Project 2: Word-Count With Fork and Pipes

Due Date: 3/7 by 11:59p

Important Reminder: As per the course Academic Honesty Statement, cheating of any kind will minimally result in receiving an F letter grade for the entire course.

Aims of This Project

The aims of this project are as follows:

- To get familiarity with creating processes under Unix.
- To implement interprocess communication using anonymous pipes.
- To gain experience with creating a multiple process IPC setup depending on dynamic parameters.

Project Specification

Write a program to output the `N` most frequently occurring words from one or more input files. Specifically, submit a `prj2.tar.gz` archive such that unpacking that archive into a directory and typing `make` within that directory builds a `word-count` executable which when invoked as:

```
$ ./word-count N STOP_WORDS FILE1...
```

outputs on standard output the `N` most frequently occurring words in one-or-more files `FILE1...` which are not in the file `STOP_WORDS`.

- A **word** is a maximal sequence of characters for which the standard C library function `isalnum()` returns non-zero or is a single quote `'` (apostrophe `'\''`). So `is'nt` will be treated as a single word.
- Words which differ merely in case are regarded as identical.
- The output should consist of `N` lines with each line containing a lower-cased word followed by a single space character followed by the count of that word across all files `FILE1...`. The lines should be sorted in non-increasing order by count; ties where words have the same count should be broken with the lexicographically greater word **preceeding** the lexicographically smaller word.
- The program should handle files which do not necessarily end with newline.
- If the arguments to `word-count` are in error, then the program should print a suitable error message on standard error and terminate.
- The program should also detect any runtime errors (like memory allocation errors, I/O errors, process errors) and terminate after outputting a suitable error message on standard error.

The program must meet the following implementation restrictions:

- The program may not directly or indirectly use any languages other than C.
- The process started at program invocation must create a separate child processes for handling each of the files `FILE1, FILE2, ..., FILEM`. Child process *i* should return a count of the distinct words in `FILE`*i* to the parent process using anonymous pipes. The child processes should be allowed to execute concurrently, at least partially.

  The parent process must combine the individual results from each file to produce the top `N` most frequently occurring words across all of `FILE1, FILE2, ..., FILEM`.
- The program should not terminate until **all** child processes have terminated. This should be the case even if a child process fails with a runtime error.
- The program may assume that all distinct words in all of `FILE1...` and their counts can fit within memory.
- There should not be any implementation restrictions on the size of entities except those defined by available resources. Hence there should not be any restriction on the size of a word or a line.

Provided Files

The ./files directory contains the following:

Makefile
: This file assumes that the project is contained entirely within a `main.c` main program. It provides the following targets:

  `word-count`
  : This will build the `word-count` program.

  `clean`
  : This will clean out all generated files.

  `submit`
  : This will build the required `prj2.tar.gz` archive.

  Typing simply `make` will build the `word-count` program, typing `make clean` will remove all generated files and typing `make submit` will create a `prj2.tar.gz` compressed archive which can be submitted.

  You will probably use multiple source files for your project; please edit the `Makefile` accordingly. When editing, watch out for tabs (the first character of any command-line **must be a tab character**).

README
: A template README; replace the XXX with your name, B-number and email. You may add any other information you believe is relevant to your project submission.

`main.c`
  A driver with a dummy `main()` function to be filled-in by you.

Hints

You may choose to follow the following hints (they are not by any means required).

Before getting started, it is probably a good idea to review the `fork()`, `wait()` and `pipe()` systems calls as well as I/O redirection. Also decide on how you will dynamically create and track the necessary `M` pipes.

1. Use your solution to `prj2` or the provided solution to create a function which given a data-structure containing the stop-words and the name of a data file outputs a word-count of the words in the data file on standard output.

2. Write a driver program in `main.c`. Make sure it prints appropriate error messages if given incorrect arguments.

3. When given a single data file, invoke the function from (1) on that data file.

4. When given multiple data files `FILE1, ..., FILEM`, invoke the function from (1) on each data file using a separate process. The output from the multiple child processes will be interspersed.

5. Ensure that the invoking process does not terminate until all parent processes have terminated.

6. Set up a single pipe per child process to send its results back to the parent rather than to standard output. Have the parent process simply copy the data from each child pipe's to standard output.

7. Change the parent to only output up to the top `N` words across all `M` data files.

8. Test and review your code until it meets all requirements.

Submission

You will need to submit a compressed archive file `prj2.tar.gz` which contains all the files necessary to build your `word-count` executable. Additionally, this archive **must** contain a `README` file which should minimally contain your name, email, the status of your project and any other information you believe is relevant (it is probably a good idea to mention which data-structure you have chosen for your word-store).

If you are using the suggested project structure, then the provided Makefile provides a `submit` target which will build the compressed archive for you; simply type `make submit`.

Note that it is your responsibility to ensure that your submission is complete so that simply typing `make` builds the `word-count` executable. To test whether your archive is complete, simply unpack it into a empty directory and see if it builds and runs correctly.

Submit your project using the submission link for this project, under **Projects** in Blackboard for this course.