ASSIGNMENT #2

practice system calls such as fork(), pipe()

CS 480 100 points Spring 2024

Check Blackboard for due date. Due by 11:59 PM.

Write a Linux program in C or C++ which uses `fork()` to start child process(es), and also uses `pipe()` to do inter-process communication.

You will first call pipe() twice to create two pipes for communication among the processes. Let us call them pipe_a and pipe_b, respectively.

You will then call fork() twice to create 3 additional processes. Among them, one process is the child of the initial process, but it is also the parent of another process. We call this process "Intermediate Parent".

The pipe_a is used for communication between the initial process and one of its child processes (the one that is not the intermediate parent). The parent will write a string to the pipe – "Go do some chores." . And the child will read from it. Similarly, the pipe_b is used for communication between the intermediate parent and the grandchild. The intermediate parent will write a string to the pipe – "Go do some chores.". And the grandchild will read from it.

Each child process or intermediate parent process will print out some information:

```
Child:  My PID is xxxx, my parent's PID is yyyy. I read from pipe A: Go do
       some chores.

Child:  My PID is xxxx, my parent's PID is yyyy. I read from pipe B: Go do
       some chores.

Intermediate Parent: My PID is xxxx, my parent's PID is yyyy. My child is
       wwww. I write to pipe B: Go do some chores.
```

After printing this information, the child or intermediate parent process should sleep for 3 seconds, then print the following message and exit:

```
Child zzzz:  I am awake.

Intermediate Parent xxxx: I am awake.
```

In the (initial) parent process, print the message:

```
Parent:  My PID is xxxx, my parent's PID is yyyy. My Children are wwww,
       zzzz. I write to pipe A: Go do some chores.
```

Then invoke the command "`ps -f --ppid …`" from your program to show all the processes involved. You can use the system() function to do this. After "—ppid", you must list the PARENT process ids of all the processes you want to list. For example, "*/bin/ps -f --ppid 26560,26803,26804,26805*".

See manual for ps command for more details ("*man ps*"). You should *not* use a general ps command here without specifying the parent ids of the processes of interest. Make sure your ps command syntax works on *turing.cs/hopper.cs*, because different variants of operating systems and shells may have some variations on command options.

If you use system(), which starts another process, you should see 5 processes in the output list (see sample output later).

The parent process should wait for all the child processes to complete, then print the following message and exit:

```
 Parent:  Child processes finished their work.
```

Note that each message is labeled according to the process that printed it, parent or child or intermediate parent. When multiple messages are printed from the same process, they should appear in the order in which they are printed. Each message should be single-spaced with a blank space between messages.

An example output:

```
Parent: My PID is 15513, my parent's PID is 15471, my children are 15514, 15515. I write to
pipe A: Go do some chores.

Parent: Issuing command: /bin/ps -f --ppid 15471,15513,15514,15515

Intermediate Parent: My PID is 15514, my parent's PID is 15513, my child is 15516. I write to
pipe B: Go do some chores.

Child: My PID is 15516, my parent's PID is 15514. I read from pipe B: Go do some chores.

Child: My PID is 15515, my parent's PID is 15513. I read from pipe A: Go do some chores.

UID        PID  PPID  C STIME TTY          TIME CMD
jzhou    15513 15471  0 11:30 pts/2    00:00:00 ./a.out
jzhou    15514 15513  0 11:30 pts/2    00:00:00 ./a.out
jzhou    15515 15513  0 11:30 pts/2    00:00:00 ./a.out
jzhou    15516 15514  0 11:30 pts/2    00:00:00 ./a.out
jzhou    15517 15513  0 11:30 pts/2    00:00:00 sh -c /bin/ps -f --ppid
15471,15513,15514,15515

Child: 15516 is awake.
Intermediate Parent: 15514 is awake.
Child: 15515 is awake.
Parent: Child processes finished their work.
```

The interlacing of messages printed by different processes will depend on the timing of your program and does not have to match the example output.

Requirement:

Use the *man* command to learn usages of the following system calls or functions. Some of them (although not all) are needed for this project.

```
getpid( )    getppid( )    wait( )    waitpid( )    execlp( )    system( )    setbuf( )
```

Output produced by `system()`, which starts a separate subtask, could come out while previously printed output is still in a buffer, and you may get garbled output. It is suggested that you call *`setbuf(stdout, NULL)`* at the beginning of the parent block to get unbuffered output.

Every time you make a system call, you must check the return value to make sure the call is successful before you proceed.

Programs must be consistently indented and commented so that a reasonable person can understand them. Use of descriptive variable names is always encouraged as an aid to readability.

Administration:

Submission is the same as before. Steps are repeated below:

You will submit a compressed file through Blackboard. The bundled file contains your source code and a Makefile. It should be named as “your-zid_project2.tar” and must be created following the procedure described below:

1. Put all your source code files (NO OBJECT or EXECUTABLE FILES) and your Makefile in a directory called “your-zid_project2_dir”. Example: z1234567_project2_dir. **Note**: ‘z’ must be in lower case.

   In your Makefile, you need to make sure your compilation produces the executable file called “your-zid_project2”. For a student with z1234567 as the zid, the executable would be *z1234567_project2*.

2. In the parent directory of your-zid_project2_dir, compress this whole subdirectory by the following command:

   ```
   tar –cvvf your-zid_project2.tar your-zid_project2_dir
   ```

   Example:

   ```
   tar –cvvf  z1234567_project2.tar      z1234567_project2_dir
   ```

“your-zid_project2.tar” is now the compressed file containing the whole subdirectory of your files. You can then transfer (e.g. using an ftp client) the tar file from turing (or hopper) to a computer on which you can open a web browser for your final submission to the Blackboard system.