COMP 2560 Winter 2024 — Lab 6

Part I

GDB is capable of debugging programs with multiple processes. In this lab, we will study how to debug a C program with fork functions.

You only need to know a few more GDB commands than what you learned in Lab 2 and follow a simple procedure detailed below.

By default, GDB will step into the parent process after the call to fork() and ***let the child process run unimpeded***. To debug both the parent and the child processes, using lab6.c as an example, do the following. Please note that lab6.c is almost identical to fork2.c which we studied and posted online. Observe the difference between lab6.c and fork2.c and think why the changes are in lab6.c.

1. Open two terminal windows. Open lab6.c file. See the screenshot below in Fig. 1.

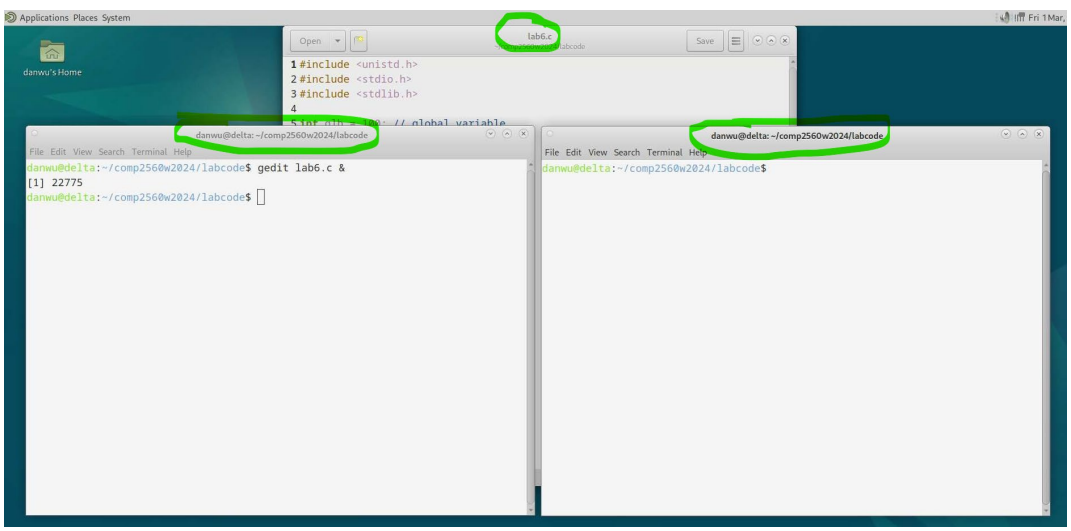

Figure 1: Step 1.

2. Pick one terminal for debugging the parent process and run GDB (e.g., gdb -tui ./a.out), set up a breakpoint on a statement after the call to the fork(…) (but not in any code the child process will be executing).

For example, in lab6.c (posted online), I set a breakpoint at line 31 “sleep(2);”. See Fig. 2. Note you

