Sierra Obi
CSC 412
Fall 2020
Program 4

Report: Processes

Implementation

Overview

The goal of this assignment was to successfully use child processes to distribute the work, and therefore improving the performance. To achieve this, I separated my program into three main components: dispatcher, distributor, and data processing. The dispatcher is the parent function that handles the distribution of the work. In my implementation, the dispatcher keeps track of the complete file catalog and creates child processes to read, sort, and create output. I represented the file catalog as a 2-D array of characters to store each file path. In order to do so, dispatcher calculates the number of files that provides the most even distribution of tasks to each of its child processes. Once that number is calculated, *n* child processes called distributors are created and assigned a set of indices in the catalog to process. These child processes write the number of files found for each of the processes and their respective indices from the file catalog. I represented this data using a array of integers and a 2-D array of integers.

Main Functions

The main functions of the program are dispatcher process, **distributor_build_tasklist**, and **data_processing_build_tasklist**. These functions are the central focus of my implementation because they initiate all of the function calls that process the input. The **dispatcher_process** function takes as input the total number of child processes that need to be created, the total number of files, the catalog of files, and the name of the .c file where the output will be written. The **distributor_build_tasklist**

function is called from the dispatcher. This function takes as input the current index of the child process, the total number of child processes, the file catalog, and the range of indices that was assigned to that distributor. This function reads of the files it was assigned from the catalog, reads them, and reports what files will be processed by the next set of child processes. This information is written to a file. The **data_processing_build_tasklist** is called from the dispatcher once the distributor has completed all of its work. In this data processing function, the data is read in from the files that were created by the distributors. That information is then organized so all of the files that belong to the same child process are put together. From here, I sorted the files in ascending order then wrote their data to a file in the sorted order. The **dispatcher_process** function then read those files created in the data processing phase and wrote them to the final .c file.

Discussion

Error Handling

This assignment was done to completion. I handle errors that relate to the formatting of strings. These primarily are a concern when opening files. I take care of any additional forward slashes and I made sure that the program consumed end line characters. In the bash scripts for version 1 and version 2, I create a folder to store the output from the distributors as the folder was originally one that I created locally.

Improvements

The most dominant time complexity in my program would be $O(n^3)$. For the function **data_processing_fetch_sorter_data**, I needed to use nested for loops in order to read in the data from the file and store it in the proper data structure. In fact, I use many nested for loops throughout my implementation. For smaller data sets, this is trivial

however this can be costly for larger data sets. In the future, I should be careful not to use as many nested loops to improve the performance.

Limitations

Version 1 and Version 2 do not compile property because of a problem in the **data_processing_build_tasklist** function. The sorting function does not work properly and causes a segmentation fault during runtime.