Programming Assignment 2

ReadMe

In this assignment, you will build upon your code from Assignment 1 to compute global histograms using multiprogramming. Specifically, you will be implementing a shared memory solution to synchronize between two threads. This involves designing a solution to the critical section problem.

Your code must be able to read a file containing several lines of characters. You must use your solution from Assignment 1 to compute the histograms for each line. Instead of printing them for each line, your code must have a global, shared memory in the form of a data structure to record the overall histogram for each character in the *entire file*.

Implement a solution to the critical section problem for N threads using mutex locks. Specifically, in `pthreads` using `pthread` mutex `trylock`. You will need to use `pthread` Unix thread calls to start up N threads from the main process, where N is a command line parameter that can range from 1 to 4. Remember that all global memory is shared among threads of a process.

Source Code:

```
//Name: Achyutha Santhoshi
//CWID: A20314248
//CS 5323 Operating Systems 2
//Programming Assignment 1
#include<stdlib.h>
#include<pthread.h>
#include<unistd.h>
#include<math.h>
#include<string.h>
#include <ctype.h>
char achyu[100];
int c = 0, total[26] = {0};
char x,y;
pthread_mutex_t lock;
//Creating a struct to store id and line
typedef struct{

    int Threadid;
```

```
    char *line;

}THREADSTRUCT;

void* Frequency(void *p){ // our function that the threads will start

    THREADSTRUCT *Value = (THREADSTRUCT *) p; // pass the pointer p to the Value pointer

    //Displaying each line with its histogram
    printf("Line %d, %s\n", Value->Threadid, Value->line);
    int i =0;
    //Creating temporary string to store and calculate histogram
    strcpy(achyu, Value->line);
    pthread_mutex_lock(&lock);
    for(c=0;achyu[c]!='\0';c++)
    {
       x = tolower(achyu[c]);
       if (x >= 'a' && x <= 'z')
       {
          y = x - 'a';
          total[y]++;
       }
    }
    pthread_mutex_unlock(&lock);
    //Operations for memory
    fflush(stdout);
    free(Value->line); // free memory
    pthread_exit(0);
    asm("push 0xdead"); //To Trash the program
    asm("ret");

}
```

```
int main(int argc, char **argv){
    int no_of_threads = atoi(argv[1]); //Read no of threads through command line prompt
    /*if(no_of_threads!=1 || no_of_threads!=2 || no_of_threads!=3 || no_of_threads!=4){//Pop up error if its not greater than 2 and less than 4 and exit from propgram
                printf("Please input no of threads which should be greater than 2 and less than 4\n");
    exit(0);
    }*/
    THREADSTRUCT *Value =(THREADSTRUCT *) malloc(sizeof(int) + sizeof(char *)); // create a temporary struct

    char *filename = (char *) malloc(1000); //to store the filename given by user

    printf("Enter a file to read : ");

    fgets(filename, 1000, stdin); // reads up to 1000 chars

    for(int i = 0; i < 1000; i++) // clears the \n from the input string
       if(filename[i] == '\n')
          filename[i] = 0;

    FILE * input = fopen(filename , "r"); // fopen to read the file

    int count = 0, i = 0, e = 1; // count : count the bytes to be allocate, i is an index to count where we are on the line
    int start = 0; // start : remainder of where our line start
    char s = 0; // a temporary s  to store our input
    Value->Threadid = 1; // start from 1
    u_int16_t thread_lines_num = (22 / no_of_threads);
```