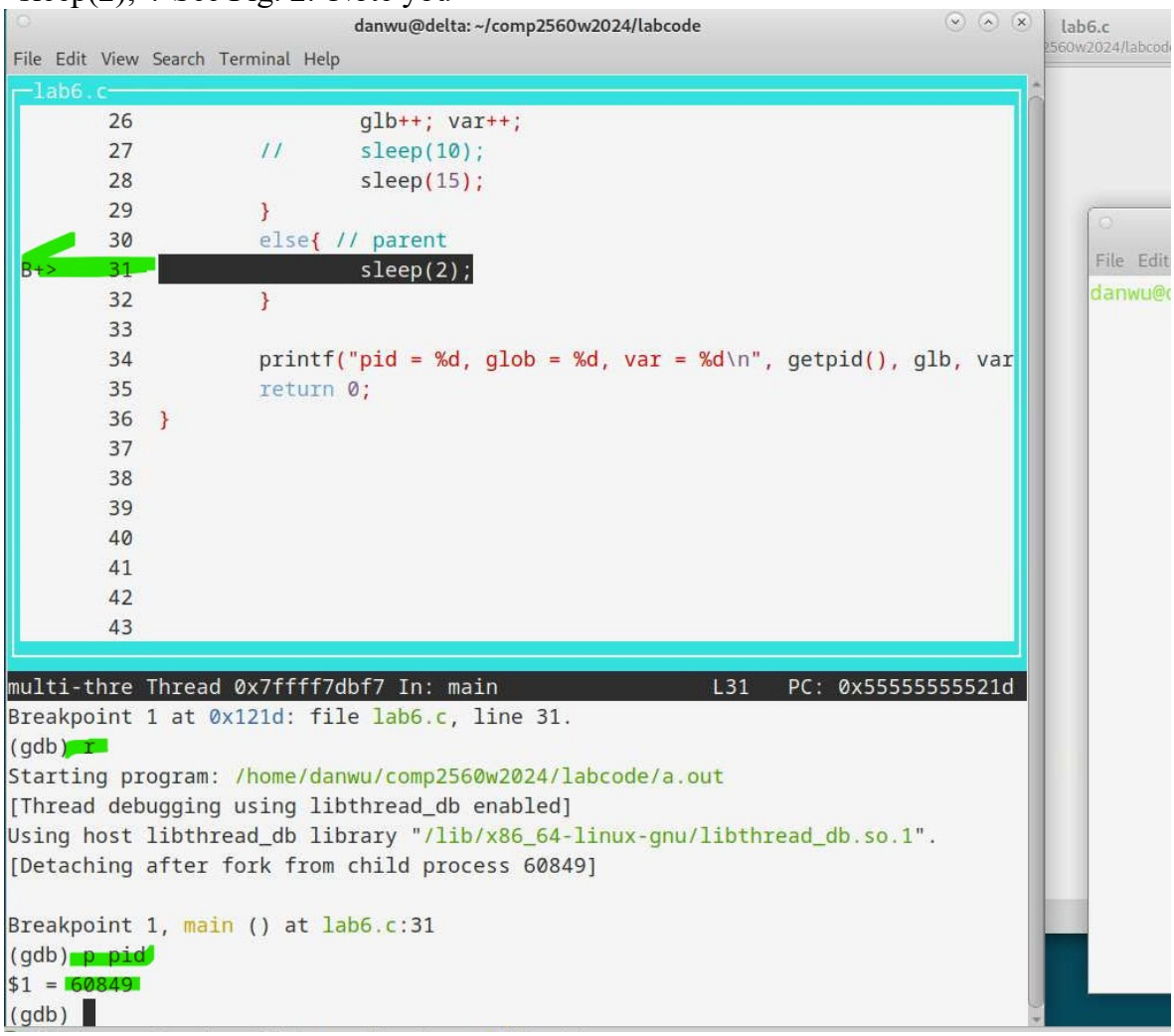

Figure 2: Step 2.

need to compile your source code file using “-g” option for debugging. For example, *“gcc -g lab6.c”*.

3. Run the parent process to the breakpoint. Note the value returned by fork(), i.e., the process ID of the child. See the screenshot in Fig. 2, from which you know the PID for the child process is 60849.

4. In the other terminal window, run GDB (just type “gdb –tui”, without “./a.out”). Then type the GDB command “attach 60849”. See below in Fig. 3. And then type a few “n” commands before you can see the source

code. See Fig 4.

