Chapter 6

Purpose: This assignment is to gain more experience with processes and to introduce threads.

Part 1) Processes

Begin with a copy of your `fork.c` program, call this copy `fork2.c`.

Before the fork, put a prompt "sleeping"; then sleep for 5 seconds.

When you test this program, while the process is sleeping you will type 20 characters followed by a return. Actually if you type the alphabet (a-z) you will be OK here. If you have trouble getting the alphabet in, increase the sleep time.

In both child and parent, just before you print, the process should read a character.

When the process printes, it should print the character just after the number.

Details:

To read a character use the `getchar` system call. Be careful, it returns an integer, that's because it wants to return a -1 on end of file. You will need an integer variable, but when you print it, treat the variable as though it were a character.

To print a character, add the variable to the list of items to be printed by the printf and add a `%c` to the format string of the printf.

Observe: which process (parent/child) prints which character.

Part 2) Threads

Make a copy of your `fork2.c` (from above) and call it `thread.c`

Turning this into a thread program is a little bit of a pain.

First, move all the top variables in main (like the loop counter) up to the top of the program. This makes them global.

Replace the fork call with a call to create a thread.

Copy the loop (and its contents) into a procedure named `ChildThread`. This procedure must be below the global variables moved earlier.

Since we no longer have process parent child behavior, set sleep in the procedure to 2 (the child) and sleep in the main loop to 1 (the parent).

The thread you create will run the loop inside the procedure.

Do not worry about the waits at the end, you can comment those out. (we could use pthread_join to replace the wait by changing the detached state, but that is not necessary for this assignment).

Observe and note the behavior. Note: Since we moved the loop counter up top to a global, the two threads are deliberately using the same loop variable; notice what happens, and note it.

Now add a variable called `i` (this is the name of your loop variable); the loop variable will now be local to the procedure.

Observe and note the behavior.

Now remove (comment out) the sleep call from inside both loops.

Observe and note the behavior.

Demo: The final version of the thread.c

Submit: What you observed and noted on a handwritten or printed sheet.

Make sure that both fork2.c and thread.c are in your home directory on the server. Your instructor will examine fork2.c. Please leave it named as such.