Homework - 1 CSE344
System Programming
Hamza YOĞURTCUOĞLU - 171044086

→ **mainA.c :** I put the **Usage** at first. I seperated command line argument with **getopt().** Then 32 bytes always received from **file1** in an infinite while loop. Each set 2 bytes are converted to ASCII character that is converted complex number. 16 complexes number are wrote in **file2.** But While writing to file2 which is locked (flock). The lock was unlocked after the writing process was finished. After all the lines have been written, processA has written the word **"finished"** and has been terminated. Because processB must understand that processA is terminated.

→ **Test A Examples:** You must execute **2** processA from terminal. The following commands write how you can run processA. Please execute the second processA before the first processA is terminated.

You can execute with Absolute path or relative path for files.

```
./programA -i file1 -o file3 -t 10
./programA -o file3 -i file1 -t 10
./programA -i file2 -o file3 -t 50

./programA -i (AbsolutePath)/file1 -o (AbsolutePath)/file3 -t 10
./programA -o (AbsolutePath)/file3 -i (AbsolutePath)/file1 -t 10
./programA -i (AbsolutePath)/file1 -o (AbsolutePath)/file3 -t 50
```

➜ File1 and File2 Examples: all character must be ASCII character like following example.

```
HamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaH
amzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHam
zaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaH
amzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHa56565656+^+'^+'^+'^mzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamz
HamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamza2342^+'^+'^+'^  +'^+'^+'
HamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaH
amzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamzaHamz
```

➜ File3 Example : When two processA is finished. The process is writing complexes numbers and **"finished"** keyword.

```
97+109i,122+97i,72+97i,109+122i,97+72i,97+109i,122+97i,72+97i,109+122i,97+72i,97+109i,122+97i,72+97i,109+122i,97+72i,97+109i
122+97i,72+97i,109+122i,97+72i,97+109i,122+97i,72+97i,109+122i,97+72i,97+109i,122+97i,72+97i,109+122i,97+72i,97+109i,122+97i
72+97i,109+122i,97+72i,97+109i,122+97i,72+97i,109+122i,97+72i,97+109i,122+97i,72+97i,109+122i,97+72i,97+109i,122+97i,72+97i
109+122i,97+72i,97+109i,122+97i,72+97i,109+122i,97+72i,97+109i,122+97i,72+97i,109+122i,97+72i,97+109i,122+97i,72+97i,109+122i
97+72i,97+109i,122+97i,72+97i,109+122i,97+72i,97+109i,122+97i,72+97i,109+122i,97+72i,97+109i,122+97i,72+97i,109+122i,97+72i
finished
72+97i,109+122i,97+72i,97+109i,122+97i,72+97i,109+122i,97+72i,97+109i,122+97i,72+97i,109+122i,97+72i,97+109i,122+97i,72+97i
109+122i,97+72i,97+109i,122+97i,72+97i,109+122i,97+72i,97+109i,122+97i,72+97i,109+122i,97+72i,97+109i,122+97i,72+97i,109+122i
97+72i,97+109i,122+97i,72+97i,109+122i,97+72i,97+109i,122+97i,72+97i,109+122i,97+72i,97+109i,122+97i,72+97i,109+122i,97+72i
97+109i,122+97i,72+97i,109+122i,97+72i,97+109i,122+97i,72+97i,109+122i,97+72i,97+109i,122+97i,72+97i,109+122i,97+72i,97+109i
122+97i,72+97i,109+122i,97+72i,97+109i,122+97i,72+97i,109+122i,97+72i,97+109i,122+97i,72+97i,109+122i,97+72i,97+109i,122+97i
72+97i,109+122i,97+72i,97+109i,122+97i,72+97i,109+122i,97+72i,97+109i,122+97i,72+97i,109+122i,97+72i,97+109i,122+97i,72+97i
109+122i,97+72i,97+109i,122+97i,72+97i,109+122i,97+72i,97+109i,122+97i,72+97i,109+122i,97+72i,97+109i,122+97i,72+97i,109+122i
97+72i,97+109i,122+97i,72+97i,109+122i,97+72i,97+109i,122+97i,72+97i,109+122i,97+72i,97+109i,122+97i,72+97i,109+122i,97+72i
97+109i,122+97i,72+97i,109+122i,97+72i,97+109i,122+97i,72+97i,109+122i,97+72i,97+109i,122+97i,72+97i,109+122i,97+72i,97+109i
122+97i,72+97i,109+122i,97+72i,97+109i,122+97i,72+97i,109+122i,97+72i,97+109i,122+97i,72+97i,109+122i,97+72i,97+109i,122+97i
72+97i,109+122i,97+72i,97+109i,122+97i,72+97i,109+122i,97+72i,97+109i,122+97i,72+97i,109+122i,97+72i,97+109i,122+97i,72+97i
109+122i,97+72i,97+109i,122+97i,72+97i,109+122i,97+72i,97+109i,122+97i,72+97i,109+122i,97+72i,97+109i,122+97i,72+97i,109+122i
97+72i,97+109i,122+97i,72+97i,109+122i,97+72i,97+109i,122+97i,72+97i,109+122i,97+72i,97+109i,122+97i,72+97i,109+122i,97+72i
finished
```

Each line contains 16 complex numbers. Finished keyword is for processB's which understand processA is terminated. I did something like this because we had to think the processB termination task.

