CSE344 – System Programming
REPORT - HW 1

Furkan ÖZEV
161044036

In this assignment, 2 different program processes are expected to perform reading and writing operations on the same files without any problems.

Program A will be executed twice, so there will be 2 processes running in parallel. Instances of program A will read from distinct input files, and write to a common destination file.

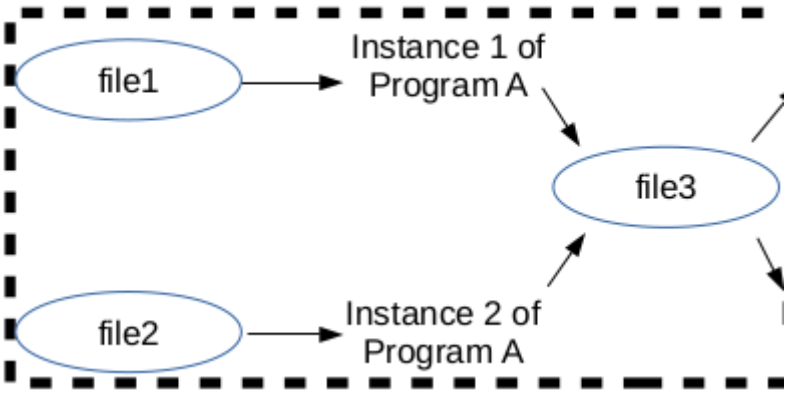
Program A receives the inputs to be used for input file path, output file path and sleep time.

From program A, it is requested to read the ascii characters in the input file as 32 bytes and convert them to 16 complex numbers.

These numbers will then be written in the first available place in the output file. The program will then be put to sleep for a while.

So in practice, we expect two processes reading simultaneously two distinct files, and filling together the same output file with lines containing 16 complex numbers each.

The scheme of Program A is as follows:



DESIGN:

First of all, I used the getopt () function to execute the program if the argument entered was correct and parse the arguments.

If the wrong argument is entered, I write the correct usage with the error message as below:

```
furkan@furkan:~/Desktop/hw1$ ./1 -i inputPat -f outputPathA -t 50
./1: invalid option -- 'f'
This usage is wrong.
Usage: ./programA -i inputPathA -o outputPathA -t time
```

After the arguments were parsed, I checked that the paths were valid:

```
furkan@furkan:~/Desktop/hw1$ ./1 -i ./wrx -o ./outputPathA -t 50
open inputPath: No such file or directory
```

And I checked valid time input:

```
furkan@furkan:~/Desktop/hw1$ ./1 -i inputPathA -o outputPath -t 56
time must be in the range [1.50].
```

I created the input and output files using the open () function. Input file open mode is read only and output file open mode is read and write.

I do all the desired operations in an infinite loop. When the processes are completed, I get out of the loop with "break".

Every time I read 32 bytes from the input file with read() function.
I designed my reading loop so as not to receive any interrupt errors while reading bytes.
And if there is at least 32 bytes to be read, program A will be terminated.

```
while(((bytesread = read(fdin, buf, SIZE)) == -1) && (errno == EINTR));
if (bytesread <= 0)
    break;
if(bytesread != 32) break;
```

After reading 32 bytes from the input file, I converted them to complex number format as desired. With every 32 bytes, there are 16 complex numbers separated by a comma.

Empty row is searched to write these 16 complex numbers to the output file. The file is locked so that the other process does not interfere with the empty row found. Like overwritting.

If the found line is not the intermediate line, that is, the line at the end of the file, Complex numbers are written at the end of the file with the write () function.

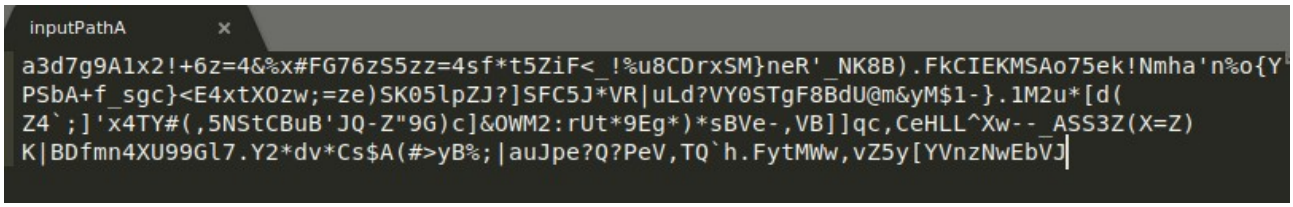
Otherwise;
Since the new bytes will be written, I increased the size of the file using the truncate () function. The complex numbers after the found row are read, the complex numbers we read from the input file are written, then the complex numbers after the empty row are written. A kind of that I moved the bytes after the empty row forward. While doing this, I used the lseek () function for this process since the file cursor should be at the empty row and sometimes at the end of the file.

The file is unlocked after the writing is finished. The file cursor is placed at the beginning of the file. And the program is sleeped using the sleep () function for the time given.

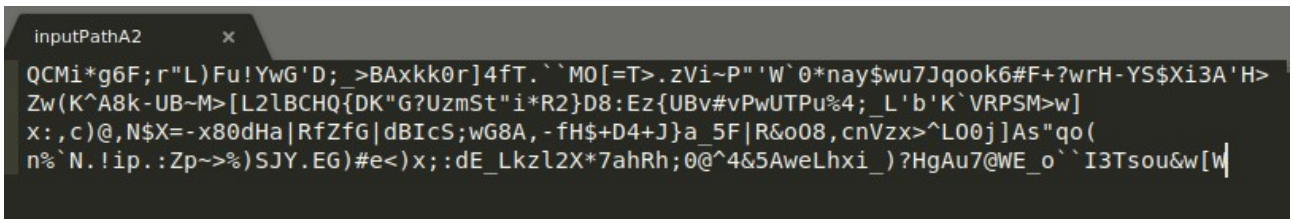
The same processes continue until there are no 32 bytes left in the input file. Files are closed after the processes are finished. Also, the memory allocated with malloc () is released with the free () function.

