ASSIGNMENT #1

Practice system calls such as fork()

CSCI 480 100 points Spring 2022

Check Blackboard for due date. Due by 11:59 PM.
Late penalty as in syllabus.

Write a Linux program in C or C++ which uses `fork` system call to start child process(es).

You will 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".

Each child, grand child or intermediate parent process will print out information like below:

```
CHILD:  My PID is xxxx, my parent's PID is yyyy.

GRAND CHILD:  My PID is xxxx, my parent's PID is yyyy.

INTERMEDIATE PARENT: My PID is xxxx, my parent's PID is yyyy. My child is
       wwww.
```

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

```
CHILD: zzzz is awake.

GRAND CHILD: zzzz is awake.

INTERMEDIATE PARENT: xxxx is 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.
```

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. The option "—ppid" *must* be followed by the PARENT process ids of *all* the processes you want to list. For example, "*/bin/ps -f --ppid 26560,26803,26804,26805"*. Manual page for ps command has more related details:

> *" --ppid pidlist*
>
> *Select by parent process ID. This selects the processes with a parent process ID in pidlist. That is, it selects processes that are children of those listed in pidlist. "*

You should *not* use a general ps command here without specifying the parent ids of the processes of interest.

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:  Children processes are finished.
```

Note that each message is labeled according to the process that printed it, parent or child or grand 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 26803, my parent's PID is 26560, my children are 26804, 26805.

INTERMEDIATE PARENT: My PID is 26804, my parent's PID is 26803, my child is 26806.

CHILD: My PID is 26805, my parent's PID is 26803.

GRAND CHILD: My PID is 26806, my parent's PID is 26804.

PARENT: Issuing command: /bin/ps -f --ppid 26560,26803,26804,26805

UID PID PPID C STIME TTY TIME CMD

jzhou 26803 26560 0 09:42 pts/2 00:00:00 ./a.out

jzhou 26804 26803 0 09:42 pts/2 00:00:00 ./a.out

jzhou 26805 26803 0 09:42 pts/2 00:00:00 ./a.out

jzhou 26806 26804 0 09:42 pts/2 00:00:00 ./a.out

jzhou 26807 26803 0 09:42 pts/2 00:00:00 sh -c /bin/ps -f --ppid 26560,26803,26804,26805

INTERMEDIATE PARENT: 26804 is awake.

CHILD: 26805 is awake.

GRAND CHILD: 26806 is awake.

PARENT: Children processes are finished.

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

Requirement:

Use man command to learn usages of the following system calls or functions. Some are useful for this project.

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

You are required to check the return value of fork() system call.

Note on output buffer:

Output produced by `system`, which starts a separate subtask, could come out while previously printed output is still in a buffer. For this assignment, it is ok if your program produces interleaved outputs from different processes (for every single process, the messages should appear in order, as explained before).

If you are interested, you can study setbuf()/setvbuf() to use unbuffered output. For example, use `setbuf(stdout, NULL)` at the beginning of your code can help get unbuffered output. The behavior of setbuf can be compiler/system specific so the use of it is not required for the assignment.

Coding style and documentation standards:

Programs must be consistently indented and commented so that a reasonable person can understand them. Use of consistent descriptive variable names is also required.

At a minimum, follow the following rules that you have used in earlier programming courses, including the use of a header box at the top of the program:

http://www.cs.niu.edu/~byrnes/csci240/240doc.htm
http://www.cs.niu.edu/~mcmahon/CS241/c241man/node6.html
http://www.cs.niu.edu/~mcmahon/CS241/c241man/node7.html

Administration:

All students in this class should have LINUX computing accounts on the host *hopper.cs.niu.edu or turing.cs.niu.edu*. Your login name for your LINUX account is your z number (with a lower case 'z'). Read the instructions on department webpage: Campus Experience – Resources. If you have issue, contact Dr. Kirk Duffin: duffin@niu.edu.

Submit your source codes on Blackboard. We will compile and run them on `turing`. No credit if your program does not compile on `turing`.

The compressed file contains your source code and a Makefile. It needs to be named as "your-zid_project1.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_project1_dir". Example: z1234567_project1_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_project1". For a student with z1234567 as her zid, the executable would be *z1234567_project1*.

   In addition, in your Makefile, you need to include the pseudo-target `clean` to remove object code and executable files. This is to help the TA to clean up their directory since their size quota is limited. Example:

```
clean:
    -rm *.o z1234567_project1
```

2. In the parent directory of your-zid_project1_dir, compress this whole subdirectory by the following command:

```
tar  -cvvf  your-zid_project1.tar  your-zid_project1_dir
```

   Example:

```
tar -cvvf z1234567_project1.tar      z1234567_project1_dir
```

"your-zid_project1.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.

Grading:

When the program is graded, a script runs a sequence of the following commands:

tar –xvf z1234567_project1.tar
cd z1234567_project1_dir
make
./z1234567_project1

These procedures should yield a working program. You need to verify by yourself that your compressed tar file can be opened properly. There will be a penalty if programs are incorrectly named or otherwise do not follow directions.

The grading of this assignment will be based on output correctness (40%), programming (40%), coding style and documentation (20%), under the condition that it complies and runs on `turing`.

For full credit, your program must follow the specs and documentation standards above. In other words, it is possible that you will get deduction even when your program is generating correct results if your coding style is bad. Here are some examples of bad style:

1. Lack of necessary documentation/comments. Or have incorrect comments;
2. Hard to understand variable names or function names;
3. Hard-coded magic numbers without obvious meaning or explanation;
4. Inconsistent style of indentation or curly braces to the point of reducing understandability.

If the variable name or function name is self-explanatory, comments can be brief or omitted. Ask yourself – will you still understand your code six months later?