FACULTY OF ENGINEERING DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

CSE 3033- OPERATING SYSTEMS
MULTITHREAD PROJECT

STUDENTS	NUMBERS
Serkan Koç	150118073
Bahadır Alacan	150118042
Mustafa Yanar	150118048

Submitted To:
Doç.Dr Ali Haydar Özer

Due Date:
11.01.2023

Implementation Details

```c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <semaphore.h>
#include <ctype.h>

// prototypes of the functions
void createThreads(int READ_THREADS, int UPPER_THREADS, int REPLACE_THREADS, int WRITE_THREADS, const pthread_t *readThreads,
             const pthread_t *upperThreads, const pthread_t *replaceThreads, const pthread_t *writeThreads);

void joinThreads(int READ_THREADS, int UPPER_THREADS, int REPLACE_THREADS, int WRITE_THREADS, const pthread_t *readThreads,
            const pthread_t *upperThreads, const pthread_t *replaceThreads, const pthread_t *writeThreads);

void calculateNumberOfLinesInFile(char *const *argv);

void *readFile(void *args);

void *upper(void *args);

void *replace(void *args);

void *writeFile(void *args);
```

First of all included necessary libraries and then created function prototypes.

```c
// constants
#define MAX_LINE_LENGTH 256
#define TOTAL_LINES 100000
// Threads
pthread_mutex_t mutexLineBuffer; // to lock the buffer
pthread_mutex_t mutexFile; // to lock the file when writing
// Semaphores to implement consumer producer problem in a chained way.
// for example read is producer and upper is consumer then upper is producer and writer is consumer
sem_t semEmpty1;
sem_t semFull1;
sem_t semEmpty2;
sem_t semFull2;

// Global Variables
int numberOfLinesInFile;
int index1 = 0; // to arrange which line where the thread left operating
int index2 = 0;

struct line_struct {
    char line[MAX_LINE_LENGTH];
};
struct line_struct line_buffer[TOTAL_LINES];



char fileName[50];
// keep track of operated lines
int countReadLine = 0;
int countUpperedLine = 0;
int countWrittenLine = 0;
```

Then defined some constants. Defined two mutexes. mutexLineBuffer is used for locking buffer when read, upper threads are working. mutexFile locks the output file

when writing to txt. Indexes are used for keeping track of where we left reading text file or writing to text file. Defined a struct here which holds an array of characters. Also created an array of this which is line_buffer which holds the line in it. Count values keep track of the last line that is operated by the threads.

```
int main(int argc,char *argv[]){

    //check for errors
    char* errorMessage = "ERROR: Usage: ./project3.o -d filename.txt -n #ReadThreads #UpperThreads #ReplaceThreads #WriteThreads\n";
    if (argc != 8){
        fprintf( stream: stderr,  format: "%s", errorMessage);
        return 1;
    }
    else if (strcmp(argv[1], "-d") != 0){
        fprintf( stream: stderr,  format: "%s", errorMessage);
        return 1;
    }
    else if (access( name: argv[2],  type: F_OK) == -1){
        fprintf( stream: stderr,  format: "ERROR: File does not exists !\n");
        return 1;
    }
    else if (strcmp(argv[3], "-n") != 0){
        fprintf( stream: stderr,  format: "%s", errorMessage);
        return 1;
    }
```

In main first checking for errors for command line arguments.

```
// define threads
int READ_THREADS = atoi( nptr: argv[4]);
int UPPER_THREADS = atoi( nptr: argv[5]);
int REPLACE_THREADS = atoi( nptr: argv[6]);
int WRITE_THREADS = atoi( nptr: argv[7]);

pthread_t readThreads[READ_THREADS];
pthread_t upperThreads[UPPER_THREADS];
pthread_t replaceThreads[REPLACE_THREADS];
pthread_t writeThreads[WRITE_THREADS];
```

