Fork.c

1. Execute the program to understand and answer each question mentioned in the source code file
    a. Get the program back to the original state for each question
    b. Question 1: Which process prints this line? What is printed?
        i. **The child process prints this after the fork, upon return from the child process the line is printed again within the parent process. The current process id is displayed.**
    c. Question 2: What will be printed if this like is commented?
        i. **If the line is commented the contents of the current folder will no longer be printed. Now the "print after execlp" will print.**
    d. Question 3: When is this line reached/printed?
        i. **This line is reached once the fork has been implemented setting the child return -0 and allow the else if statement to be entered, following the removal of the line in question 2 the "print after execlp" prints.**
    e. Question 4: What happens if the parent process is killed first? Uncomment the next two lines.
        i. **The parent is returned, and the after process id =0 is printed, followingthis the child process is returned as the zombie is reaped and the contents of the current folder are printed as well as the "in Child:.., Parent:..." statements.**

Mfork.c

1. Execute the program once to understand and answer the question
    a. Question 1: How many processes are created? Explain.
        i. ***Parent process calls fork 1 after print***
            1. ***<u>Fork 1</u>***
                a. ***Prints 2 then forks 2***
                    i. ***<u>Fork 2</u>***
                        1. ***Prints 3 forks 3***
                            a. ***<u>Fork 3</u>***
                                i. ***Returns***
        ii. ***Parent process calls fork 2 after print***
            1. ***<u>Fork 2</u>***
                a. ***Prints 3 then forks 3***
                    i. ***<u>Fork 3</u>***
                        1. ***Returns***
        iii. ***Parent process calls fork 3 after print***
            1. ***<u>Fork 3</u>***
                a. ***Returns***
        iv. **<u>Underlined</u> above represent the processes created, a total of 7 processes get created.**

Pipe-sync.c

1. Update the program to answer the question in the source code file.
    a. **DONE**

Fifo_producer.c and fifo_consumer.c

1. Create a fifo and open it for writing and reading respectively
    a. Use slides 37 and 38 in Chapter 3
2. Compile the programs
3. Open 4 terminals and answer the following questions
    a. What happens if you only launch a producer (but no consumer)?
        i. **The program hangs, waiting for a consumer.**
    b. What happens if you only launch a consumer (but no producer)?
        i. **The program hangs, waiting for a producer.**
    c. If one producer and multiple consumers, then who gets the message sent?
        i. **It seems to be random, while I know it is not a cant decipher why it jumps from terminal to terminal.**
    d. Does the producer continue writing messages into the fifo, if there are no consumers?
        i. **No it errors with "BROKEN PIPE"**
    e. What happens to the consumers, if all the producers are killed?
        i. **If the producers are killed the consumer exits with "Read 0 bytes"**

Shared_memory3.c

1. Understand the code
2. Compile/execute the program
3. Question 1: Explain the output.
    a. **The string before the fork is first string in both buffers**
    b. **Once forked the fork prints the falues of both buffers that it inherited which is first string**
    c. **After copying in STR 2 into both of its buffers and exiting the do_child the values of the buffers checked are different**
    d. **This is because after forking and updating both the shared and unshared of the child process only the value of the shared buffer has changed, and after leaving the child process the unshared buffer it contained was effectively lost and only shared buffer was updated by the child process. When checking after child process exits the value of the parents shared will have been updated by child whereas its unshared will have remained unchanged.**

Thread-1.c

1. Compile and execute the program
    a. Gcc -o thread1 thread-1.c -pthread
    b. ./thread1
2. Observe execution and answer the two questions referenced in the source code file
    a. Question 1: Are changes made to the local or global variables by the child process reflected in the parent process? Explain.
        i. **No they remain unchanged because the fork instantiates a separate process that does not have a defined shared memory space, this means when child process exits none of the variables of the parent process have been modified by the child.**
    b. Question 2: Are changes made to the local or global variables by the child thread reflected in the parent process? Separately explain what happens for the local and global variables.
        i. **Yes, after thread exits the variables have been assigned eachothers values via the child_fn function. The reason these are changed unlike the fork is because a thread is within the process of the "parent" fork creates a "copy" of the parent and then works in its own space, whereas a thread shares space with the process that called it meaning modifications it makes to variables in its scope are in the same scope as the calling process, the scope of global and local are similar.**