Assignment 5

1 Synopsis

Please read the document, ***General Requirements Programming Assignments***, posted on the course website and make sure you follow it when you do this assignment. This assignment is designed to be similar to Assignment 2. You are to write a multi-threaded program that searches for all occurrences of a pattern in an input file, replaces those occurrences by a redaction string, and writes the modified file to a specified output file. For this program you will use the *Pthreads* library.

2 Program Invocation, Input, and Output

This program will be called `redact`. The program must find ***all*** occurrences of a particular string, which we will call the ***pattern***, in a second, usually much longer, string which we will call the ***text***. Each occurrence is to be ***redacted***, meaning the copy in the original text is to be replaced by a redaction string, as if it were blacked out using a black marker. Program invocation is different than it was in the second assignment.

The `redact` program expects four command line arguments: the number of threads to invoke, the string to be matched, henceforth called the ***pattern***, the pathname to the input text file that contains the string to be searched, and the pathname of the file in which the program will write its output. Correct usage is

```
redact <number of threads> <pattern>  <input file> <output file>
```

If the pattern contains characters special to the shell, or blanks, it should be enclosed in single quotes or double quotes, depending on which characters it contains. For example, to redact all occurrence of the pattern "Lincoln is on a $5 bill" in the file `denominations` , writing to the file `denominations.redact` using 16 threads, you would enter

```
redact 16 'Lincoln is on a $5 bill' denominations  denominations.redact
```

or alternatively

```
redact 16 "Lincoln is on a \$5 bill" denominations denominations.redact
```

whereas to redact the pattern "Lincoln's on a $5 bill" in denominations, you have to enter

```
redact 16 "Lincoln\'s on a \$5 bill" denominations denominations.redact
```

The pattern and the text can contain newline characters. These are treated as ordinary characters. A newline matches a newline. A pattern with a newline can be entered on the command line in one of two ways, by preceding the newline with a backslash, or by enclosing the entire string in single quotes:

```
redact 16 "Lincoln\
is on a \$5 bill" denominations denominations.redact

redact 16 'Lincoln
is on a $5 bill' denominations denominations.redact
```

2.1 Redaction String

- The redaction string shall be a string whose length is equal to the length of the pattern, so that the number of characters in the file is unchanged. Each thread will use a redaction string based on the following rule:
- A redaction string is a string consisting of a single character repeated M times, where M is the length of the pattern. This character is called the redaction character.
- The redaction character used by each thread is based on the thread's rank in the program. Threads will be assigned ranks from *0* to $N-1$ where N is the total number of threads. The set of redaction characters is the set of digits, lowercase letters, then uppercase letters, then underscore, then blank, in sequence, for a total of 64 different redaction characters:

  “`0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ_ `“
- If these characters are indexed r_0,r_1, ...,r_{63}, then the thread with rank k will use character $r_{k\%64}$as its redaction character.

2.1.1 Example and Clarification

A pattern might have the property that it overlaps itself. The pattern “`wallawalla`” in the text below is matched five times, as shown:

```
wallawallawallabingwallawallawallawallabang
wallawalla
     wallawalla
                   wallawalla
                        wallawalla
                             wallawalla
```

A pattern might extend across the two segments assigned to different threads, which is why there might be two different redaction strings used in the same part of the text. If different threads redact the above example, unless there is a precedence rule, the output would not be uniquely determined. The rule that disambiguates it is that a thread with higher rank has precedence over one with lower rank. If threads 0 through 4 were to redact in this example the output should be

```
000001111111111bing2222233333444444444bang
```

because thread 1 used 1's, overwriting the 0's that thread 0 used. Similarly, thread 4 used 4's, overwriting the 3's written by thread 3, and so forth.

2.2 Error Handling

If one or more arguments are missing, if the input file can not be opened for reading, or if the output file cannot be created or written to because of permissions, the program should print an error message on standard output that includes how to use it correctly and it should exit. If during processing the program results in an error such as being unable to allocate memory or other programmatic failures, it should print a message on standard error that it failed and it should exit.

3 Program Implementation Requirements

1. Only the main thread can perform input and output and usage error handling. It is an error if any other thread does so.
2. The program must produce correct results regardless of the size of the pattern or the input file. There is no upper bound on the lengths of either of these, up to the limits of the physical hardware.
3. The program must not use a library function such as `strstr()` to check for the pattern in the text. A process needs to check whether the pattern is in the text must use any one of the various string search algorithms that exist, such as *brute-force*, *Knuth-Morris-Pratt*, *Boyer-Moore*, or *Rabin-Karp*. I suggest a simple brute force algorithm.
4. The program should read the file into an array in memory and modify that array directly. It should not make a copy of that array, although it is free to make small copies of pieces of it as needed, $O(NM)$ where M is the pattern length and N is the number of threads. (That will make it easier to solve.)
5. The load should be balanced as uniformly as possible among the threads.
6. The program will have a critical section when two different threads try to write their redaction strings
7. The program must be documented and written to comply with the requirements stated in the ***General Requirements Programming Assignments*** referred to above.