Then defined threads with the user input values.

```
//mutex initialization
pthread_mutex_init( mutex: &mutexLineBuffer,  mutexattr: NULL);
pthread_mutex_init( mutex: &mutexFile,  mutexattr: NULL);


//semaphore initialization
sem_init( sem: &semEmpty1,  pshared: 0,  value: numberOfLinesInFile);
sem_init( sem: &semFull1,  pshared: 0,  value: 0);
sem_init( sem: &semEmpty2,  pshared: 0,  value: numberOfLinesInFile);
sem_init( sem: &semFull2,  pshared: 0,  value: 0);
```

Initialized the mutexes and semaphores.

```
void createThreads(int READ_THREADS, int UPPER_THREADS, int REPLACE_THREADS, int WRITE_THREADS, const pthread_t *readThreads,
            const pthread_t *upperThreads, const pthread_t *replaceThreads, const pthread_t *writeThreads) {
    long i;
    int rc;

    for (i = 0; i < READ_THREADS; i++){
        rc = pthread_create( newthread: &readThreads[i],  attr: NULL,  start_routine: &readFile,  arg: (void *)i);
        printf( format: "Read Thread %d is created\n", i);
        if (rc){
            printf( format: "ERROR; return code from pthread_create() is %d\n", rc);
            exit( status: -1);
        }
    }
    for (i = 0; i < UPPER_THREADS; i++){
        rc = pthread_create( newthread: &upperThreads[i],  attr: NULL,  start_routine: &upper,  arg: (void *)i);
        printf( format: "Upper Thread %d is created\n", i);
        if (rc){
            printf( format: "ERROR; return code from pthread_create() is %d\n", rc);
            exit( status: -1);
        }
    }
//    for (i = 0; i < REPLACE_THREADS; i++){ /* creating replace threads */
//        rc = pthread_create(&replaceThreads[i], NULL, &replace, (void *)i);
//        printf("Replace Thread %d is created\n", i);
//        if (rc){
//            printf("ERROR; return code from pthread_create() is %d\n", rc);
//            exit(-1);
//        }
//    }
    for (i = 0; i < WRITE_THREADS; i++){ /* creating write threads */
        rc = pthread_create( newthread: &writeThreads[i],  attr: NULL,  start_routine: &writeFile,  arg: (void *)i);
        printf( format: "Write Thread %d is created\n", i);
        if (rc){
            printf( format: "ERROR; return code from pthread_create() is %d\n", rc);
            exit( status: -1);
        }
    }
}
```

Create thread method creates the threads with the user input values.

```
void *readFile(void *args){
    long tid;
    tid = (long)args;
    while (1){
        // Produce
        FILE *file = fopen( filename: fileName, modes: "r");
        if (file == NULL) {
            perror( s: "Error opening file");
            return (NULL);
        }
//        printf("%d",countReadLine);
        char line[MAX_LINE_LENGTH];
        // YOU CAN USE THE BELOW SLEEP TO KEEP TRACK OF OUTPUT IN SLOW MOTION
//        sleep(4);
// this semEmpty1 is initialized with the number of lines so this will work
// until the end of file
        sem_wait( sem: &semEmpty1);
        // locking the mutex because there are common variables
        pthread_mutex_lock( mutex: &mutexLineBuffer);
        // fseek arranges the line index to be read
        fseek( stream: file, off: index1, whence: SEEK_SET);
        if (fgets( s: line, n: sizeof(line), stream: file) != NULL) {
//            printf("%s" ,line);
            strcpy( dest: line_buffer[countReadLine].line, src: line);
            printf( format: "Read_%ld      Read_%ld read the line %d which is \"%s\"\n", tid, tid, countReadLine,
                    line_buffer[countReadLine].line);
//            printf("--R%ld--%s\n",tid, line_buffer[countReadLine].line);
            countReadLine++;
//            printf("%s", line_buffer->line);

//            sleep(4);
        }
        index1 += strlen( s: line);
        pthread_mutex_unlock( mutex: &mutexLineBuffer);
        sem_post( sem: &semFull1); // post semFull1 so that upper thread starts working
        fclose( stream: file);

    }
```

