Assignments

Assignment - In progress

Complete the form, then choose the appropriate button at the bottom.

Title	A Multi-process Sorter: Part 1
Due	Nov 1, 2017 11:55 pm
Number of resubmissions allowed	10
Accept Resubmission Until	Nov 2, 2017 12:00 am
Status	Not Started
Grade Scale	Points (max 100.00)

Instructions

CS214 Project 1: Basic Data Sorter - multiproc

Abstract
This is the second in a series of projects that will involve sorting a large amount of data. In this second phase, you will write a multi-process C program to sort a list of records of movies from imdb alphabetically by the data in a given column. You will make use of the concepts learned in lecture including file/directory access, forking processes, etc.

Introduction
File formats for this part of the project are the same as in the first. The CSV file with movie metadata will remain the same. The sorting algorithm will also remain the same. If you properly modularized your code in Project 0, you should be able to reuse almost all of your code.

In this project, you will read in a directory name and walk through the directory structure to find all .csv files. There may be multiple levels of directories that you will need to recurse through. You will then fork child processes to sort each of the files and output the results to a different file. You should NOT use exec for this project. You should write one program that, when copied from the parent to the child process, will continue running. You can use the return value of fork() in a conditional to choose different blocks of code to run within your code. You will want to make sure to prevent zombie and orphan child processes. You will also want to make sure you to not create fork bombs that will bring down machines. In all cases of bad input, you should fail gracefully (e.g. no segfaults).

You will output metadata about your processes to STDOUT. This metadata will show the total number of processes created and the pids of all created processes.

Methodology
a. Parameters
Your code will read in a set of parameters via the command line. Records will be stored in .csv files in the provided directory. As mentioned above, directory structures may have multiple levels and you must find all .csv files. Your code should ignore non .csv files and .csv files that do not have the correct format of the movie_metadata csv (e.g. csv files that have other random data in them).

Remember, the first record (line) is the column headings and should not be sorted as data. Your code must take in a command-line parameter to determine which value type (column) to sort on. If that parameter is not present (?-> throw an error, or default behavior). The first argument to your program will be '-c' to indicate sorting by column and the second will be the column name:

```
./sorter -c food
```

Be sure to check the arguments are there and that they correspond to a listed value type (column heading) in the CSV.

For this phase you'll extend your flags from one to three. The second parameter to your program will be ‘-d’ indicating the directory the program should search for .csv files. This parameter is optional. The default behavior will search the current directory.

```
./sorter -c food -d thisdir/thatdir
```

The third parameter to your program will be ‘-o’ indicating the output directory for the sorted versions of the input file. This parameter is optional. The default behavior will be to output in the same directory as the source file.

```
 ./sorter -c  movie_title -d thisdir -o thatdir
```

b. Operation

Your code will be reading in and traversing the entire directory. In order to run your code to test it, you will need to open each CSV and read it for processing:

```
./sorter -c  movie_title -d thisdir -o thatdir
```

Your code's output will be a series of new CSV files outputted to the file whose name is the name of the CSV file sorted, with "-sorted-<fieldname>" postpended to the name.

e.g: 'movie_metadata.csv' sorted on the field 'movie_title' will result in a file named "movie_metadata-sorted-movie_title.csv".

On each new file in a directory you encounter, you should fork() a child process to do the actual sorting.

On each new directory you encounter, you should fork() a child process to process the directory.

To STDOUT, output the following metadata in the shown order:

```
Initial PID: XXXXX
PIDS of all child processes: AAA,BBB,CCC,DDD,EEE,FFF, etc
Total number of processes: ZZZZZ
```

You may assume the total number of files and directories will not exceed 255.

c. Structure

Your code should use Mergesort to do the actual sorting of records. It is a powerful algorithm with an excellent average case.

Results:
Submit your "sorter.c", "sorter.h" and "mergesort.c" as well as any other source files your header file references.

Document your design, assumptions, difficulties you had and testing procedure. Include any test CSV files you used in your documentation. Be sure to also include a short description of how to use your code. Look at the man pages for suggestions on format and content. Do not neglect your header file. Be sure to describe the contents of it and why you needed them.

Here are some extra credit options for you. You can choose to do either, both, or neither.

Extra Credit (10 points):
Interpret the -c option as a comma separated list of fields. Cascade your sort over the fields. For example, if the command is:

```
./sorter -c food,drink,price
```

The sorter should sort first on the field food, then on drink, then on price. E.g. if there are multiple records where food is “Vegetable”, the order of these records would then be sorted by the drink field. If there are multiple records where food is “Vegetable”, and drink is “Water”, then those records would further be sorted by the drink field.

Document your algorithm and show examples of some runs of your program on test data.

Extra Credit (10 points):
Create a metadata output file in addition to STDOUT. In this file, output the hierarchy of forked process ids and their associated directory/file names. If one views this file, it should be clear which processes forked which processes, and be able to follow the execution of code. Document the format of your file, and the design of the code that produces the output. Consider using the linux utilities like gnuplot to visualize your data. This might require the use of shared memory/pipe constructs to transfer data back and forth between processes.

Submission

Assignment Text

This assignment allows submissions using both the text box below and attached documents. Type your submission in the box below and/or use the Browse button or the "select files" button to include other documents. Save frequently while working.

Source

Attachments

No attachments yet

Select a file from computer Browse... No file selected. or select files from workspace or site

Submit Preview Save Draft Cancel

Don't forget to save or submit!