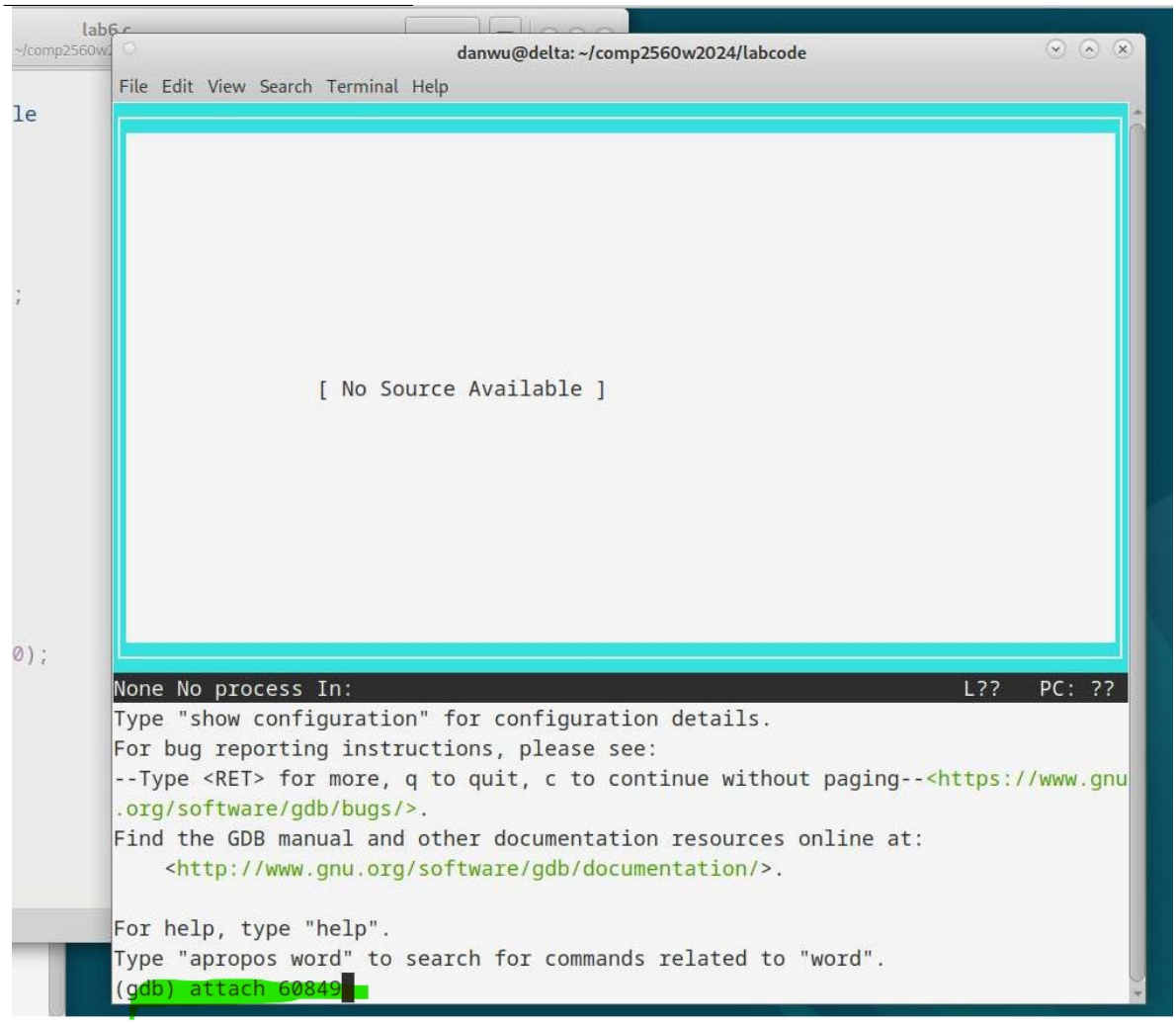

Figure 3: Step 3.

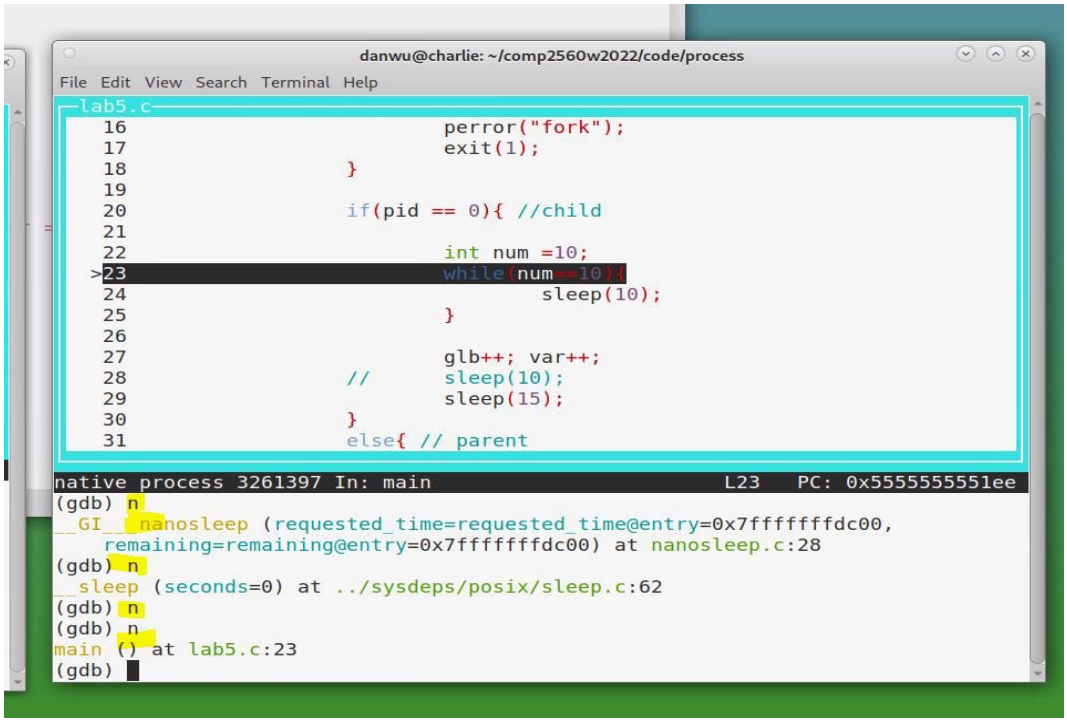

Figure 4: Step 4.

5. Now, in each terminal window, you can debug the parent process and the child process, respectively. You can see in the source code of lab6.c, an infinite loop from line 22 to 25 is deliberately added so that the child process is trapped in an infinite loop once you attach it to GDB. The actual work of the child process is in fact at line 27 and beyond.

6. How could you get out of the infinite loop of the child process so that you can debug the code starting at line 27? By changing the value of the variable "num". How? Use the GDB command "p" as shown in Fig. 5.

