HW2: Programming Assignment **v4 9.12.2019.11:00PM**

Working with Parent and Child processes

The objective of this assignment is to write and test a program with fork(), exec(), wait() and return values of child processes. You will be a decoding an image in this assignment.

Due Date: Thursday, September 26, 2019, 11:00 pm

Extended Due Date with 20% penalty: Friday, September 27, 2019, 11:00 pm

1. Purpose

Write a C program called `Starter` that reads a list of numbers from a file, whose name may be provided as a command line argument or the default file *coded_image_1.txt* is used. `Starter` forks three child processes for each number, that will run programs `Red`, `Green`, and `Blue` which will extract red, green, and blue values respectively from each number. Each of them returns the decoded value, which will be assembled into an image.

2. Description of assignment

You will be creating four programs: `Starter.c`, `Red.c`, `Green.c`, and `Blue.c`. Two coded image files, *coded_image_1.txt* and *coded_image_2.txt*, are provided to test your program.

`Starter.c`: `Starter.c` takes an optional argument that is the name of the .txt file which contains a list of numbers. If no argument is provided `Starter` should work with the default file *coded_image_1.txt*. Let us assume that an argument is passed and the name of the .txt file is *coded_image_2.txt*. The `Starter` will read the numbers from *coded_image_2.txt*, save them in an array, then send each element of the array to the child processes for further processing.

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

The Starter then saves the status. After all the processes are complete the Starter will assemble the saved status values into a .ppm image (A code snippet is provided below).

`Red.c, Green.c, Blue.c`: Each of these programs receives *one* number (the coded value) as an argument, which represents the color value of a pixel. They extract the component Red, Green, and Blue color values and return the color value as status. The extracted value is obtained by performing operations given below:

Program	Extraction Operation
Red	(CODED_VALUE >>16) & 0xFF
Green	(CODED_VALUE >> 8) & 0xFF
Blue	(CODED_VALUE) & 0xFF

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 `Starter` should indicate the process ID of the child it forked, and the `Red`, `Green` and `Blue` 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.

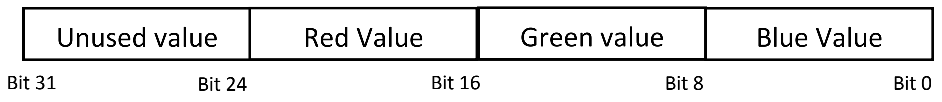
A good starting point is to implement the functionality for the `Red.c`, `Green.c`, and `Blue.c` programs, and then write the code to manage its execution using the Starter program.

3. Input and Output

The "*coded_image_X.txt*" files contain a 76 lines of space-separated integers in the form:

coded_pixel0001 coded_pixel0002 coded_pixel0003 … … coded_pixel0076
coded_pixel0077 coded_pixel0078 coded_pixel0079 … … coded_pixel0152
...
…
coded_pixel5700 coded_pixel5701 coded_pixel5702 … … coded_pixel5776

Each coded value is in the format:



Hence extracting the 8-bit color value uses the right shift operator. For example, coded pixel 151339 in 32-bit binary is {00000000 00000010 01001111 00101011} and thus the red value is 2, green value is 79 and the blue value is 43.

Use fscanf() function in a loop, to read the values from the file.

The dimensions of the coded image are 76x76x3. Each coded_pixel contains the red, green, and blue values, in the appropriate positions and combined into a single integer.

For a file *coded_image_X.txt,* the name of the generated image file should be ***coded_image_X_output.ppm.*** You can assume the name of the file passed will not be longer than 18 characters including the .txt extension.

The code needed to get the name of output file from the name of input file is as shown below:

```
char outputFileName[30]= "";
char extension[12] = "_output.ppm";
strcpy(outputFileName, filename);
char* pos = strchr(outputFileName, '.');
*pos = '\0';
strcat(outputFileName, extension);
```

The easiest way to create an image file using a C program is to generate a .ppm file, which can be done using the code below.

```
FILE *fout = fopen(outputFileName, "wb");
fprintf(fout, "P6\n%i %i 255\n", IMAGE_HEIGHT, IMAGE_WIDTH);
for (int i = 0; i < IMAGE_HEIGHT*IMAGE_WIDTH; i++)
{
      fputc(RED_VALUE[i], fout);   // 0 .. 255
      fputc(GREEN_VALUE[i], fout); // 0 .. 255
      fputc(BLUE_VALUE[i], fout);  // 0 .. 255
}
fclose(fout);
```

4. Task Requirements

1. The `Starter` must read the numbers from the .txt file, the name of which might be passed as an argument to it. If no argument is passed, Starter works with *coded_image_1.txt*. Then save the numbers in an array and then send each number, one at a time, to the child processes. Each of the other three programs must accept the string as an argument.

