HW2: Programming Assignment **v1 9/12/21, 8PM**

Working with Parent and Child processes

The objective of this assignment is to write and test a program that creates three child processes and obtains status information from them. It uses fork(), exec(), wait() and WEXITSTATUS(status) commands.

Due Date: 9.16.2021 11 PM

Extended Due Date with 20% penalty: 9.17.2021 11 PM

1. Purpose

Write a C program called `Initiator` that reads strings from a file, whose name may be provided as a command line argument to the program. `Initiator` forks three child processes for each line, that will run programs `Pell`, `Composite`, and `Total`.

2. Description of assignment

You will be creating four programs: `Initiator.c`, `Pell.c`, `Composite.c`, and `Total.c`.

`Initiator.c`: `Initiator.c` takes a mandatory argument that is the name of the .txt file which contains a string. The `Initiator` will read all the lines from the text file, and send each line to the child processes for further processing.

a. The `Initiator` is responsible for executing the fork() functions to launch the child process.
b. Each child process runs the exec() function to run the program needed (`Pell`, `Composite`, or `Total`), while also supplying the arguments that the new program needs to complete its execution.
c. The wait() function is used to wait for the completion of the execution of the child processes, and WEXITSTATUS(status) function is used to obtain the result (as an eight-bit integer) from the three child programs.

The Initiator then saves the status. After all the processes are complete the Initiator will output the counts of each type of characters (a code snippet is provided below).

`Pell.c, Composite.c, Total.c`: Each of these programs receives *one* line as an argument. They print the respective numbers and return the result.

All print statements must indicate the program that is responsible for generating them. To do this, please prefix your print statements with the program name. The `Initiator` should indicate the process ID of the child it forked, and the `Pell`, `Composite`, and `Total` programs should indicate their own process ID. The example output section below depicts the expected format of the output and must be strictly adhered to.

A good starting point is to implement the functionality for the `Pell.c`, `Composite.c`, and `Total.c` programs, and then write the code to manage its execution using the Initiator program.

3. Input and Output

For example, the "input.*txt*" files contain the string "10" and "3" (provided in 2 separate lines)".
Use fopen() function to read the string from the file.

Notes:

- You can assume that the input numbers to be between 1(inclusive) and 25 (inclusive).
- Note that the end of a line is indicated differently in text files for Windows and Linux system.

4. Task Requirements

1. The `Initiator` must read the characters from the .txt file, the name of which will be passed as an argument to it. Then send each line(String), one at a time, to the child processes. Each of the other three programs must accept the string as an argument.

2. The `Initiator` should spawn up to 3 processes using the fork() function for each line from the input file and must ensure that one full cycle of fork(), exec() and wait() is completed for a given process before it moves on to spawning a new process.

3. Once it has used the fork() function, the `Initiator` will print out the process ID of the process that it created. This can be retrieved by checking the return value of the fork() function.

4. Child-specific processing immediately follows. The exec() function loads the `Pell`/`Composite`/`Total` program into the newly created process. This exec() function should also pass the value to the `Pell`/`Composite`/`Total` program. For this assignment, it is recommended that you use the execlp() function. The "man" page for exec (search for "man exec") gives details on the usage of the exec() family of functions.

5. When the `Pell/Composite/Total` program is executing, it prints out its process ID; this should match the one returned by the fork() function in step 3.

6. The `Pell/Composite/Total` program then prints the respective numbers (refer to 6.d) and returns the result(refer 6.a,b,c).

    a. Pell should return the nth pell number (n being the value of string).
    b. Composite should return the nth composite number (n being the value of string).
    c. Total should return the sum of first n numbers (n being the value of string).
    d. respective numbers should be printed from the child process.

7. `Pell/Composite/Total` program should return the result. Each value is received by the `Initiator` and then should be printed. You can use the WEXITSTATUS() macro to determine the exit status code (see man 2 wait).

8. Parent-specific processing in the `Initiator` should ensure that the `Initiator` will wait() for each instance of the child-specific processing to complete. Once all the processes are complete output the result as mentioned in 6.a,b,c to the terminal.

For ease of grading, your program **must** fork()/exec() the programs in this order for each Number: `Pell-Composite-Total`.