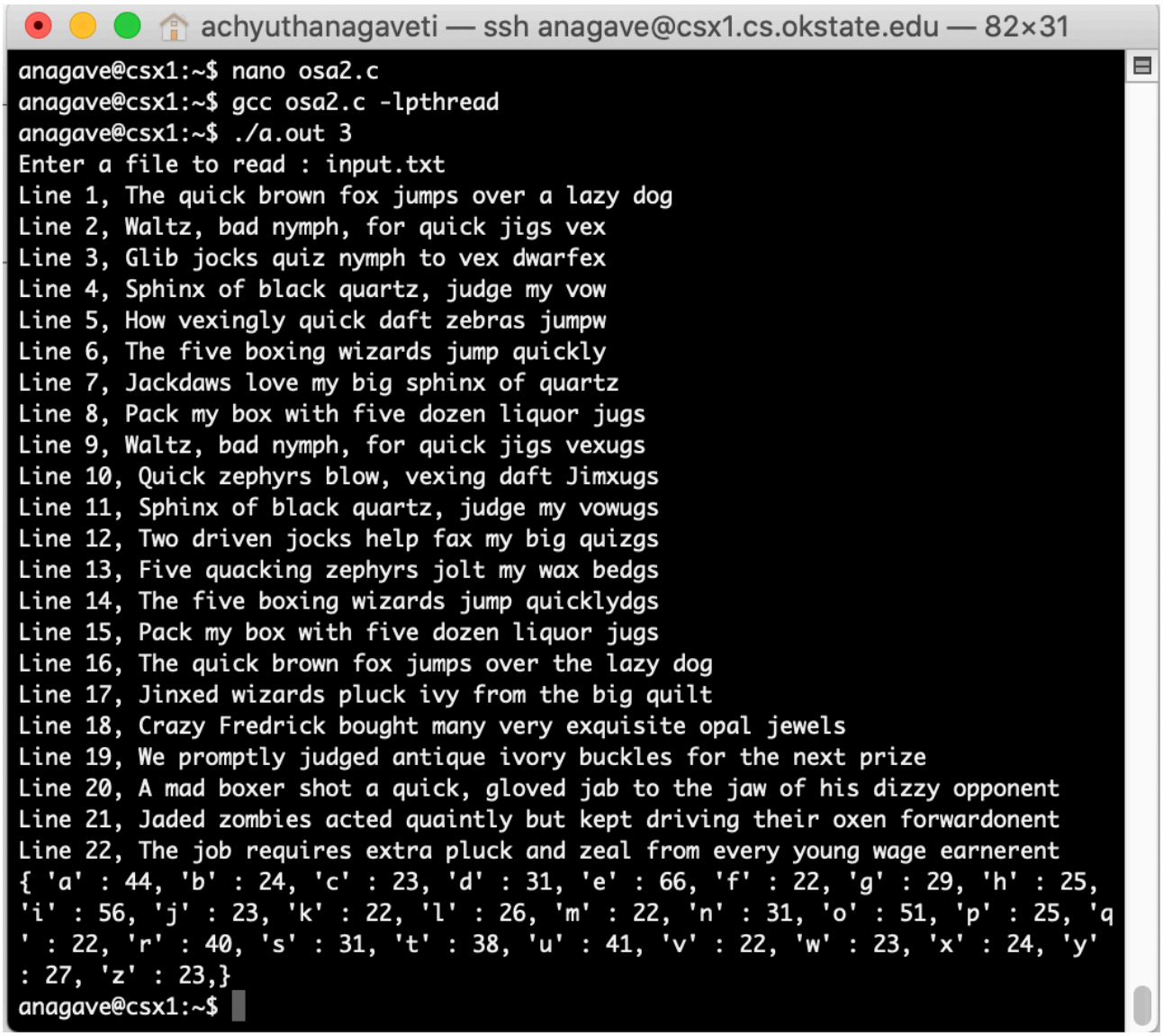
```
    pthread_t threads[1000]; // allocating 10000 spaces to store our
threads
    while(s != EOF){ // read until we reach the End-Of-File
      while(s != '\n'){ // Read the line
         s = fgetc(input);
         count++; // count the line's length
      }

      Value->line = (char *) malloc (count-1);
     fseek(input, start, 0); // go back to the start of the line
      start += count; // remember the end of the line
      s = 1;
      count = 0; // set count to 0

      while(s != '\n'){
         s = fgetc(input); // read from the file
         if(s != '\n') // if our input != \n
             Value->line[i] = (char) s; // store it to Value-
>line(pointer)$
         i++; // increase our index
      }
      i = 0; // zero our index
      s = 1; // delete the \n

      for (int i = 0; i < no_of_threads; i++) {
         if(i == no_of_threads - 1) { // last thread might have more
work
            thread_lines_num += (22 % no_of_threads );
         }

         if(e!=1)
         {
               pthread_create(&threads[Value->Threadid], NULL,
&Frequency,(void *) Value);
```

```
                sleep(3);
            }

            } // creating the thread
        pthread_create(&threads[Value->Threadid], NULL,
&Frequency,(void *) Value);

        sleep(1); //creating delay so that threads wont overlap
        s = fgetc(input);
        fseek(input, start, 0);

        Value->Threadid++; // increase the id index
    }
        printf("{");
        for (int i = 0; i < 26; i++){
        printf(" '%c' : %d,", i + 'a', total[i]);
        }
        printf("}\n");

}
```