After the creation of threads read thread starts to work. In here thread id is passed as argument to this function. Whole project works as consumer producer problem in a chained way. Read thread is producer here for the upper thread. So the solution to that is using semaphores. semEmpty1 semaphore is initialized with the number of lines in the text file. When semEmpty1 reaches to 0 read thread will not work anymore. Then locked the buffer. Because we are not only reading text file here but also writing them to buffer. In critical section we read the text file and add the line to the buffer. Also increment countReadLine by one so that we can keep track of the order of the lines. After that unlocked the mutex and used sem_post(&semFull1). This will let the upper thread to know that there is an element in the buffer that you can work with.

```
void *upper(void *args){
    long tid;
    tid = (long)args;
    while (1) {
        // this semaphore is used if read thread adds something to buffer initalized with 0
        sem_wait( sem: &semFull1);
        pthread_mutex_lock( mutex: &mutexLineBuffer);

         printf( format: "Upper_%ld     Upper_%ld read index %d and converted “%s”", tid, tid, countUpperedLine,
                line_buffer[countUpperedLine].line);
//           printf("U%ld ",tid);
          for (int j = 0; line_buffer[countUpperedLine].line[j] != '\0'; j++) {
              line_buffer[countUpperedLine].line[j] = toupper( c: (unsigned char)line_buffer[countUpperedLine].line[j]);
          }
//          printf("--U%ld--%s\n",tid, line_buffer[countUpperedLine].line);

              printf( format: " to “%s”\n", line_buffer[countUpperedLine].line);
          countUpperedLine++;

        pthread_mutex_unlock( mutex: &mutexLineBuffer);


        sem_post( sem: &semFull2); // post semFull2 so that write thread starts working
    }
```

When there is a line in the buffer then the upper method can start working. Likewise the upper method consumes the read method and produces for the write method. semFull1 is waited here that lets the method to know that there is a line in the buffer or not. Locking the buffer then converting the line to upper case. Also incrementing countUpperedLine by one to keep track of where we left. At the end of this method sem_post(&semFull2) is used to let the write method know that there is a line that the write method can write to a text file.

```
void *writeFile(void *args){
    long tid;
    tid = (long)args;
    while (1){
        // this semaphore is used if upper thread converts the line so that write can work
        sem_wait( sem: &semFull2);
        pthread_mutex_lock( mutex: &mutexFile);
        FILE *fp;
        fp = fopen( filename: fileName,  modes: "r+");
        fseek( stream: fp,  off: index2,  whence: SEEK_SET);
        fputs( s: line_buffer[countWrittenLine].line,  stream: fp);
        fflush( stream: fp);
        index2 += strlen( s: line_buffer[countWrittenLine].line);
        fclose( stream: fp);
        printf( format: "Writer_%ld      Writer_%ld write line %d back which is “%s”\n", tid, tid,
                countWrittenLine, line_buffer[countWrittenLine].line);
//        printf("--W%ld--%s\n",tid, line_buffer[countWrittenLine].line);
        countWrittenLine++;
        pthread_mutex_unlock( mutex: &mutexFile);
    }
}
```

