Schuyler Asplin
CSCD 340
ch5 homework (code)

1) They both change the variable with no problem. Even though it's named the same thing in both files, it's a different process so they are two different pointers which can changed separately.

2) Both parent and child try to write to the same file, but only the child writes it's message out. This makes sense since the program is syncronous and the parent will complete the fork method, triggering the childs. I'm not getting any errors for attempting to open it, and both open methods return a file-descriptor of 3.

3) Without any added code, the parent prints first. I'm a little confused as to why the child isn't printing first, but in any case, adding the pause ethod and a SIGCONT fixes the ordering. It actually works with just the pause method, so I'm not sure that my signal method is correctly implemented since it
s not changing the behavior, but the output is corerct.

4) it made the child process run ls!!! so cool :)
execlp and execvp use PATH environment variables so entire path doesn't have to be entered. (I still need to use "/bin/ls", just "ls" doesn't work).
execle lets us specify the environment. Of the process. I specified hostname and port variables, and told it to print it's environment. Had to do some reading on env variables and talk to friends, but these are like global variables for the process that we can access by preceding it with a $.
execlp is like execl but it uses your PATH environment vairable. I haven't been able to get it to work but my understanding is that it'd let me specify "/ls", instead of "/bin/ls".
Seems like execvpe lets you specify environment variables and also include your own PATH.

5) Ok so if I call wait in both the parent and the fork, the parent conditional print pid first, then the child, then the child wait() executes and returns -1 signifying an error, and then the parent's wait function executes and returns the childs pid. CRAZY!
Ok so: waiting in parent, wait gives us the id of the process it's waiting on (parent is waiting for child).
Since the child process doesn't have any children to wait on, it returns an error (-1). This is because wait() causes a process to wait until one of it's children terminates.

6) waitpid() is useful anytime we have multiple children. wait would wait for any of them to terminate, but we can specify a child to wait on with waitpid(). It also gives us lots of options to specify certain behaviors. I modified p5.c to have 2 children, and the parent calls waitpid with a pid arg value of -1, meaning that it'll wait for any child with the same group pid as it's own. We can tell by the output that it's working, since parent prints it's pid first, but then it waits for child2 to continue. That value that itprints when done waiting for child2 is child2's pid.

7) It compiles and runs but the child's printf doesn't get printed to stdout.

8) Please don’t give me any credit for this problem - I haven't been able to figure this one out. I think I'm getting close with read and write but I haven't been able to decipher any online resources that are using pipe. I used a lot of the ostep code but that doesn't use the pipe command.