 # Name: Nicholas Fotinakes

 # Date: 2022-03-15

 # Title: Lab2 - Questions and Observations

 # Description: This pdf contains answers to Lab2 questions and additional observations

Step 3 Question: Run the program by typing ./ExecutableName and take a note of your observation.

When running the program without passing an argument I get "Segmentation fault (core dumped)" error printed. This is due to the process trying to read a memory location for which it doesn't have permission, and thus an exception is being sent to the kernal. The program for steps 1-4 is expecting to read an argument passed when the program is executed (command line argument) which will tell the usleep() how long to sleep for. By not passing the command line argument upon execution, we get this error. Safeguarding against segmentation fault could be included in the source code.

Step 4 Question: Re-run the program by typing ./ExecutableName 3000. Note that the delay in the loop depends on the command line argument you give, here the delay is 3000 microseconds. Enter delays of 500 and 5000, what happens?

When using 500 as the command line argument, the print statements of each process execute quicker, as the delay is only 500 microseconds. When passing 5000, the print statements for the parent and child print slower, as there is now a delay of 5000 microseconds. I even tried using 500000 which would be a delay of .5 seconds to see the print statements happening much slower.

Observations:

When trying to redirect the output for the first part of Lab 2 to a file, I noticed the redirected output did not match the output in the terminal. The terminal shows the processes each switching back and forth as the parent and child execute after the fork. When redirected to a file, each parent and child loop executed fully and the "Before forking" statement even printed twice. The only thing I could find as causing this issue was the line buffer being different when redirecting the output to a file. If I manually flush the line after each print statement using fflush(stdout) I then get redirected output that matches the program output in the console. NOTE: After speaking with Professor Lamble I learned this is expected for the redirection as the one process is granted I/O first and the other blocked until completion. She mentioned I could use fflush() if I wanted, so I decided to leave it in.

Based off the Professors suggestions I removed the code that involved passing a command line argument for Step 5, as this step did not need it or call the sleep function. I decided to keep the “Before forking” portion of code from the previous steps.

Snapshot of some of step 4 output (full test verification in output text file):

```
nickf@nickf:~/cst334/wk2$ ./lab2 500

 Before forking.
                        Parent Process 0
Child Process 0
                        Parent Process 1
Child Process 1
                        Parent Process 2
Child Process 2
Child Process 3
                        Parent Process 3
Child Process 4
                        Parent Process 4
                        Parent Process 5
Child Process 5
Child Process 6
                        Parent Process 6
                        Parent Process 7
Child Process 7
```

Snapshot of step 5 (also included as separate png file):

```
nickf@nickf:~/cst334/wk2$ ./lab2step5

 Before forking.
lab2  lab2.c  lab2step5  lab2step5.c  Makefile
Child Complete
nickf@nickf:~/cst334/wk2$
```