writeFile method is the consumer of the upper method. There is semFull2 semaphore waiting for a line that is converted to uppercase. Locking the text file so that other write threads can not access it at the same time. Doing the writing and incrementing the countWrittenLine by one.

```
void joinThreads(int READ_THREADS, int UPPER_THREADS, int REPLACE_THREADS, int WRITE_THREADS, const pthread_t *readThreads,
            const pthread_t *upperThreads, const pthread_t *replaceThreads, const pthread_t *writeThreads) {
    long i;
    for (i = 0; i < READ_THREADS; i++)
        pthread_join( th: readThreads[i],  thread_return: NULL);

    for (i = 0; i < UPPER_THREADS; i++)
        pthread_join( th: upperThreads[i],  thread_return: NULL);

//    for (i = 0; i < REPLACE_THREADS; i++)
//        pthread_join(replaceThreads[i], NULL);
//
    for (i = 0; i < WRITE_THREADS; i++)
        pthread_join( th: writeThreads[i],  thread_return: NULL);
}
```

```
// joining threads
joinThreads(READ_THREADS, UPPER_THREADS, REPLACE_THREADS, WRITE_THREADS, readThreads, upperThreads, replaceThreads,
            writeThreads);

// destroying
sem_destroy( sem: &semEmpty1);
sem_destroy( sem: &semFull1);
sem_destroy( sem: &semEmpty2);
sem_destroy( sem: &semFull2);

pthread_mutex_destroy( mutex: &mutexLineBuffer);
pthread_mutex_destroy( mutex: &mutexFile);
```

At the last section of the main method we joined the threads and destroy the mutexes and semaphores.

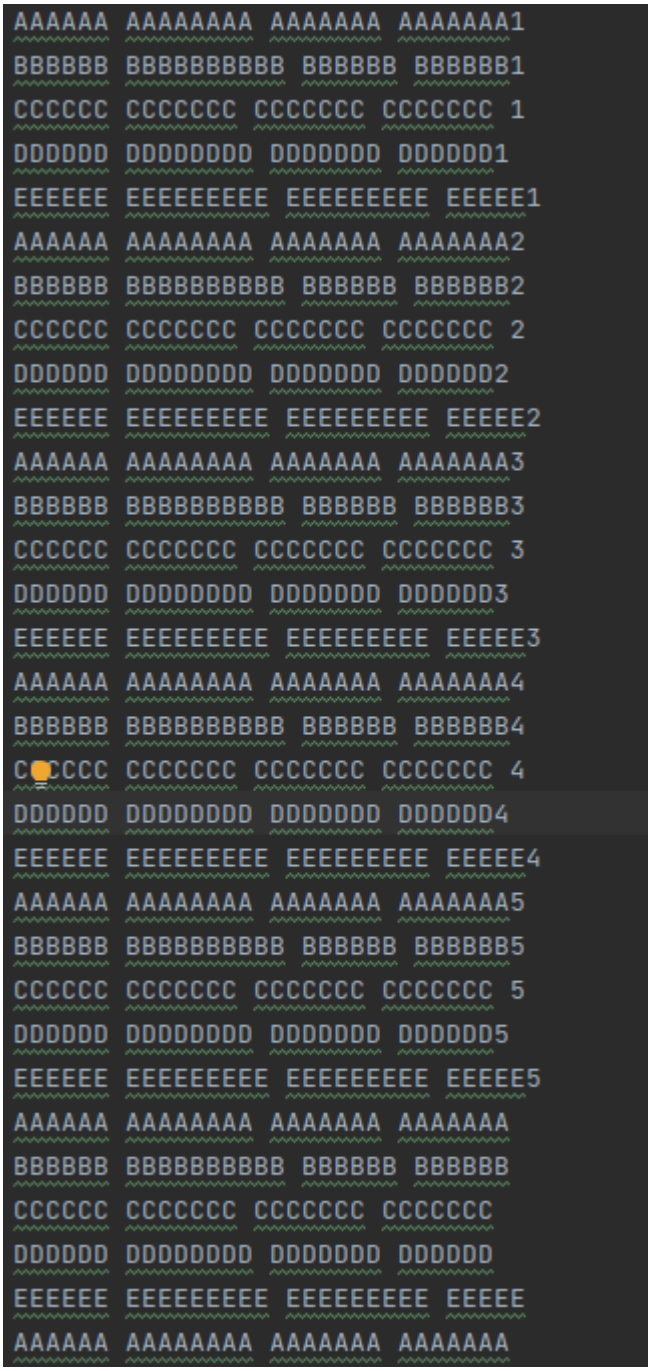
Some of The Outputs

```
aaaaaa aaaaaaaa aaaaaaa aaaaaaa1
bbbbbb bbbbbbbbb bbbbbb bbbbbb1
cccccc ccccccc ccccccc ccccccc 1
dddddd ddddddd ddddddd dddddd1
eeeeee eeeeeeeee eeeeeeeee eeeee1
aaaaaa aaaaaaaa aaaaaaa aaaaaaa2
bbbbbb bbbbbbbbb bbbbbb bbbbbb2
cccccc ccccccc ccccccc ccccccc 2
dddddd ddddddd ddddddd dddddd2
eeeeee eeeeeeeee eeeeeeeee eeeee2
aaaaaa aaaaaaaa aaaaaaa aaaaaaa3
bbbbbb bbbbbbbbb bbbbbb bbbbbb3
cccccc ccccccc ccccccc ccccccc 3
dddddd ddddddd ddddddd dddddd3
eeeeee eeeeeeeee eeeeeeeee eeeee3
aaaaaa aaaaaaaa aaaaaaa aaaaaaa4
bbbbbb bbbbbbbbb bbbbbb bbbbbb4
cccccc ccccccc ccccccc ccccccc 4
dddddd ddddddd ddddddd dddddd4
eeeeee eeeeeeeee eeeeeeeee eeeee4
aaaaaa aaaaaaaa aaaaaaa aaaaaaa5
bbbbbb bbbbbbbbb bbbbbb bbbbbb5
cccccc ccccccc ccccccc ccccccc 5
dddddd ddddddd ddddddd dddddd5
eeeeee eeeeeeeee eeeeeeeee eeeee5
aaaaaa aaaaaaaa aaaaaaa aaaaaaa
bbbbbb bbbbbbbbb bbbbbb bbbbbb
cccccc ccccccc ccccccc ccccccc
dddddd ddddddd ddddddd dddddd
eeeeee eeeeeeeee eeeeeeeee eeeee
aaaaaa aaaaaaaa aaaaaaa aaaaaaa
bbbbbb bbbbbbbbb bbbbbb bbbbbb
cccccc ccccccc ccccccc ccccccc
dddddd ddddddd ddddddd dddddd
eeeeee eeeeeeeee eeeeeeeee eeeee
aaaaaa aaaaaaaa aaaaaaa aaaaaaa
bbbbbb bbbbbbbbb bbbbbb bbbbbb
cccccc ccccccc ccccccc ccccccc
```