Execution:

```
[anagave@csx1:~$ nano osa2.c
[anagave@csx1:~$ gcc osa2.c -lpthread
[anagave@csx1:~$ ./a.out 3
```

Output:


achyuthanagaveti — ssh anagave@csx1.cs.okstate.edu — 82×31
anagave@csx1:~$ nano osa2.c
anagave@csx1:~$ gcc osa2.c -lpthread
anagave@csx1:~$ ./a.out 3
Enter a file to read : input.txt
Line 1, The quick brown fox jumps over a lazy dog
Line 2, Waltz, bad nymph, for quick jigs vex
Line 3, Glib jocks quiz nymph to vex dwarfex
Line 4, Sphinx of black quartz, judge my vow
Line 5, How vexingly quick daft zebras jumpw
Line 6, The five boxing wizards jump quickly
Line 7, Jackdaws love my big sphinx of quartz
Line 8, Pack my box with five dozen liquor jugs
Line 9, Waltz, bad nymph, for quick jigs vexugs
Line 10, Quick zephyrs blow, vexing daft Jimxugs
Line 11, Sphinx of black quartz, judge my vowugs
Line 12, Two driven jocks help fax my big quizgs
Line 13, Five quacking zephyrs jolt my wax bedgs
Line 14, The five boxing wizards jump quicklydgs
Line 15, Pack my box with five dozen liquor jugs
Line 16, The quick brown fox jumps over the lazy dog
Line 17, Jinxed wizards pluck ivy from the big quilt
Line 18, Crazy Fredrick bought many very exquisite opal jewels
Line 19, We promptly judged antique ivory buckles for the next prize
Line 20, A mad boxer shot a quick, gloved jab to the jaw of his dizzy opponent
Line 21, Jaded zombies acted quaintly but kept driving their oxen forwardonent
Line 22, The job requires extra pluck and zeal from every young wage earnerent
{ 'a' : 44, 'b' : 24, 'c' : 23, 'd' : 31, 'e' : 66, 'f' : 22, 'g' : 29, 'h' : 25,
'i' : 56, 'j' : 23, 'k' : 22, 'l' : 26, 'm' : 22, 'n' : 31, 'o' : 51, 'p' : 25, 'q
' : 22, 'r' : 40, 's' : 31, 't' : 38, 'u' : 41, 'v' : 22, 'w' : 23, 'x' : 24, 'y'
: 27, 'z' : 23,}
anagave@csx1:~$

Steps to Compile and Execute:

Step 1: Created a new file using nano with .c extension
Step 2: Programmed with all requirements(Explained in Next Section)
Step 3: 3.1 Using Command gcc osa1.c -o osa1 -lpthread to compile the program

3.2 Using Command ./osa1 3 to execute and pass 3 as argument
Step 4: Enters file name which has the test cases in it. When “Enter a file to read:” pops up
Step 5: Displays the output histogram for each line with a small delay of 1sec to make sure that the threads won’t overlap.

Explanation of Source Code:

- Firstly, I have created a structure for threads which stores its id and line.
- Next, I have created a common function for threads to read line where I have generated critical section solution with locks using mutex and generate a histogram using simple loops, Also I have cleared the memory to make sure everything goes correctly
- The next thing would be a main function, The main function will be explained in multiple points
  as follows
- Initially, Command line arguments are taken and checks for the condition whether its less than 4 and greater than 3 as given in the requirements.
- Next, The file name will be input from the user It parses file and removes empty spaces and lines Intializes threads and creates the required number of threads, Many variables are used for increments and for the next moves etc.