Adam Jean-Laurent

CSC 412

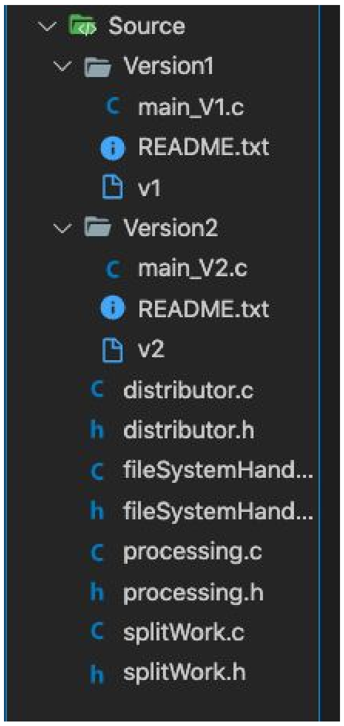
Prof. Hervé

24 October, 2020

Programming Assignment 4 Report

Shared Code:

I chose to use a lot of shared code between versions 1 and 2. The reason being that they for the most part achieve the same goal and call similar functions, just one uses child processes and the other doesn't. I was very consciously writing my functions in version 1 so that their use could be abstracted to version 2, so that I could avoid rewriting the same functionality in slightly different ways. So my folder structure for my source code is as the following, main functions are in their respective version folder, and all shared header and source files live directly in the Source directory.



```
Source
  Version1
    main_V1.c
    README.txt
    v1
  Version2
    main_V2.c
    README.txt
    v2
  distributor.c
  distributor.h
  fileSystemHand...
  fileSystemHand...
  processing.c
  processing.h
  splitWork.c
  splitWork.h
```

Temporary Directory And Generated Files:

Because the inter-process communication in this assignment happened through reading and writing to files, I chose to store all generated communication files in a directory called /temp/. The temp directory is created in my bash script right before calling my compiled binary, and deleted after it finishes running.

There are two types of generated files, the first are to-do lists created by distributors. These files hold the fragment files that a data processing process should process. For example let's say we have a to-do list file called **toDoList_5.txt**.

toDoList_5.txt:

Data-Set-1A3.txt

Data-Set-1/A43.txt

Data-Set-1/H35.txt

This can be interpreted as A3.txt, A43.txt and H35.txt all have an index of 5, and are a part of fragment 5. This list is used by the data processing process 5.

The next type of generated files are fragment files, these are only used in version 2 and represent a re-ordered fragment after the data processing stage. For example let's say we have a fragment file called **Fragment_5.txt**.

Fragment_5.txt:

// -- By - Adam Jean-Laurent

#include “totallyRealHeaderFile.h”

This file is the result of data processing processing 5, and will be reconstructed in the output source file as the 5th fragment. (well 6 because the fragments are 0 indexed but I digress).

Splitting Work:

I talked about this a bit in my Doxygen comments, but in order to make version 2 use processes efficiently I needed to split the number of fragment files each process distributed as evenly as possible. At first I thought I could use simple division and divide NumberOfFiles/NumberOfProcesses, while this works for examples such as 12/4 = each process distributes 3 files, it doesn't work for something like 15/6 = 2.something. So I wrote a function that will split the work as evenly as possible and minimize the difference in number of files each process takes care of.

Other than those personal design decisions, all other code and functionality is essentially what is on the instructions.

Testing:

I have tested both versions 1 and 2 with every dataset provided, passing to my script file extensions that don't end in .c, that do end in .c, directory paths not ending in '/', directory paths ending in'/', directory paths with the full paths, as well as with relative paths. In all of these scenarios the source file was put back together correctly and compiled with no errors. I haven't experienced any crashes while testing. I have included a README in the Script folder to document it's usage as well. Though I didn't fully test this, I believe that the scripts / C programs will work with paths with spaces in them, so hopefully that extra credit works.

If I Had More Time

If I had more time I would have for sure done the Version 3 extra credit. I think that I would have been able to use most if not all the same functionality other than putting some thought into what parameters to pass execXY() when calling it from the dispatcher.