input file

```
Read_1      Read_1 read the line 0 which is "aaaaaa aaaaaaaa aaaaaaa aaaaaaa1
"
Read_1      Read_1 read the line 1 which is "bbbbbb bbbbbbbbbb bbbbbb bbbbbb1
"
Read_1      Read_1 read the line 2 which is "cccccc ccccccc ccccccc ccccccc 1
"
Upper_5     Upper_5 read index 0 and converted “aaaaaa aaaaaaaa aaaaaaa aaaaaaa1
” to “AAAAAA AAAAAAAA AAAAAAA AAAAAAA1
”
Upper_5     Upper_5 read index 1 and converted “bbbbbb bbbbbbbbbb bbbbbb bbbbbb1
” to “BBBBBB BBBBBBBBBB BBBBBB BBBBBB1
”
Upper_5     Upper_5 read index 2 and converted “cccccc ccccccc ccccccc ccccccc 1
” to “CCCCCC CCCCCCC CCCCCCC CCCCCCC 1
”
Writer_6      Writer_6 write line 0 back which is “AAAAAA AAAAAAAA AAAAAAA AAAAAAA1
”
Writer_6      Writer_6 write line 1 back which is “BBBBBB BBBBBBBBBB BBBBBB BBBBBB1
”
Writer_6      Writer_6 write line 2 back which is “CCCCCC CCCCCCC CCCCCCC CCCCCCC 1
”
Read_1      Read_1 read the line 3 which is "dddddd dddddddd ddddddd ddddddd1
"
```

```
Writer_6      Writer_6 write line 416 back which is “BBBBBB BBBBBBBBBB BBBBBB BBBBBB
”
Upper_3     Upper_3 read index 417 and converted “cccccc ccccccc ccccccc ccccccc
” to “CCCCCC CCCCCCC CCCCCCC CCCCCCC
”
Writer_5      Writer_5 write line 417 back which is “CCCCCC CCCCCCC CCCCCCC CCCCCCC
”
Read_0      Read_0 read the line 420 which is "aaaaaa aaaaaaaa aaaaaaa aaaaaaa
"
Read_0      Read_0 read the line 421 which is "bbbbbb bbbbbbbbbb bbbbbb bbbbbb
"
Read_0      Read_0 read the line 422 which is "cccccc ccccccc ccccccc ccccccc
"
Upper_5     Upper_5 read index 418 and converted “dddddd dddddddd ddddddd dddddd
” to “DDDDDD DDDDDDDD DDDDDDD DDDDDD
”
Upper_5     Upper_5 read index 419 and converted “eeeeee eeeeeeeee eeeeeeeee eeeee
” to “EEEEEE EEEEEEEEE EEEEEEEEE EEEEE
”
Upper_0     Upper_0 read index 420 and converted “aaaaaa aaaaaaaa aaaaaaa aaaaaaa
” to “AAAAAA AAAAAAAA AAAAAAA AAAAAAA
```