RUN EXAMPLE:

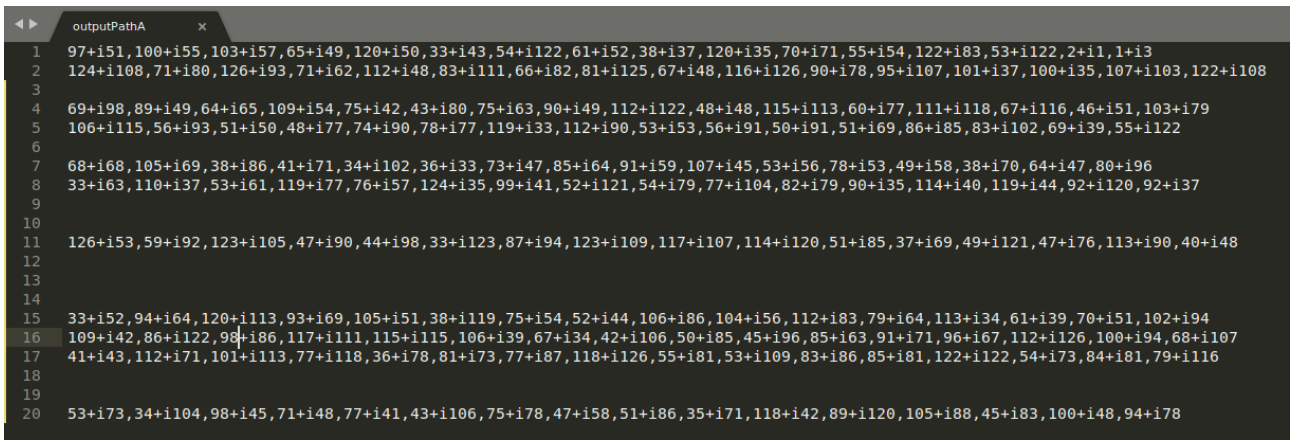
INPUT FILE – A: (320 bytes)



INPUT FILE – A2: (320 bytes)



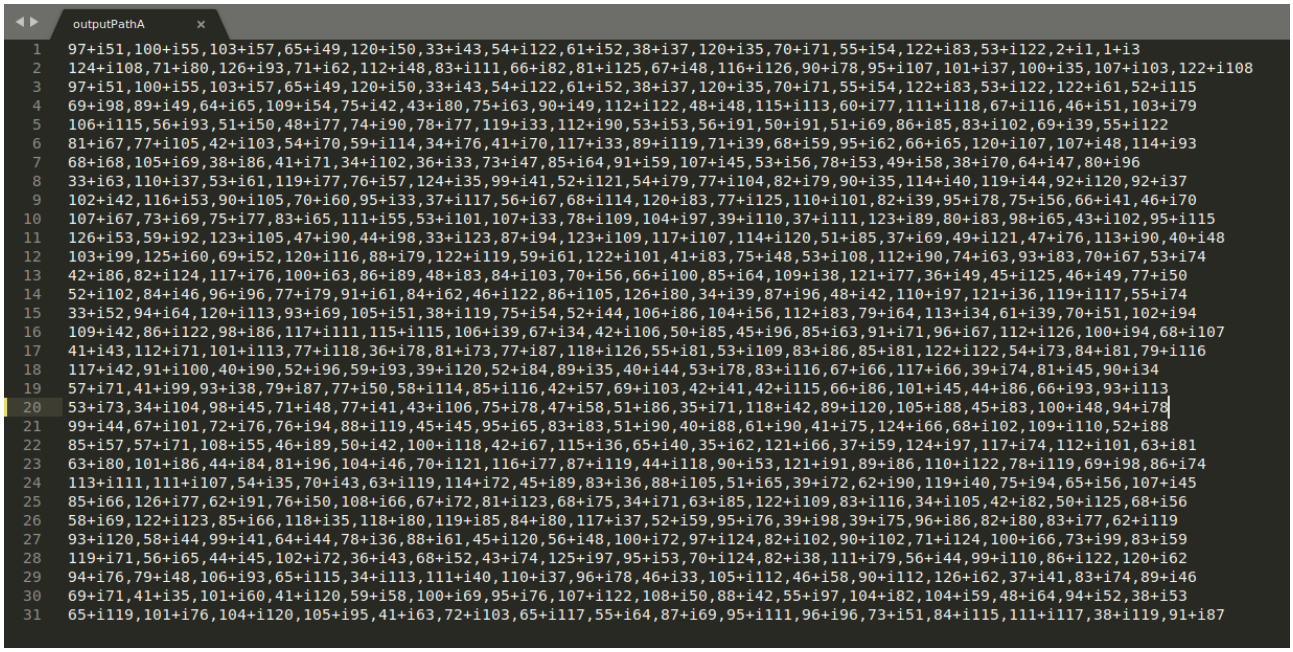
OUTPUT FILE:



COMPILE & RUN:

```
furkan@furkan:~/Desktop/hw1$ gcc -o A A.c
furkan@furkan:~/Desktop/hw1$ ./A -i inputPathA -o outputPathA -t 1 & ./A -i inpu
tPathA2 -o outputPathA -t 5
[1] 3826
[1]+  Done                    ./A -i inputPathA -o outputPathA -t 1
```

FINAL OUTPUT FILE:



NOTICED:

I was only able to make program A.
I didn’t upload program B because it was not working properly and there were errors.