2. The `Starter` should spawn 3 processes using the fork() command for each value in the array 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() command, the `Starter` will print out the process ID of the process that it created. This can be retrieved by checking the return value of the fork() command.

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

5. When the `Red`/`Green`/`Blue` program is executing, it prints out its process ID; this should match the one returned by the fork() command in step 3.

6. The `Red`/`Green`/`Blue` program then extracts the color value and prints the result.

7. `Red`/`Green`/`Blue` program should return the result. Each status received by the `Starter` should be used to determine the values of the given pixel. You can use the WEXITSTATUS() macro to determine the exit status code (see man 2 wait).

8. Parent-specific processing in the `Starter` should ensure that the `Starter` will wait() for each instance of the child-specific processing to complete. Once all the processes are complete use the code snippet above to write the values in to the .ppm file, then close the file.

5. Example Outputs

1. This is the output when decoding the final pixel in *coded_image_1.txt* (Note: your process IDs may be different)

```
machine% ./Starter
.
.
.
Starter: Forked process with ID 16220.
Starter: Waiting for process [16220].
Red[16220]: Received coded value 151339
Red[16220]: Decoded into 2
Starter: Child process 16220 returned 2.
Starter: Forked process with ID 16221.
Starter: Waiting for process [16221].
Green[16221]: Received coded value 151339
Green[16221]: Decoded into 79
Starter: Child process 16221 returned 79.
Starter: Forked process with ID 16222.
Starter: Waiting for process [16222].
Blue[16222]: Received coded value 151339
Blue[16222]: Decoded into 43
Starter: Child process 16222 returned 43.
```

```
Starter: coded_image_1_output.ppm file written and closed.
```

This should be the image generated using coded_image_1_output.ppm

2. This is the output when decoding the final pixel in coded_image_2.txt (Note: your process IDs may be different)

```
machine% ./Starter coded_image_2.txt
.
.
.
Starter: Forked process with ID 98269.
Starter: Waiting for process [98269].
Red[98269]: Received coded value 11711171
Red[98269]: Decoded into 178
Starter: Child process 98269 returned 178.
Starter: Forked process with ID 98270.
Starter: Waiting for process [98270].
Green[98270]: Received coded value 11711171
Green[98270]: Decoded into 178
Starter: Child process 98270 returned 178.
Starter: Forked process with ID 98271.
Starter: Waiting for process [98271].
Blue[98271]: Received coded value 11711171
Blue[98271]: Decoded into 195
Starter: Child process 98271 returned 195.
Starter: coded_image_2_output.ppm file written and closed.
```

The image generated as coded_image_2_output.ppm is not provided in this document.
Hint: The image shows a building.

Information about.PPM files: PPM stands for "portable pixel map", this is designed to be a simple image format that works everywhere. Additional information can be found here: http://netpbm.sourceforge.net/doc/ppm.html

The first line of our .ppm file is just the string "P6", indicating a .ppm file in byte format.
The second line is three numbers: the image width, the image height, and the maximum color value. In our case, this will be "76 76 255"

The rest of the file is 76 lines, each containing 76 byte-triplets. The first byte represents the red value, the second represents the green value, and the third represents the blue value. Each byte has a value between 0-255.

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,
  - `Starter.c`
  - `Red.c`
  - `Green.c`
  - `Blue.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 also 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
Questions in the README file	5 points
Providing a working Makefile	5 points

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

1. How many of the least significant bits of the status does WEXITSTATUS return?
2. Which header file has to be included to use the WEXITSTATUS?
3. What is the return value for the fork() in a child process?
4. Give a reason for the fork() to fail?
5. In the program written by you, 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.

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. You may assume that the text is limited to a single line, in order to keep the assignment simple.
4. 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.

Additional files: In addition to this document, additional files coded_image_1.txt and coded_image_2.txt are available in Canvas for this assignment.

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