CS 5463: Fundamentals of Systems – Programming Assignment 3 (100 points + 20 extra points)

Description: The main objectives of this project is to practice **process management** with system calls and **inter-process communication** with files. It is an extension of Assignment 1.

In this assignment, you need to design a program (says, `file-handler`) that can simultaneously handle multiple input text files by creating **multiple process** (where each process is responsible to handle one input file). Here, for each process, it will need to find out the number of words in its text file. Specifically, the process will open the text file and parse through its content. It will count the total number words in the file. The typical format to run the program with multiple input files is as follows:

```
>./file-handler file-1.txt, file-2.txt, . . ., file-n.txt
```

Detailed Requirements: First, the program needs to determine the number (i.e., n) of input files and their names using the parameters of `main()` function (i.e., the `argc` and `argv` parameters). For each input file `file-k.txt`, the main program should create a child process k using the system call `fork()` to handle the file, which will count the total number of words in the file. At the end, the k's child process should write the total number of words (as an integer) to an output file named as output-count-k.txt.

The main program/process should wait the completion of all child processes using `wait()` or `waitpid()`. Then, the main process will read the output files of all its child processes one by one to find out the number of words in each file, and accumulate the number to be an overall-total. The main program will print out the word count for each file, and then the overall total number of words at the end.

You may use the functions/codes in your assignment 1 in this assignment for each process to handle it input file. Or you can write different functions. As an example, suppose that there are 3 input files, the program will run as follows:

```
>./file-handler file-1.txt, file-2.txt, file-3.txt

The main process will create 3 child processes:
Process pid-1 was created!
Process pid-2 was created!
Process pid-3 was created!

The input file-1.txt has xxx words
The input file-2.txt has yyy words
The input file-3.txt has zzz words

The overall total number of words is xyz.
```

Here, xyz = xxx + yyy + zzz, which are all integers.

For word counting, you could simply use the space character as the delimiter. Anything that is not separated by the space will be counted as a single word. For instance, the example "`The first program is a Hello-world`." will be reported as 6 words (where Hello-world is counted as a single word).

Extra Points: 20 points
Extra points will be given if the main process communicate with its child processes with **pipes** (either regular or named pipes).

Programming Environment and Submission:

- You should debug and test your program on one of the Linux machines (`fox01` to `fox04.cs.utsa.edu`);
- Your program needs to have proper comments (e.g., explanations for functions and variables etc.). It will be graded based on both functional correctness and clarity of the necessary comments.
- You should prepare one `Makefile` for this assignment to make it easy to compile your program; See the related document for how to create and use `Makefile`;
- You should also have a `Readme.txt`. In this text file, you need to report: a) the status of your program (completed or not; partial credit will be given even the program is not completed); b). how the main program communicate with its child processes; c) instructions on how to compile and run your codes with `Makefile`; d) what help (if any) you received from your classmates or others; and e) comments and suggestions to improve this assignment;
- You should create a tar-ball for all the above files with the name of `assign3_abc123.tar.gz` and upload the file on Blackboard, where abc123 should be your UTSA ID. See below on how to create a tar-ball file.

Since your program may involve **multiple files (.c, .h, and the executable),** you will need to bundle the enter folder into one file, which will then be uploaded to Blackboard. We will use the Linux tool **tar** to bundle all these files and **gzip** to compress the file. The resulting file is a **.tar.gz** file, also known as a **tarball**. These two separate tools are often used together and a lot of source code is distributed as a **.tar.gz** file.

To create this tarball, you will need to navigate to the **parent** directory of your project folder. The command to create this is:

```
tar -zcvf assignX-abc123.tar.gz [name of lab/project folder]
```

The options **z** zips the file, **c** creates the archive, **v** stands for verbose and shows what is being added to the archive, and **f** indicates the final file name. For consistency, the name of the tarball and project folder should be the same.

```
tar -zcvf assignX-abc123.tar.gz assignX-abc123
```

The resulting file will be **`assignX-abc123.tar.gz`**, which is the file you will submit to Blackboard.

Unpacking the Tarball

To unpack the tarball, you will need to navigate to the directory in which is exists. From there, the Linux command is:

```
tar -xzvf [name of tarball].tar.gz
```

The options x stands for extract and z means to unzip. This command will extract the files to the folder name that was originally bundled together. So, you need to make sure the original folder name when you were creating the tarball is correct.

So, with the assign1 example above, the command is:

```
tar -xzvf assign1-abc123.tar.gz
```