AAAAAA AAAAAAAA AAAAAAA AAAAAAA1
BBBBBB BBBBBBBBBB BBBBBB BBBBBB1
CCCCCC CCCCCCC CCCCCCC CCCCCCC 1
DDDDDD DDDDDDDD DDDDDDD DDDDDD1
EEEEEE EEEEEEEEE EEEEEEEEE EEEEE1
AAAAAA AAAAAAAA AAAAAAA AAAAAAA2
BBBBBB BBBBBBBBBB BBBBBB BBBBBB2
CCCCCC CCCCCCC CCCCCCC CCCCCCC 2
DDDDDD DDDDDDDD DDDDDDD DDDDDD2
EEEEEE EEEEEEEEE EEEEEEEEE EEEEE2
AAAAAA AAAAAAAA AAAAAAA AAAAAAA3
BBBBBB BBBBBBBBBB BBBBBB BBBBBB3
CCCCCC CCCCCCC CCCCCCC CCCCCCC 3
DDDDDD DDDDDDDD DDDDDDD DDDDDD3
EEEEEE EEEEEEEEE EEEEEEEEE EEEEE3
AAAAAA AAAAAAAA AAAAAAA AAAAAAA4
BBBBBB BBBBBBBBBB BBBBBB BBBBBB4
CCCCCC CCCCCCC CCCCCCC CCCCCCC 4
DDDDDD DDDDDDDD DDDDDDD DDDDDD4
EEEEEE EEEEEEEEE EEEEEEEEE EEEEE4
AAAAAA AAAAAAAA AAAAAAA AAAAAAA5
BBBBBB BBBBBBBBBB BBBBBB BBBBBB5
CCCCCC CCCCCCC CCCCCCC CCCCCCC 5
DDDDDD DDDDDDDD DDDDDDD DDDDDD5
EEEEEE EEEEEEEEE EEEEEEEEE EEEEE5
AAAAAA AAAAAAAA AAAAAAA AAAAAAA
BBBBBB BBBBBBBBBB BBBBBB BBBBBB
CCCCCC CCCCCCC CCCCCCC CCCCCCC
DDDDDD DDDDDDDD DDDDDDD DDDDDD
EEEEEE EEEEEEEEE EEEEEEEEE EEEEE
AAAAAA AAAAAAAA AAAAAAA AAAAAAA
output file