HW2: Programming Assignment

Working With Parent and Child Processes v.9.18.25 9:34 AM

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: Oct 2, 2025

Extended Due Date with 10% per day penalty until Oct 2, 2025.

1. Description of assignment

Write a C program called **`Generator.c`** that reads numbers from a file, whose name will be provided as a command line argument. Generator forks three child processes for each line, which will run programs: **`OddEven`**, **`PerfectSquare`**, and **`Factorial`**.

You will create four programs:
`Generator.c`
`OddEven.c`
`PerfectSquare.c`
`Factorial.c`

`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 (OddEven, PerfectSquare, Factorial).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.

`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 (**`OddEven`**, **`PerfectSquare`**, or **`Factorial`**). 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 their 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 process executed.

`OddEven.c, PerfectSquare.c, Factorial.c`:

Each of these programs receives one line as an argument. The programs are used to perform the respective computation using the argument provided and return a result.

`OddEven.c:` Prints whether the number is odd or even, then returns the number itself.

`PerfectSquare.c:` Prints whether the number is a perfect square, then returns the number itself.

`Factorial.c:` Prints the factorial of the number, and returns the factorial value if it is ≤ 255; otherwise returns n % 255.

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 (**`OddEven, PerfectSquare, Factorial`**) 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 **`OddEven.c, PerfectSquare.c`**, and **`Factorial.c`** programs, and then write the code to manage their execution using the Generator program.

2. Input and Output

For example, the "input.*txt*" file could contain the strings "5" and "8" on two separate lines which would mean to run all three child processes with the first input to **`OddEven.c`** being 5 and then 8.
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 numbers from the .txt file, the name of which will be passed as an argument to it. Then send each number, one at a time, to the first child process. The first child process must accept the number 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 **`OddEven / PerfectSquare / Factorial`** program into the newly created process. This exec() function should also pass the value to the **`OddEven / PerfectSquare / Factorial`** program. For this assignment, it is recommended that you use the execlp() function. The "man" page for exec gives details on the usage of the exec() family of functions.
5. When the **`OddEven / PerfectSquare / Factorial`** program is executing, it prints out its process ID; this should match the one returned by the fork() function in step 3.
6. The **`OddEven / PerfectSquare / Factorial`** program then prints the respective outputs and returns a result as follows:
   a. **`OddEven.c`** should print whether the number is odd or even, and return the number itself.
   b. **`PerfectSquare.c`** should print whether the number is a perfect square, and return the number itself.
   c. **`Factorial.c`** should print the factorial of the number. If the factorial ≤ 255, return the factorial value. Otherwise, return n % 255 (% is modulo operation, i.e. remainder after an integer division).

7. Each program should return its result 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 values returned as mentioned in 6.a, 6.b, 6.c to the terminal.

   IMPORTANT: Your program must fork(), exec(), wait() the programs in this order for each input provided:
   `OddEven → PerfectSquare → Factorial`

   Notes:
   a. When you are making makefile, make sure "`make all`" or "`make`" command is not creating any zip file. You can use "`make zip`" or any other command to make zip file.

   b. Check the numbers in the input file and make sure those are between 1 and 20.

4. Example Outputs

This is the output when analyzing the file input.txt which contains the strings on two separate lines:
5
8
(Note: your process IDs will most likely be different)

```
===========================================
Generator Process: Processing line "5"
===========================================
Waiting for the Child Process: (PID: 254516)
OddEven: 5 is Odd
./OddEven Process finished (PID: 254516). Returned: 5

Waiting for the Child Process: (PID: 254517)
PerfectSquare: 5 is Not a Perfect Square
./PerfectSquare Process finished (PID: 254517). Returned: 5

Waiting for the Child Process: (PID: 254518)
Factorial: 5! = 120
./Factorial Process finished (PID: 254518). Returned: 120
===========================================
Generator Process: Processing line "8"
===========================================
Waiting for the Child Process: (PID: 254519)
OddEven: 8 is Even
./OddEven Process finished (PID: 254519). Returned: 8

Waiting for the Child Process: (PID: 254520)
PerfectSquare: 8 is Not a Perfect Square
./PerfectSquare Process finished (PID: 254520). Returned: 8

Waiting for the Child Process: (PID: 254521)
Factorial: 8! = 40320
./Factorial Process finished (PID: 254521). Returned: 30
```

Notice for Factorial:
For 5, factorial is 120 → small enough to return as exit status.
For 8, factorial is 40320, which is too large for exit status (8-bit only), so the rule applies: it returns n % 255 (here, 8).

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, with descriptive comments within:
  - `Generator.c`
  - `OddEven.c`
  - `PerfectSquare.c`
  - `Factorial.c`
- A Makefile that performs both a make build as well as a make clean. Note that you will have four executables.

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 it to make sure it is indeed a valid archive file, by trying to extract it.

Filename Convention: The archive file must be named as:
<FirstName>-<LastName>-HW2.<zip>

Example: 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
Proper submission with required files, compilation and program running	10
Expected program structure with parent/child processes	30
Correct reading of input file	15
Correct OddEven, PerfectSquare, Factorial implementation	45

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.