HW2: Programming Assignment v9/13/24 2PM

Working With Parent and Child Processes

Write a program that reads strings from a file, creates three child processes to do the required computations and obtains status information from them. It uses fork(), exec(), wait() and WEXITSTATUS(status) system calls.

Due Date: 9/20/2024 11 PM

Extended Due Date with 10% per day penalty until 9/22/2024 11 PM

1. Description of assignment

Write a C program called `Generator` that reads strings from a file, whose name be provided as a command line argument to the program. Generator forks three child processes for each line, that will run programs ***`Fibonacci`*** `Numbers`, `Perrin Sequences, and Composite Numbers`.

You will be creating four programs: `Generator.c, Fibonacci.c, Perrin.c and Composite.c` described below.

`Generator.c` takes one mandatory argument which is the name of the .txt file. `Generator.c` will read all the lines from the given .txt file and send each line value to the first child process and then the other child processes will use the result obtained from the previous child Process as their input argument.

a. The `Generator` is responsible for executing the fork() functions to create the child processes.
b. Each created child Process runs the exec() function to run the program needed (`Fibonacci Numbers, Perrin Sequences or Composite Numbers`).The `Generator` should provide the required argument for the first child process to complete its execution. The second child process should use the result returned by the first child process as an argument and the third child process will use the result obtained from the second child process as input.
c. The wait() function is used to wait for the child processes to complete its execution. The Macros WIFEXITED(status) and WEXITSTATUS(status) are used to obtain the result(as an eight-bit integer) from the three child processes.

The `Generator` saves the status(result) obtained from each child process. After the completion of the execution of all the processes the `Generator` will print the outputs from each child processes executed.

`Fibonacci`.c, `Perrin`.c, `Composite`.c:

Each of these programs receives one line as an argument. The programs are used to print the sum of the respective series of numbers using the argument provided and return a result.

Note: 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.

`Generator.c` should indicate the process ID of the child process it created and the child processes (`Fibonacci, Perrin, Composite`) should indicate their own process ids. The example output that is shown below depicts the expected format of the output and must be strictly adhered to.

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

2. Input and Output

For example, the "input.*txt*" file could contain the strings "5" and "10" on two separate lines which would mean to run all three child processes with the first input to Fibonacci.c being 5 and then 10.
Use fopen() function to read the string from the file.

Notes:

- You can assume that the input numbers to be between 1(inclusive) and 20 (inclusive).
- Note that the end of a line is indicated differently in text files for Windows and Linux system. So please test your program in a linux environment e.g. on the CS Machines.

3. Task Requirements

1. The `Generator` 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 first child process. The first child process must accept the string as an argument.

2. The `Generator` 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 `Generator` 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 `Fibonacci/Perrin/Composite` program into the newly created process. This exec() function should also pass the value to the `Fibonacci/Perrin/Composite` program. For this assignment, it is recommended that you use the execlp() function. The "man" page for exec (search for "man exec" on google or if you are on linux you can type that into a shell) gives details on the usage of the exec() family of functions.

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

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

   a. `Fibonacci` should print the sum of the first n Fibonacci Numbers and the nth Fibonacci number (assume n is the argument provided).
   b. `Perrin` should print the sum of the first n numbers of the Perrin Sequences and the nth Fibonacci number (assume n is the argument provided).
   c. `Composite` should print the sum of the first n Composite Numbers (assume n is the argument provided) and the nth Composite number.
   d. `Fibonacci` should return (assume n is the argument provided):
      n if the Sum of the first n Fibonacci numbers is greater than 50.
      Otherwise, Sum of the first n Fibonacci numbers.
   e. `Perrin` series should return:
      n if the Sum of the first n Perrin numbers is greater than 100.
      Otherwise, Sum of the first n Perrin numbers.
   f. Composite should return:
      n if the Sum of the first n Composite numbers is greater than 200.
      Otherwise Sum of the first n Composite numbers.

7. `Fibonacci/Perrin/Composite/` program should return the respective results after executing. Each result is received and stored by the `Generator`. The stored value is used as the argument for the next child process that is forked. You can use the WEXITSTATUS() macro to determine the exit status code (see man 2 wait).

   Note: Please be careful of the data types of the input arguments and the returned results from each child process.

8. Parent-specific processing in the Generator should ensure that the Generator will wait() for each instance of the child-specific processing to complete. Once all the processes are complete output the value returned as mentioned in 6.d, e, f to the terminal.

IMPORTANT: Your program **must** fork(), exec(), wait() the programs in this order for each input provided: `Fibonacci-Perrin-Composite.`

4. Example Outputs

1. This is the output when analyzing the file input.txt which contains the strings on two separate lines: -
5
10
(Note: your process IDs will most likely be different)

```
===========================================
Generator Process: Processing line "5"
===========================================
Waiting for the Child Process: (PID: 254516)
Fibonacci: Sum of first 5 Fibonacci numbers is 7
Fibonacci: The nth Fibonacci number is 5
./Fibonacci Process finished (PID: 254516). Returned: 7

Waiting for the Child Process: (PID: 254517)
Perrin: Sum of first 7 Perrin numbers is 20
Perrin: The nth Perrin number is 5
./Perrin Process finished (PID: 254517). Returned: 20

Waiting for the Child Process: (PID: 254518)
Composite: Sum of first 20 Composite numbers is 367
Composite: The nth Composite number is 32
./Composite Process finished (PID: 254518). Returned: 20

===========================================
Generator Process: Processing line "10"
===========================================
Waiting for the Child Process: (PID: 254519)
Fibonacci: Sum of first 10 Fibonacci numbers is 88
Fibonacci: The nth Fibonacci number is 55
./Fibonacci Process finished (PID: 254519). Returned: 10

Waiting for the Child Process: (PID: 254520)
Perrin: Sum of first 10 Perrin numbers is 49
Perrin: The nth Perrin number is 12
./Perrin Process finished (PID: 254520). Returned: 49

Waiting for the Child Process: (PID: 254521)
Composite: Sum of first 49 Composite numbers is 1846
Composite: The nth Composite number is 69
./Composite Process finished (PID: 254521). Returned: 49
```

5. 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,
    - `Generator.c`
    - `Fibonacci.c`
    - `Perrin.c`
    - `Composite.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.

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.< zip>. E.g. if you are John Doe and submitting for assignment 2, then the zip file should be named John-Doe-HW2.zip

6. Grading

The assignments must compile and function correctly on machines in the CSB-120 Lab. Assignments that work on your laptop on your 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	5 points
Providing a working Make file	10 points

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

- Output as shown above in Example outputs.

7. 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 10%/day for a maximum of two days.
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 from the child programs to the parent program.

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

9/11/24: Output revised.

9/13/24: We are not including any questions; thus no answers need to be included. That sentence has been removed from the README.txt requirements.