CS 452 Program 1
Multi-process External Sort

Overview

The purpose of this programming assignment is to reinforce your understanding of the mechanisms used in process management system calls (creation, execution, suspension and termination), and in fundamental InterProcess Communication (pipes, signals).

Activities

- Write a program that uses communicating multiple processes to perform a common task in parallel.
- This is to be an ***individual*** programming assignment.
- Submit a professional-looking design document, your ***well-documented*** source code, and a sample run / diagnostic log.
    - The design document should discuss your overall approach, describe data structures used, and indicate assumptions/limitations of your program.
- Be prepared to demonstrate your solution.

Programming Assignment (Parallel External Sort)

At this point you should have a good understanding of the relationship between the fundamental system calls and library functions involved in process management on a Linux system. You should also have experience with using processes that communicate via pipes and signals. Use this knowledge to write a program that uses multiple processes combined with IPC to implement a parallel version of the well-known Merge algorithm.

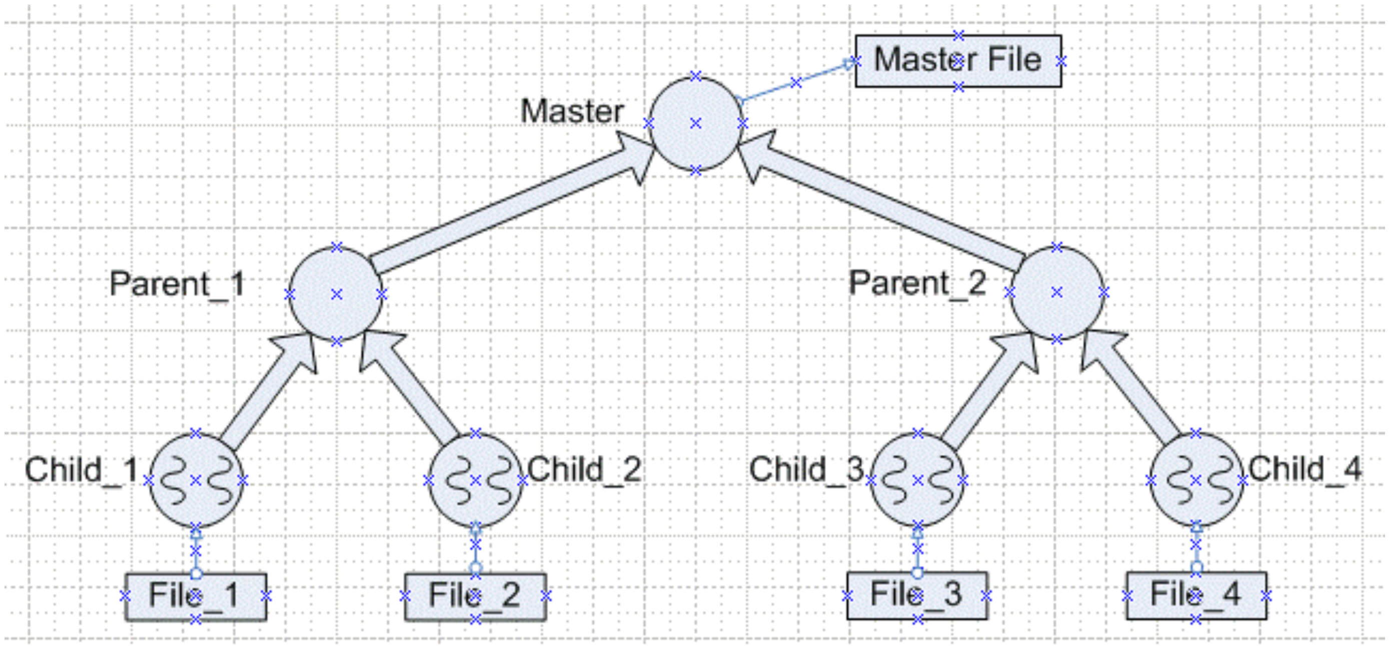
The intended usage is to be able to produce a single sorted file when presented with multiple unsorted files (a process sometimes referred to as an "external sort"). For example, suppose each file is composed of a chronological record of business transactions that took place at a specific branch office; say, a list of all the sales generated at a particular store. There are multiple branch offices, hence multiple files. The Main office of the corporation needs to create a single Master Transaction file that is sorted by customer number.

There is not enough memory to store and sort the entire Master file, so the sort is performed externally. Each process running at a branch office sorts its small, local file by customer number. Then the branch office processes participate in a company-wide hierarchical Merge operation that allows the Main office to write the sorted Master file to disk.

Consider a simplified version of this scenario in which you need to sort and merge four large files of integers. Then your Master program should:

- spawn 2 internal parent processes
- the parent processes should each spawn 2 "leaf-node" child processes
- the child processes should each input and sort a file
- the child processes should then send their sorted values via a pipe to their parent process
- the parent processes should simply merge (not re-sort) the incoming values
- the parent processes should send their sorted values via a pipe to the Master process, which merges them and outputs the final sorted file

The figure below illustrates this activity. Note that only the leaf nodes do actual sorting. All other nodes simply do Merge operations and communicate values.



This represents a very simple case. A more typical use would involve thousands of files, being merge/sorted into a single file. The reason it is called an external sort is that it is not necessary for every element to be in memory at one time (as would occur in most other sorts). Only each individual file is sorted in RAM in the leaf nodes. The sorted files can then simply be piped, one element at a time, into the appropriate output file or pipe.

Guidelines:

- you may use any language you wish, provided you still utilize the standard Linux process management and IPC system calls (or their Windows/Mac equivalents).
- your solution must be scalable (i.e. it should work with a range of processes/files). Your solution must always spawn at least one process per file. You may assume that the number of files to sort will always be a power of 2.
- include informative/diagnostic output to enable tracking the progress of your solution. For example, indicate when you are spawning a new process, reading files, merging values, etc. It is your responsibility to create a decipherable "log" of the activity of your solution. This can be in any format you choose.
- **Note**: a final requirement is that you use (and demonstrate the use of) some form of version control system

in the development of your solution.

Extra Credit

- Incorporate threads into your solution; for example, the leaf-node processes might perform a multi-threaded sort.
- Provide a graphical interface that displays/visualizes the progress of your solution.