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 leverages some of the work you did in Assignment 3. You are to write a multi-threaded program that searches for ***all*** occurrences of a string which we will call the ***pattern***, in a second, usually much longer, string which we will call the ***text***, 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.

Your program should solve this problem efficiently and with speedup nearly proportional to p, the number of threads. In this context "efficiently", means that the efficiency of the program is high, i.e., the total *work* done by the parallel program is not much more than the work done by a sequential program that solves the same problem.

2 Program Invocation, Input, and Output

I'll call the executable program `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 requires 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 white-space characters such as spaces or tabs, 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
```

and 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 and white-space 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, if the first argument is not a positive integer, 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 the entire array, although it is free to make copies of small pieces of it as needed, using no more than $\Theta(NM)$ bytes of memory, where M is the pattern length and N is the number of threads.
5. The load should be balanced as uniformly as possible among the threads.
6. The program will have a critical section where two different threads try to write their redaction strings. It must prevent race conditions on this code.
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.

Assignment 3 described methods of getting the file size.

5 Program Grading Rubric

The program is worth 15% of the final grade and 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** (70 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 plates of arbitrary size.
  - The program should produce the same output no matter how many processes it is given, except for the elapsed time.
  - It should handle errors correctly.

- **Performance** (10 points)

  The program should be as efficient as possible. There are ways to solve this problem that are more efficient than others. When run with successively greater numbers of processes, the elapsed time should decrease, except that if the number gets too large (larger than the available processors generally), it will start to increase again. Check that this behavior occurs on average. On any one run, it may not be exactly like this, but over many runs it should behave like this.

- **Compliance with the Programming Rules** (20 points)

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

6 Submitting the Homework

1. Use the `submithwk_cs49365` program to submit your program. It can be run on any `cslab` host, so login to a `cslab` host when you're ready to submit. To submit, if your file is named `myhwk.c`, you'd enter

   ```
   $ cd ~
   $ submithwk_cs49365  -t 5  myhwk.c
   ```

   The program will try to copy `myhwk.c` into the directory

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

   and if it is successful, it will display the message, "`File hwk5_`*`username`*`.c successfully submitted.`"

   where *`username`* is your username. 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 entering the command

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

   and making sure you see a nonempty file named `hwk5_`*`username`*`.c` where *`username`* is your user name and whose date of last modification is the time at which you ran the command.

2. ***You can do step 1 as many times as you want. Newer versions of the file will overwrite older ones.***

7 Deadline

This assignment must be submitted by its ***deadline***, which is ***Monday, May 6, at 7:00 PM. After that, you will not receive credit for completing it.***