5. Example Outputs

1. This is the output when analyzing the file input.txt which contains the string:-
10
15
(Note: your process IDs may be different)

```
denver%machine%

Initiator[735610]: Forked process with ID 735611.
Initiator[735610]: Waiting for Process [735611].
Pell[735611] : Number of terms in Pell series is 10
Pell[735611] : The first 10 numbers of the Pell sequence are:
0, 1, 2, 5, 12, 29, 70, 169, 408, 985,
Initiator: Child process 735611 returned 217.
Initiator[735610]: Forked process with ID 735612.
Initiator[735610]: Waiting for Process [735612].
Composite[735612]: First 10 composite numbers are:
4, 6, 8, 9, 10, 12, 14, 15, 16, 18,
Initiator: Child process 735612 returned 18.
Initiator[735610]: Forked process with ID 735613.
Initiator[735610]: Waiting for Process [735613].
Total[735613] : Sum = 55
Initiator: Child process 735613 returned 55.
Initiator: Pell: 217
Initiator: Composite: 18
Initiator: total Count: 55
Initiator[735610]: Forked process with ID 735614.
Initiator[735610]: Waiting for Process [735614].
Pell[735614] : Number of terms in Pell series is 15
Pell[735614] : The first 15 numbers of the Pell sequence are:
0, 1, 2, 5, 12, 29, 70, 169, 408, 985, 2378, 5741, 13860, 33461,
80782,
Initiator: Child process 735614 returned 142.
Initiator[735610]: Forked process with ID 735615.
Initiator[735610]: Waiting for Process [735615].
Composite[735615]: First 15 composite numbers are:
4, 6, 8, 9, 10, 12, 14, 15, 16, 18, 20, 21, 22, 24, 25,
Initiator: Child process 735615 returned 25.
Initiator[735610]: Forked process with ID 735616.
Initiator[735610]: Waiting for Process [735616].
Total[735616] : Sum = 120
Initiator: Child process 735616 returned 120.
Initiator: Pell: 142
Initiator: Composite: 25
Initiator: total Count: 120


denver%machine%
```

6. What to Submit

Use the CS370 *Canvas* to submit a single .zip or .tar file that contains:

- All .c and .h files listed below and descriptive comments within,
  - `Initiator.c`
  - `Pell.c`
  - `Composite.c`
  - `Total.c`
- A Makefile that performs both a *make build* as well as a *make clean.* Note that you will have four executables.
- A README.txt file containing a description of each file and any information you feel the grader needs to grade your program, and answers for the 5 questions

For this and all subsequent assignments, you need to ensure that you have submitted a valid .zip/.tar file. After submitting your file, you can download it and examine to make sure it is indeed a valid zip/tar file, by trying to extract it.

Filename Convention: The archive file must be named as: <FirstName>-<LastName>-HW2.<tar/zip>. E.g. if you are John Doe and submitting for assignment 2, then the tar file should be named John-Doe-HW2.tar

7. Grading

The assignments much compile and function correctly on machines in the CSB-120 Lab. Assignments that work on your laptop on your particular flavor of Linux/Mac OS X/Windows, but not on the Lab machines are considered unacceptable.

The grading will be done on a 100 point scale. The points are broken up as follows:

Objective	Points
Correctly performing Tasks 1-8 (10 points each)	80 points
Descriptive comments	5 points
Compilation without warnings	3 points
Questions in the README file	7 points
Providing a working Makefile	5 points

A readme file should be created by you it should have the following:

- Output as shown above in Example outputs.
- Answers to the questions provided below

Questions: (To be answered in README file. Each question worth 1 point unless mentioned)

1. What is the return value for the fork() in a child process?
2. Give a reason for the fork() to fail?
3. How many of the least significant bits of the status does WEXITSTATUS return?
4. Which header file has to be included to use the WEXITSTATUS
5. In this program, are we doing a sequential processing or a concurrent processing with respect to the child processes? Sequential processing is when only one of the child processes is running at

one time, and concurrent processing is when more than one child process can be running at the same time.
6. Is it possible to have any anomalies in the output from child process and the output from parent process. Provide a reason for that possible situation. (2 points)

8. Late Policy

Click here for the class policy on submitting late assignments.

Notes:

1. You are required to work alone on this assignment.
2. Late Policy: There is a late penalty of 20%. The late period is 24 hours.
3. Note that although WEXITSTATUS(status) is primarily intended for returning the status to the parent, here we are exploiting this capability to transmit the result type from the child programs to the parent program.

Revisions: Any revisions in the assignment will be noted below.

9/12/2021:The input.txt used in the example actually contains the string:-

10

15

Also the output has been updated