→ mainB.c : I put the **Usage** at first. I seperated command line argument with **getopt().** I lock comman file3 in infinite loop. In order for the lines to not mix, I locked comman file the file3, which is the common file of the processes, so that other processes cannot enter when processB or processA in the file. I had to check processA finished or not. So processB is checking 2 'finished' keyword. If processB sees 2 finished keyword back to back. It will understand all lines processed and processA was finished. It will terminated. If one processA is finished but other processA is working. ProcessB is moving 'finished' to end of file , so that eventually two 'finished' keywords come back to back. First finished processB clear the file3 and writes 'bitti' keyword. Last live processB understand that all processes(two ProcessA and one processB) terminated then clear the file3 and closes the file3 and file4.

After the reading line, the file3 is unlocked, then the dtf operation is performed for the line and written to the common file of the processB's.

I read DFT operation in following website :

https://en.wikipedia.org/wiki/Discrete_Fourier_transform

Then comman file4 is locked for writing 16 complex numbers.

→ Test B Examples: You must execute **2** processB from terminal. The following commands write how you can run processB. Please execute the second processB before the first processB is terminated.

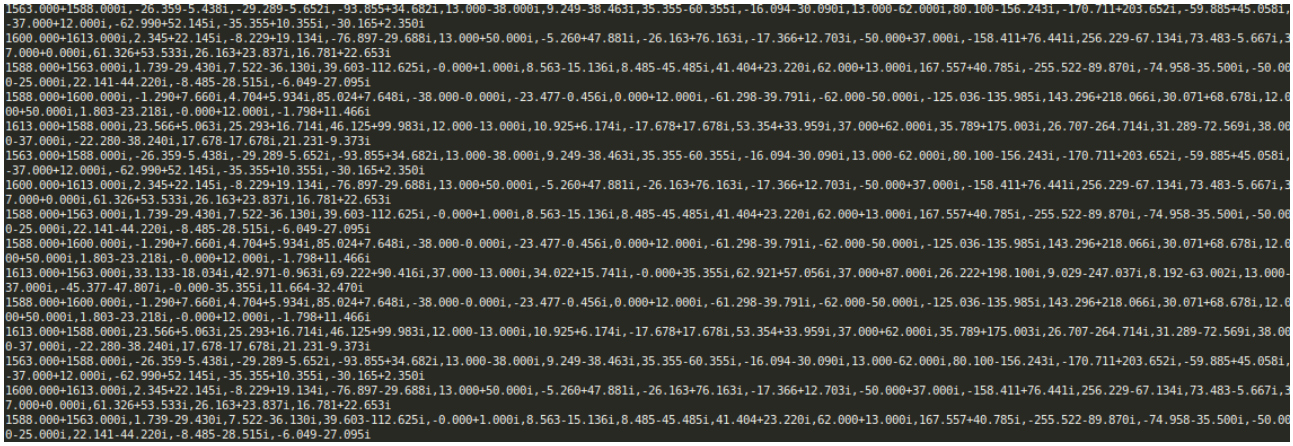
You can execute with Absolute path or relative path for files.

./programB -i file3 -o file4 -t 1
./programB -o file4 -i file3 -t 1
./programB -i file3 -o file4 -t 50

./programB -i (AbsolutePath)/file3 -o (AbsolutePath)/file4 -t 10
./programB -o (AbsolutePath)/file4 -i (AbsolutePath)/file3 -t 10
./programB -i (AbsolutePath)/file3 -o (AbsolutePath)/file4 -t 50

Two instances of program B can be executed (in arbitrary order, before, after or between theprogram A executions), with the same inputPathB and outputpathB arguments.

→ **File4 Example :** processB is reading a line that is performed dft operation. Then 16 complex number is written to file4.



if inputPathA files contain each 32x+y and 32a+b bytes, then the file denoted by outputPathB contains exactly x+a lines, each with 16 complex numbers.

→ **mainC.c :** I put the **Usage** at first. I seperated command line argument with **getopt()**. I found each length of line and number of line. I accepted the number of newlines as the number of lines. Then I calculated the cumulative magnitude of each line. Thus, each line will be cumulative magnitude in the array and a buffer will not be required.

What we know now :

- Number of Lines (int)
- Each length of Line (int array)
- Each Line Cumulative Magnitude (int array)

Then I sorted with in place mergesort algorithm that is not create a new array for sorting operations. Small amount of extra storage space is used for auxiliary variables. That is so smaller than all file. The algorithm is indexing left list and right list in tempArray. Then left and right are swap according to magnitude.

