CS 425
Lab 2
Matthew Irizarry

2.1 Global Variables

1. What are the values printed in Line A and be, respectively?

   Value A prints 20, and Value B prints 5.

2. Are these two values the same or different? Explain.

   To answer this, we must consider what the `fork()` function does when it is performed. In a more simple example, such as calling `fork()` before a “Hello World” print, it will print twice. However, one will execute as a child process and the other will execute as a child process.

   In this more complex example, the process gets forked, runs as a child first, and then the parent process completes. This is the reason that the value gets printed first, and then the value B gets printed.

2.2 Local variable pid

1. Based on the above program, what is your observation on the output? Explain.

   In this example, the process gets forked and it returns the PID of the parent, as it is above zero. Once this is completed, the child process is run, and then the PID that is returned is equal to zero. Therefore, the child process is being run. However, this order may not always be consistent as the OS decides which order these processes are executed.

2. Based on the modified program, what is your observation on the PIDs regarding the programs listed by the ps command?

   The output, shown below, indicates the PIDs were actually the same as displayed from the data printed to the standard output.

```
matt        9037  0.0  0.0   2772   948 pts/1   S+   22:47   0:00 ./pid.out
matt        9038  0.0  0.0   2772    92 pts/1   S+   22:47   0:00 ./pid.out
```

2.3 Counting Processes

1. How many processes are created (including the parent)?

   In the example with three `fork()` functions one after the other, there will be 2^3 processes created. We can add a simple “hello world” print after the three fork() functions and see that it prints to the screen eight times. If we modify the code to display the process IDs to the standard output, we can count the number of unique processes that are created. In my case, the PIDs created were 10297-10303. These are the seven “parent” processes that are created, and the final one is classified as a child process. If we consider the tree that actually gets created here, we understand why there are 7 processes that are classified as parents, and only one is classified as a child.

2. Can you count processes by simply using local variables similar to the sample code we showed in the lecture?

   I believe you can, because in the example, it shows 7 unique parent process IDs, and the child process is only instantiated once. We can confirm this by adding a sleep function at the end of the program, and looking at the process IDs associated with this program. When doing this, we see that there are indeed 8 processes that run, and that follows the rule of 2^n (where n is the number of forks) called.

2.4 Seeing a Zombie: A special process state…

1. Can the kill command kill the child of seezombie? Explain.

   After running the compiled seezombie code in the background, the PID of the process is printed to the stdout. We can confirm this is the correct PID

because we can use the command `ps aux | grep seezombie` and see that there are two processes running. One is the parent process, and the other is the child process. Here is what my `ps aux | grep seezombie` returns.

```
matt   11245  0.0  0.0   2640   948 pts/1 SN   23:20   0:00 ./zombie.out
matt   11247  0.0  0.0      0     0 pts/1        ZN   23:20   0:00 [zombie.out]
<defunct>
```

We can see the parent process is 11245 and the child process is running with a PID of 11247. Attempting to run the command `kill -9 11247` which would ideally kill the child process is unsuccessful. Why does this happen? Because the parent process is sleeping for 100 seconds which is keeping the child alive, metaphorically speaking of course. Even when you try and kill the child process, it will not respond because the parent process is basically overruling the instructions and saying that it needs a response from the child process and is constantly keeping it running.

2. Can the kill command kill the process seezombie? How about its child? Explain.

   The kill command can absolutely kill the parent process, and when doing so, also kills the child as well. When a process receives a SIGTERM, it gracefully shuts down the process and attempts to shut down the children as well. However, there are some instances where even this won't kill child processes. For example, if the child processes spawn daemons that run independently of the parent process. In this example, however, the child process runs because the parent process calls the fork() function, which tells the OS that when the parent shuts down, so should all the children.

CS 425
Lab 2
Matthew Irizarry

3. Can you see a zombie process if you remove/comment the statement sleep(100), or change sleep(100) to sleep(1)? Explain.

   If you change the sleep(100) to sleep(1), you can still see the zombie process. However, if you remove the sleep function entirely, the process finishes and terminates too quickly to see any zombie process.

4. Can you see a zombie process if you remove/comment the statement wait() and the if statement? Explain.

   Removing the wait and if statements still results in a zombie process, as the sleep function is still timing out the function.