CSE 333 – OPERATING SYSTEMS

PROJECT #3 REPORT

Group Members:

Ömer Faruk Çakı 150117821

Havva Karaçam 150315029

Main

Inside main, we get command line arguments and parse them. After that the first thing we do is getting the line count of the file with our lineCount() function and we set our read threads' threshold to that value.

Read Threads:

We first create our thread initializer with: `pthread_t readThreads[readThreadCount];`

Then using a for loop, we set our threads execution function as _read()

_read() functions works togother with getReadNum() function. getReadNum() function isstart checking from the 0^{th} index until find an available line to read(which is not yet read by any other read threads) and return its line index number to _read() function. In order to prevent the same line number returned to few different read threads we use a mutex lock inside before executing getReadNum() and release it right after we get the line number to read. We don't need to use a mutex to read that line because unique value will be returned for all read threads, so we unlock the mutexRead mutexright after getReadNum() method returns an index.

After we get our line number, thread reads that line from the file by executing getLine() function.

While loop inside the _read ensure that a thread willcontinue executing by processing the next line until all lines finished, so a thread wont die after processing a single line. After we process every line we set readCompleted value to 1.

Upper Threads:

We first create and initialize upper threads inside main. Upper threads run the **upper()** function. Upper function is also attached with getUppercaseIndex() function which returns the index value which can be processed, that means that index is already read by a reader thread and currently residing in records[index].

We use mutexUpper mutex in order to ensure that not more than a upper thread will execute the same value(converting uppercase).

After a upper thread convert the text to uppercase it changes the string store inside records[index].text to new uppercased value and sets records[index].upper = 1 to mark as uppercased.

Again, using the while(1) loop inside the upper() we process all values by upper threads until there is no one left. We use upperCompleted int value to keep trackcount of values processed.

Following line is exits the thread function when there is no non-uppercased threads left

```
if (readCompleted == 1 && upperCompleted >= (limit + 1))
```

Replace Threads:

Replace threads are very similar to upper threads we explained above. Only difference is replace threads uses the replace() function. They also share the same mutexUpper mutex in order to get index number to process from getReplaceIndex() function. Because replace threads and uppercase threads shouldn’t work on the same value at the same time.

Write Threads:

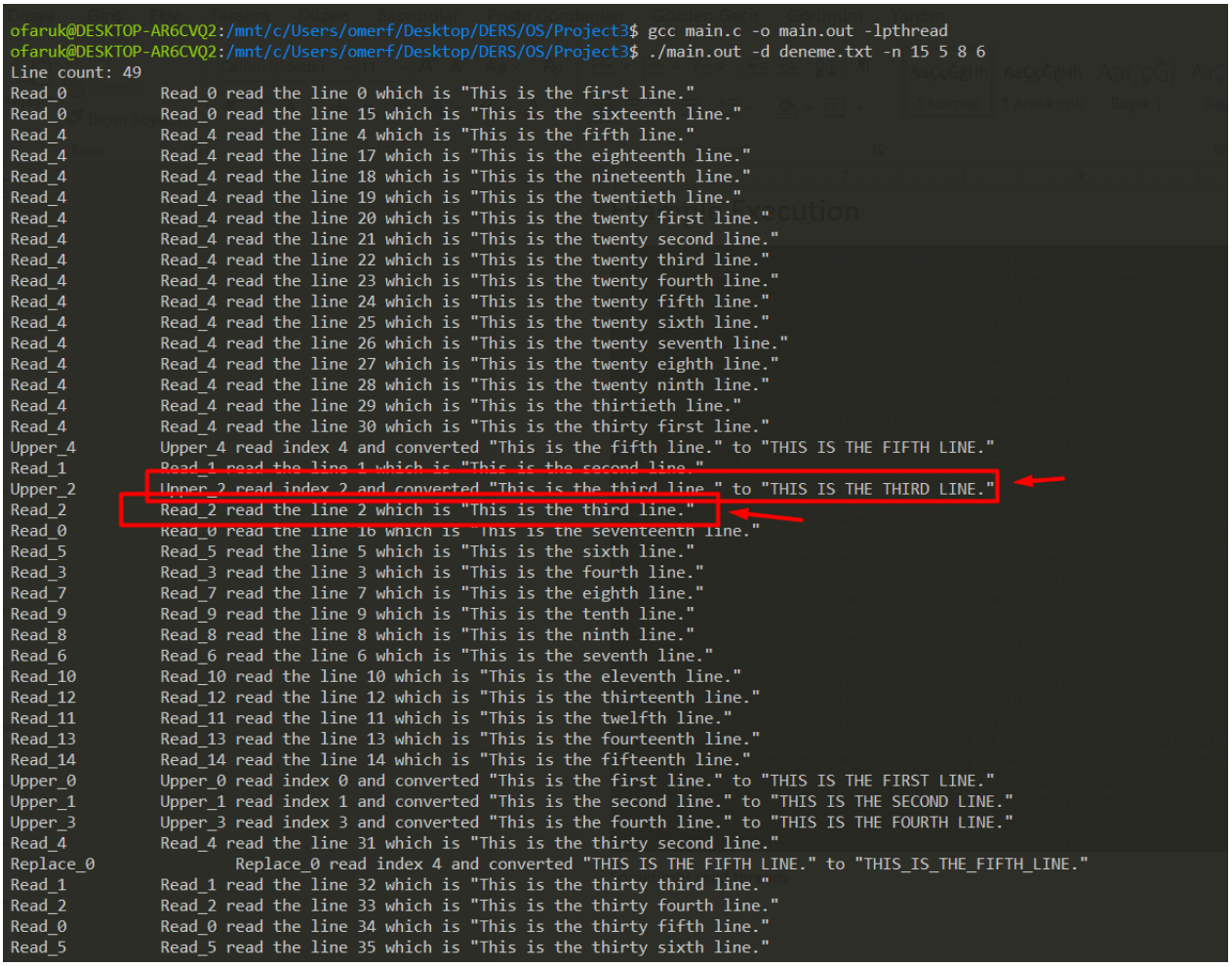
The getWriteIndex() method inside write() used to get lines which are already process by upper threads and replace threads and ready to write into file.

