Project 4: Unix Process Control and Pipes

Due: Monday 13 May 2013, 23:59:00 Pacific USA time zone.

Points on this assignment: 150 points with 10 bonus points available.

Work submitted late will be penalized as described in the course syllabus. You must submit your work twice for this and all other homework assignments in this class. Ecampus wants to archive your work through BlackBoard and EECS needs you to submit through TEACH to be graded. If you do not submit your assignment through TEACH, it cannot be graded (and you will be disappointed with your grade). Make sure you submit your work through TEACH. Submit your work for this assignment as a *single bzip file* through TEACH. The same single bzip file should also be submitted through BlackBoard.

Place all of the files you produce for this assignment in a single directory, maybe something called Homework4.

In this assignment you will explore Unix process creation and control and the use of Unix pipes for inter-process communication. This assignment is more complex than the previous assignments. It is probably the most complex homework assignment in the class. You have 2 weeks to complete this assignment. However, I urge you to not delay beginning it.

You are going to want to become very familiar with chapters 24-27 and 44 from TLPI. Section 44.4 is especially good to (re)read. If the text has source code (and I know it does), read that too. Knowing how to read code makes you better at writing code. You can download all the source code from the book as well (the CS311 VM already has the book source code downloaded).

1. **5 points**. When you are ready to submit your files for this assignment, make sure you submit a *single bzip file*. Review homework #1, problem #1 if you need a refresher on how to do this. If your file is not a single bzip file, you cannot receive points on this homework assignment.

2. **145 points.** Write a C program (on `os-class` or the CS311 VM) called `uniqify`. `uniqify`'s job is to read a text file (as `stdin`) and output the unique words in the file, sorted in alphabetic order with the count of the occurrence of each word. The input is from `stdin`, and the output is to `stdout`. I have provided a sample bash shell script (`uniqify.bash`) and 6 sample data files that will allow you to test the output of your program.

Internally, the program should be organized into 3 types of processes (or stages).

1. A single process reads the input parsing the lines into words,
2. A group of process does the sorting, and
3. A single process suppresses duplicate words and writes the output.

You should organize the processes so that every parent process waits for its child processes to terminate (no zombies). Using a signal handler for SIGCHLD is a good idea, as is a SIGPIPE.

You must use the system sort command (`/bin/sort`) with no arguments to do the actual sorting and your program must arrange to start the processes and plumb all the pipes (this means no calls to `popen()`). The number of sorting processes is a command line argument to the program (`-c #`), with the parser distributing the words round robin to the sorting processes (see note below if "round robin" is new to you).

The I/O with the pipes should be done using the stream functions `fgets()` and `fputs()`, with `fdopen()` for attaching to the pipes, and `fclose()` for flushing the streams. After each sorting step is complete, work must still be done to merge and "uniquify" the words prior to printing the list and counts from the suppresser.

In this assignment words are all alphabetic and case insensitive, the parser should convert all alphabetic characters to lower case. All non-alphabetic characters delimit words and are discarded by the parser.

Provide timings based on the size of the input file and the number of sort sub-processes. Ensure that you test it with multiple types and sizes of files, including word lists and free form prose (the 6 sample files should just about cover it). These timings should be plotted in a fashion that makes sense. Use MS Excel or LibreOffice Calc to produce a spreadsheet of your timing results and create a graph showing time on the vertical axis and input size as the horizontal axis. 15 points (of the 140).

Create a slide using wither MS PowerPoint or LibraOffice Draw that shows the parent/child topology for your application and a slide that shows the input/output pipeline topology for you program. Make sure you clearly label each process and edge. If you use more than one process for a stage in the pipeline, note it in your slide. Examples of this can be found in the rev-cat document. 15 points (of the 140).

This is a complex assignment. Break it into pieces. My recommendation is to first solve processing the input, removing non-alpha characters, down-casing all the letters, and splitting a line of words into multiple words. From there, you can work on the fork and exec of the sort sub-processes and get all the pipes in order (this will be more complex than you think). Make sure you can accept a command line argument for the number of sort sub-processes. Once you get all the sort sub-

processes plumbed, start working on recombining the results. Merging the output from the sort sub-process to eliminate duplicates and NOT storing all the data and/or resorting data will take some analysis. You do not want to be doing this at 11:00 pm the night the assignment is due.

One of the joys of programming in C is that you get to write a few low level functions on your own. For example, the string handling functions in C are pretty spartan. Depending on how you want to go, you don't actually *have* to remove/replace non-alpha characters, just use them as delimiters. There are some C functions that can make tokenizing a string a lot easier (Google can be your friend).

For up to **10 points extra credit**, create a Bash shell script (`uniqifyTest.bash`) that will test and validate your code. It should:

1. Build your `uniqify` application (using your `Makefile`).
2. Run the `uniquify.bash` script on each of the input files and produce a base output file.
3. Run your `uniqify` code on each of the input files, varying the number of sort sub-processes (the –c # command line option) from 1 to 9, and comparing the output from your code to the output from the `uniqify.bash` script.
4. Clearly identify when the base output file and your code's output file differs.
5. At a minimum, you need to use the 6 sample input files provided to you.
6. Except for the simple.txt file, the sample input files were found at Project Gutenberg web site. Some of them had the UTF-8 characters purged from the content before providing them to you.

Notes:

1. Round Robin means that you cycle through sort sub-processes sending words from the input. If your input contains "A B C D E F" and you have 2 sort sub-processes, sub-process #1 will get words "A C E" and sub-process #2 will get words "B E F". This ensures that each sub-process will receive approximately the same about of work.
2. You do not (and should not) need to store the output from all the sort sub-processes and resort it before uniqifying it. If you already know the data from the sort sub-processes is in order, you can assemble the merged word lists also in order.
3. The final set of timing runs for your spreadsheet will probably need to be run on `os-class`. You'll need to have multiple (physical) processors to see any meaningful speedup from the parallel sort sub-processes.
4. The sample input files can also be found on `os-class` in (not everyone on `os-class` needs to have a personal copy of these files): `/usr/local/classes/eecs/spring2013/cs311-400/Homework4`
5. You can assume that you are working with ASCII content files (no UTF-8 characters).
6. You do need a Makefile for this assignment.

The point distribution for `uniqify` is:

Task	**Number of points**
Just for you C code compiling	5
Excel/LibraOffice spreadsheet with timing graph (as described above)	15
PowerPoint/LibraOffice topologies (as described above)	15
Makefile	10
Un-ordered/un-counted output (non-unique)	40
Ordered counted output	60
Total	**145 points**

Things to include with the assignment (in a single bzipped file):

1. C source code (.c and .h files) for the solution to the posed problems (all files).
2. A Makefile to build your code.
3. A file describing how you ran your tests.
4. A PowerPoint/LibraOffice file showing the process topologies.
5. An Excel/LibraOffice file showing the timing results.
6. The extra credit script (if you do that portion).

A file describing how you ran your tests, including where you found input files
Please combine all of the above files into a single bzip file prior to submission.