4 Program Design Considerations

The program will be assessed on how well it is designed. The main thread must get the command line argument and process them. Each thread should work on a portion of the text string. You should use the same agglomeration scheme as was used in Floyd's algorithm, with each segment differing in length by at most one. When each thread has finished its work, it exits and the main thread must join them. The main thread must print output to the given file.

4.1 Getting File Size

There are two ways to get the size of a file without reading the file.

4.1.1 Method 1: Using `stat()`

If you need to determine the file size without opening the file, use the `stat()` system call (only in POSIX-compliant systems):

```c
#include <sys/stat.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdint.h>

int main( int argc, char* argv[])
{
    struct stat statbuf;

    /* Check usage */
    if ( 2 > argc ) {
        printf("Usage: %s filename\n", argv[0]);
        exit(1);
    }

    /* Call stat() to fill statbuf struct */)
    if ( stat(argv[1], &statbuf) == -1 ) {
        printf("Could not stat file %s\n", argv[1]);
        exit(1);
    }
    /* Print size of file. intmax_t is a type that holds big numbers */
    printf(" %8jd\n", (intmax_t)statbuf.st_size);
    return 0;
}
```

4.1.2 Method 2: Using `lseek()`

You can also use `lseek()` to get the file size, but this requires opening the file using the `open()` function defined in `<fcntl.h>`:

```
#include <sys/stat.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdint.h>
#include <fcntl.h>

int main( int argc, char* argv[])
{
    int      in_fd;
    intmax_t filesize;

    /* Check usage */
    if ( 2 > argc ) {
        printf("Usage: %s filename\n", argv[0]);
        exit(1);
    }

    /* open file    */
    if ( (in_fd = open(argv[1], O_RDONLY)) == -1 ) {
        fprintf( stderr, "could not open %s for reading\n", argv[1]);
        exit(1);
    }

    filesize = lseek(in_fd, 0, SEEK_END);

    close(in_fd);

    /* Print size of file. intmax_t is a type that holds big numbers */
    printf(" %8jd\n", (intmax_t)filesize);
    return 0;
}
```

5 Program Grading Rubric

The program will be graded based on the following rubric out of 100 points:

- The program must compile and run on any `cslab` host. If it does not compile and link on any `cslab` host, it loses 80 points.

- **Correctness** (60 points)
  - The program should do exactly what is described above. Incorrect output, incorrectly formatted output, missing output, or output containing other characters are all errors.
  - It must process patterns and text files of arbitrary size.
  - The program should produce the same output no matter how many threads it is given, except for the elapsed time.
  - It should handle errors correctly.
- **Design and clarity** (8 points)

  Is the program organized clearly? Are functions and variables designed in an appropriate way? Are the tasks divided appropriately? If not, the program will lose points.
- **Compliance with the Programming Rules** (16 points)

  Are all of the rules stated in that document observed? Programs that violate them will lose points accordingly.

6 Submitting the Homework

This project is due by the end of the day (i.e. 11:59PM, EST) on ***Sunday, December 15, 2019***. Follow these instructions precisely to submit it.

Your program should be a single source code file, written in C. It must have a `.c` suffix. It does not matter what base name you give it because the `submithwk` program will change its base name anyway to the form `username_hwk5.c` (or more accurately, `username_hwk5.`*XXX*, where *XXX* is the suffix you gave it.)

Assuming this file is in your current working directory and is named `myhwk5.c` you submit by entering the command

```
submithwk_cs49365 -t  5  myhwk5.c
```

The `-t` tells the command it is a plain text file. The program will copy your file into the `hwk4` sub-directory

```
/data/biocs/b/student.accounts/cs493.65/hwks/hwk5/
```

and if it is successful, it will display the message, "`File username_hwk5.c successfully submitted.`"

You can only run this command on a cslab host. You cannot run it on eniac, so remember to ssh into a cslab host before doing this!

You will not be able to read this file, nor will anyone else except for me. But you can double-check that the command succeeded by typing the command

```
ls -l /data/biocs/b/student.accounts/cs493.65/hwks/hwk5/
```

and making sure you see a non-empty file there.

If you put a solution there and then decide to change it before the deadline, just replace it by the new version. Once the deadline has passed, you cannot do this. I will grade whatever version is there at the end of the day on the due date. You cannot resubmit the program after the due date.