Then I read each line according to cumulative magnitude of lines. I wrote the line temp file. Each line is written after sorted file is written char by char to file4. I didn't use any buffering thing. I just used file. Then temp file is removed. That's all.

→ **Test C Examples:** You must execute 1 processC from terminal. The following commands write how you can run processC. Please execute after the all processA-B are finished.

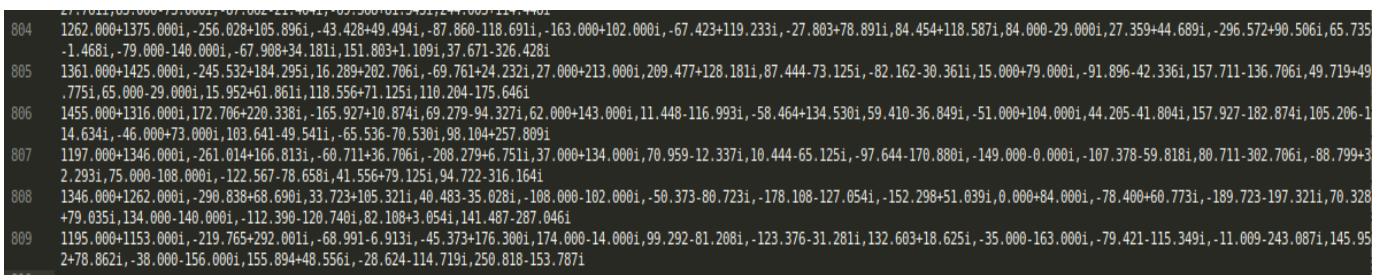
./programC -i file4
./programC -i (AbsolutePath)/file4

→ **Sorted File4 Example :** processC is reading a line that is performed dft operation. Then 16 complex number is written to file4.

…

…

…



Program C can run as a single instance and sort the lines in ascending order of the file denoted by inputPathC using mergesort.

Finally, I left example of file1 and file2.

file1 has 16080 character = 32 * 502 + 16
file2 has 9850 character = 32 * 307 + 26

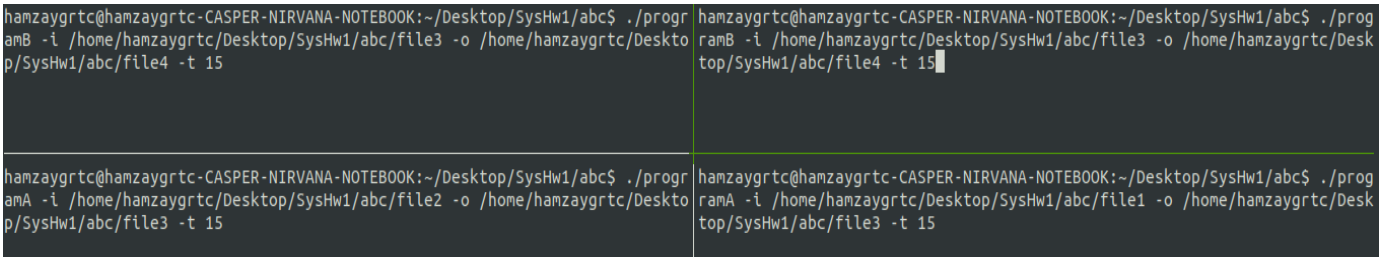
We see file4 that has 307 + 502 = 809 lines.

→ Compiler and Run :

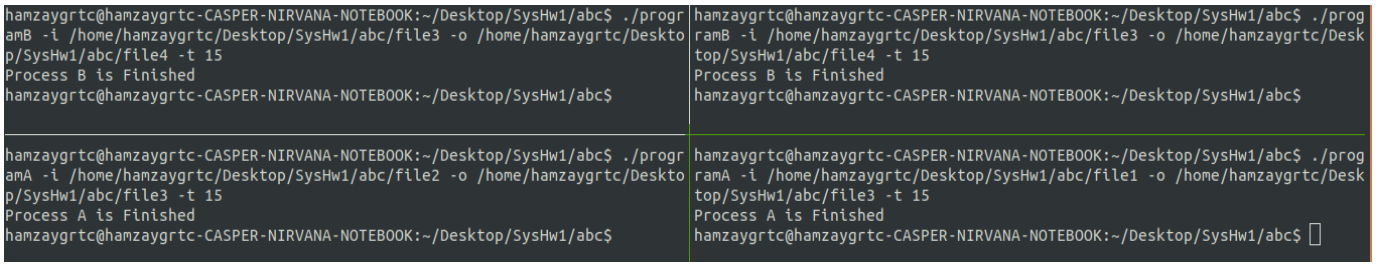
Makefiles Commands:

make #that command compiler all 3 programs
make clean #that command cleans all object files

To run : In this example, I executed processBs then execute processAs.



Then all processes finished.



If you want to see outputs. Check above **outputs.**

→ Note : 1) All operations work fine. It must be run in 4 processes before any process is finished.
2) Two instances of program B can be executed(in arbitrary order, before, after or between the program A executions)

Thanks :)