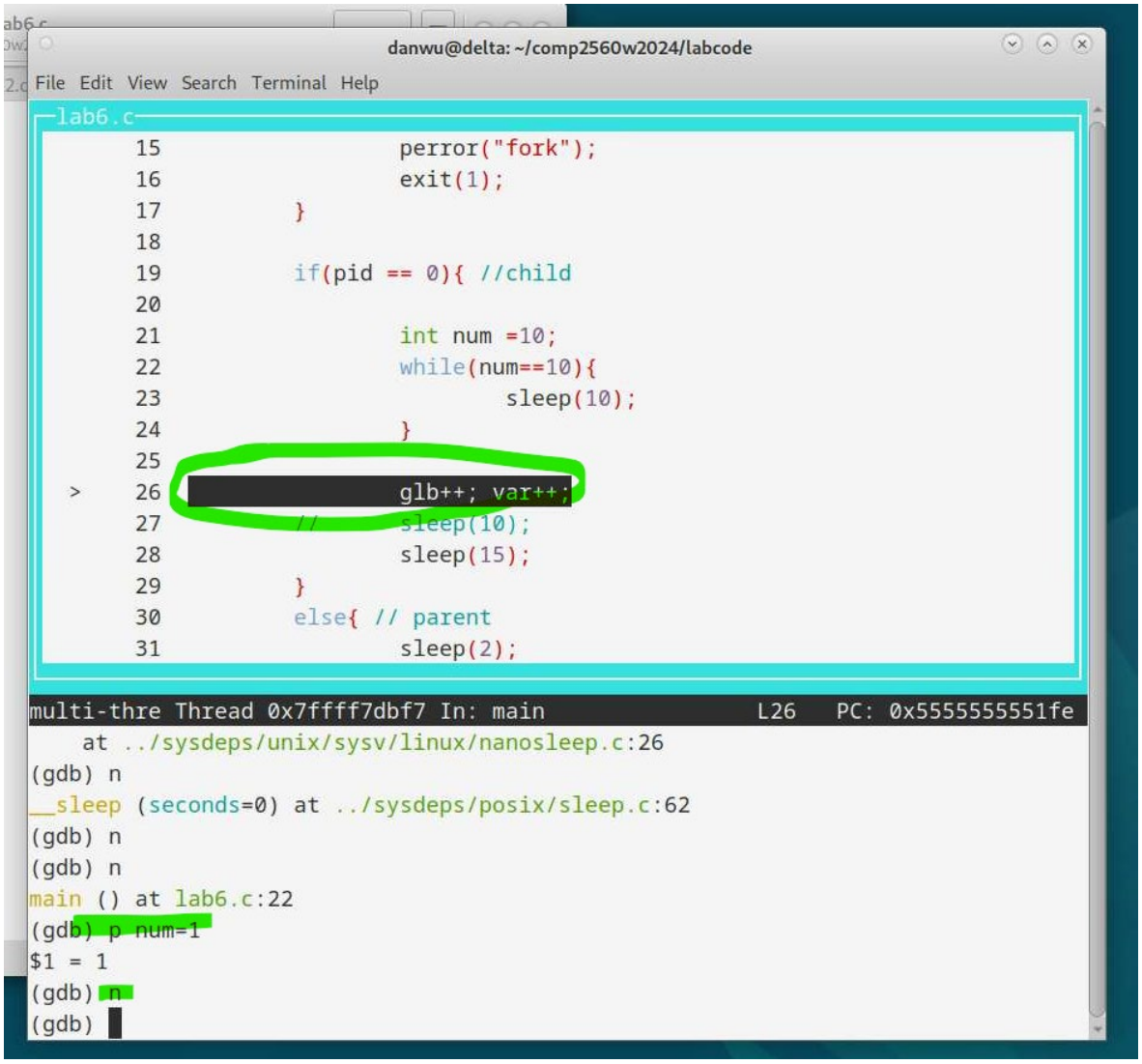

Figure 5: Step 6.

7. After the variable "num" has been assigned value 1 by the "p" command, you will be out of the infinite loop by typing the "n" command and you can now continue to debug the rest of the child's code. See Fig. 5 above.

The above example presents one possible way to debug a program involving fork (…) and child processes.

Part II

Question 1. Pick another simple program of your choice with fork (…) function and practice how to debug both the parent and child processes. (you may need to add an infinite loop in advance to trap the child process once attached). Record a short video with audio to demonstrate and explain the steps such that you can successfully attach the child process.

Question 2. Discover what happens within a parent process when a child closes a file descriptor inherited across a fork. In other words, does the file remain open in the parent, or is it closed there? You need to design and write a small program to investigate the answer to question 2. Put comments in code to explain your design idea.

Submission Requirement

1. The shareable link for the video for question 1.
2. The source code of Question 2 and a short text file explaining your design and finding.

Both are due by 11: 59 PM, Mar. 10.