpoint at the following statement in the source code:

   ```
   while((n1=read(fd1, buffer, 512) > 0))
   ```

c) Run the program by typing

   r copy.c   copy.bak

d) Your program will now execute all the statements before the while statement mentioned in b) where you set up a breakpoint and stops execution at the breakpoint, i.e., the while statement in b). Now you want to inspect the values in the variables fd1 and fd2. These variables should hold the return values (file descriptors) from the two “open(….)” function calls in the “if” statement just above the “while” statement in the source code. Take note of the values in fd1 and fd2.

e) Use the gdb command “n” to execute the while statement (and its enclosed read function) and gdb will stop at the next statement which is ,

   ```
   if(write(fd2, buffer, n1) != n1){
   ```

   inspect the values in the variables fd1, fd2, and n1.

f) Compare the values of fd1 and fd2 you saw in e) with the values of fd1 and fd2 you inspected in d). Are the fd1 and fd2 values the same in d) and e). What’s the n1 value? Is it what you expected?

g) Execute the

```
if(write(fd2, buffer, n1) != n1){
```

statement and remaining statements in the code using the gdb command “n” until the program terminates.

h) Quit the gdb.

Please submit

i) The script file and its associated timeline file recording your GDB debugging session from step a) to h) above.

ii) A short text file explaining what the cause of the problem might be (why the file descriptors are corrupted?). The answer to this question is probably beyond the scope of this course. Use your best guess.

(iii) It is also very easy to make changes to the “copy.c” source code so that it works as expected. Use the script command again with the timing option to record a terminal session in which you make the change, compile the changed code, and run the executable to demonstrate it works as originally expected. Submit the script file and the timeline file.

Please use proper names for the script files and timeline files needed for submission so that it is easy for marker to distinguish which pair of files is for which questions. For example, you could use script_i.txt and timinga_i.t for submission required in i) , and script_iiI.txt and timing_iiI.t for submission required in iii).

The deadline to submit is Oct 4, 11:59 PM.