Mathew Varghese (RUID: 170009651)
Shilp Shah (RUID: 170009205)

Assignment 1 Documentation

Design

The main part of the project was figuring out how to traverse the directory using multi-processing. We have 4 files: scannerCSVsorter.c, modCSVsorter.c, modCSVsorter.h, mergesort.c. The last 3 are from the previous assignment and they essentially sort a CSV file based on a columnHeader.

In scannerCSVsorter.c, we traverse the directories. In this file we have a function called traverseDirectory and it takes a pointer to char holding a column name to sort on, the current directory, pointers to characters that hold the start directory and the output directories. The function will iterate over every entry in the start directory. If the entry is a file, the function will fork() and the child process will check if it is a csv of valid format, while the parent keeps iterating through the entries. If it is, the child will call the sorting function in modCSVsorter.c and output the new sorted file into the output directory. If the function comes across an entry that is a directory, it will fork() and the child will call the traverseDirectory function on the entry.

We also keep track of the PIDs in the traverseDirectory method as well. When the method starts, we print the Parent PID. Everytime we fork() and a new child process is made, we store its PID in a pipe that is accessible by all the child processes. At the end of the function, we check if it is the parent process and if it is, we print out the PIDs of all the children that were spawned by reading from the pipe. And for every pid we read we increment by one to keep track of the total number of processes.

Going into a subdirectory and writing to pipe:

```
if(pid==0)//child will go through the sub-directory
{
      pid_t childpid = getpid();
      write(pipefd[1], &childpid, sizeof(pid_t));
      depth += 1;
      snprintf(path, sizeof(path), "%s/%s", startDir, entry->d_name);
      //printf("Directory:%s  AND Path:%s\n", entry->d_name, path);
      traverseDirectory(columnName, path, outputDir);
      return;
}
```

Reading from pipe:
```
      if (depth==0)
      {
            int numProcesses = 0;
            printf("CHILD PIDs: ");
            pid_t pidPlaceHolder;

            while(read(pipefd[0], &pidPlaceHolder, sizeof(pid_t)))
            {
                        printf("%d, ", pidPlaceHolder);
                  numProcesses += 1;
            }
            printf("\nThe number of processed are: %d\n", (numProcesses+1));
      }
```

<u>Issues</u>

We encountered many issues while implementing our solution to this assignment. One of the hardest parts was creating a path of sub-directories while forking and doing recursion. This required a lot of debugging and thought because of how fork( ) and recursion work. Essentially, we used the snprintf( ) function to format the path each time we do a recursive call to enter another sub-directory. The code is given below and that is used within the child process of the fork so it runs for each sub-directory.

```
snprintf(path, sizeof(path), "%s/%s", startDir, entry->d_name);
traverseDirectory(columnName, path, outputDir); //Recursive call
```

Another issue we ran into was adjusting our code from Assignment0 to fit in this assinment. We did not make our code modular enough from Assignment0 so that had to be adjusted to be used in this project. We made our code from Assignment0 into the function:
```
      void sorting(char *headerName, char *fileName, char *filePath, char
      *outputDir, char *outputFileName)
```

This was done so that it can be called multiple times by different child processes. We also added more parameters into the function to be able to handle input and output directory names. Within the sorting function, it does everything from Assignment0, including calling the merge sort function.

Test Cases

Initially, we were mainly testing with directories and sub-directories and getting the correct file path for each one. So we created 3 levels of sub-directories and printed the file path for each one. After debugging our traverseDirectory( ) function, we were able to get correct files paths. Examples given below:

Full path: `/ilab/users/srs304/assign1/`
Created new directories: `/folder1/subFolder/subSubFolder`

File path for folder1: `/ilab/users/srs304/assign1/folder1`
File path for subFolder: `/ilab/users/srs304/assign1/folder1/subFolder`

This is working all variations of files and folders.

Next step was to create CSV files in these subFolders and be able to open and sort them. When we come across a CSV file, we take the file path from the folder and concatenate the fileName to the end of it to get the full path to the CSV file. This was, we were able to open the CSV files and read them for sorting.

Also, we did not get full credit for Assignment0 so we went back and edited the mergesort function so handle the cases we got wrong previously. We used the test cases from Assignment0 that was released on Sakai to make sure our sorting function works properly.

Final Result

In the end, we were able to traverse all the directories and access each file. We created child processes for each sub-directory and each file. Then the child process is responsible to checking if the file is a valid CSV file. Then our sorting function is called for each valid CSV which does the actual sorting itself and makes an output file of an appropriate name in the appropriate output directory.