getWriteIndex() function check records[index].upper and records[index].replace values one by one and return the first record index which has upper=1 and replace=1.

Write thread does its job by executing writeToLine() function and tell which value to write into which file’s which line. writeToLine and getWriteIndex functions both executed inside wireteMutex mutex.

writeToLine() accomplish the writing task by using a temporary text file instead directly editing the original file. Because we think that this is not the objective of this project and other way was way more complicated to code. So that first we write new text which changed line into a new text file and we rename it to the original text file, after that we delete the temporary text file so it seems like we just edited the original file.

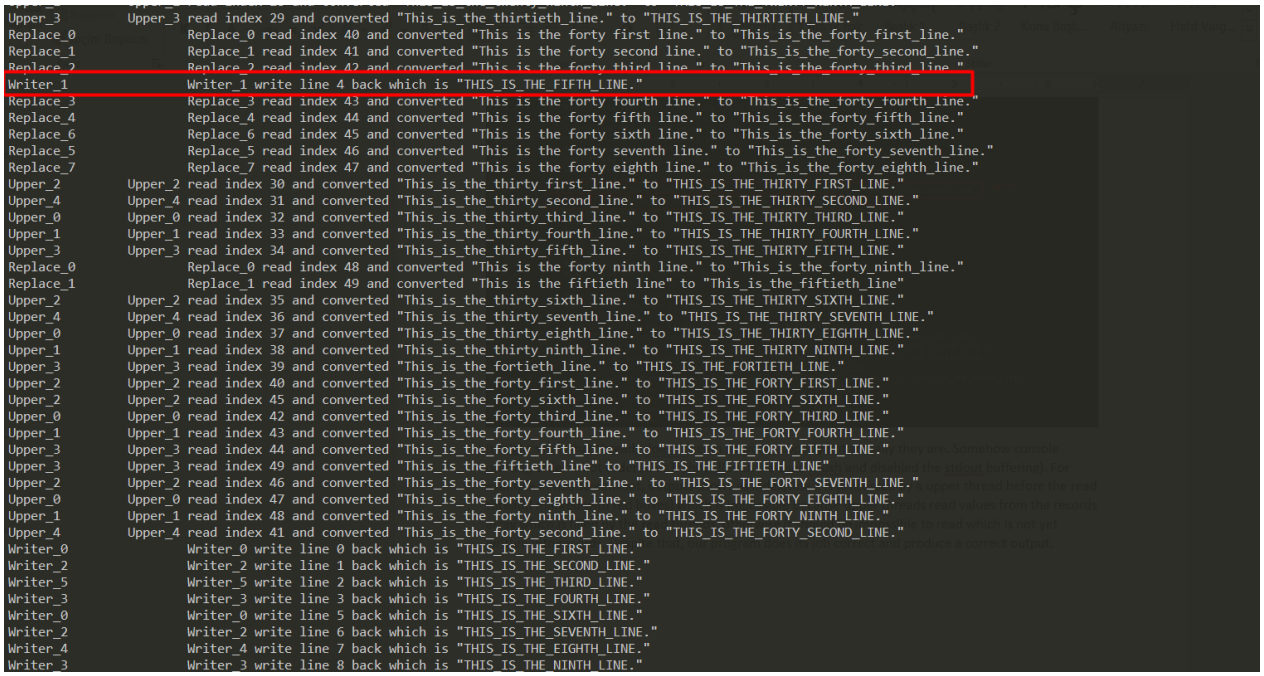
Example Execution



It might seem that threads doesn't seem to work properly but actually they are. Somehow console printing order is not ordered as it should(even we used fflush and disabled the stdout buffering). For example in this screenshot, outputs shows as index/line 2 is processed by a upper thread before the read thread(I marked with red boxes) which is impossible because upper threads read values from the records array which is filled by the read threads, otherwise it should be impossible to read which is not yet written into records. Despite that, our program does its job correct and produce a correct output.

And it also seems like a single read thread blocking other read threads and reads many line at once, I think it because we already opened the same file to count its lines and Operating System is cached it. We believe that's why readers threads are running that much fast then they should be in normal.

Also, our write threads outputs are all together, that doesn't mean our writer threads are not asynchronous, they are executing concurrently as given in this screenshot:



In this run time, only a writer thread manage to run before other replace and upper threads. Because write threads are way more slower compared to upper and replace threads because the need disk I/O operations.

Thanks