EECS3221E (Fall 2018) Assignment Two

You have to work individually. You are NOT allowed to view or exchange documents with your peers. We treat all sorts of cheating very seriously. Do not copy code from anywhere, even as a `template". As part of marking procedure, TA's always test your codes using a popular tool to detect any software plagiarism. No late submission will be accepted. You are required to program in your PRISM account. Your work will be graded based on: i) correctness of programming logic; ii) clarity of your code and your coding style; iii) whether your code follows the specification. It is your responsibility to explain clearly every detail you do in the code with appropriate comments to avoid any possible confusion in marking.

Your task is to write a C program that creates three groups of Pthreads, an IN group, a WORK group, an OUT group, to encrypt an input text file into a secret code or decrypt the secret code into the original text file according to a given KEY value.

The original main thread is not part of these three groups. The `main()` function should open the source file, and create/initialize a common buffer, and create all IN, WORK, OUT threads. Then, the main thread waits for all these threads to finish. All IN, WORK and OUT threads share the same buffer. Each slot in the common buffer stores 3 pieces of information: one data byte from the source file, its offset in the source file and a flag to indicate its status in the buffer, such as unprocessed, changing, processed, an so on.

```
typedef  struct {
     char  data ;
     off_t offset ;
     char state;
} BufferItem ;
```

Each IN thread goes to sleep (use `nanosleep`) for some random time between 0 and 0.01 seconds upon being created. Then, it reads the next single byte from the input file and saves that byte and its offset in the file to the next available empty slot in the buffer. Then, this IN threads goes to sleep (use `nanosleep`) for some random time between 0 and 0.01 seconds and then goes back to read the next byte of the file until the end of file. If the buffer is full, IN threads go to sleep (use `nanosleep`) for some random time between 0 and 0.01 seconds and then go back to check again.

Meanwhile, upon being created each WORK thread sleeps (use `nanosleep`) for some random time between 0 and 0.01 seconds and it reads next byte in the buffer and process one byte of data, either encrypts or decrypt according to the working mode. Then the WORK thread goes to sleep (use `nanosleep`) for some random time between 0 and 0.01 seconds and goes back to process next byte in the buffer until the entire file is done. If the buffer is empty, the WORK threads go to sleep (use `nanosleep`) for some random time between 0 and 0.01 seconds and then go back to check again. If running in the encrypt mode, each WORK thread will encrypt each data byte in the buffer, from original ASCII code to secret code for each character in the file, according to the following formula:

```
if(data>31 && data<127 )
      data = (((int)data-32)+2*95+KEY)%95+32 ;
```

If running in the decrypt mode, each WORK thread decrypts each data in the buffer, from secret code to original ASCII code, according to the following formula:

```
 if (data>31 && data<127 )
     data = (((int)data-32)+2*95-KEY)%95+32 ;
```

where KEY is a key value (between 0 and 127). If you use the same value for both encryption and decryption, it can perfectly recover the original data.

Similarly, upon being created, each OUT thread sleeps (use `nanosleep`) for some random time between 0 and 0.01 seconds and it reads a processed byte and its offset from the next available nonempty buffer slot, and then writes the byte to that offset in the target file. Then, it also goes to sleep (use `nanosleep`) for some random time between 0 and 0.01 seconds and goes back to copy next byte until nothing is left. If the buffer is empty, the OUT threads go to sleep (use `nanosleep`) for some random time between 0 and 0.01 seconds and then go back to check again.

Since all threads access common data, synchronization will be required. You may wish to look at the man pages for *pthread_create*, *pthread_mutex_init*, *sem_init* and other related pthread API's. Use critical sections of code. To achieve maximum concurrency, you should make your critical sections as small as possible. For example, IN threads should not have one big critical section where they do all of the following together: (a) read from the file; (b) write to the buffer. Instead, they should have 2 separate critical sections, one for each of (a) and (b). Similarly, WORK and OUT threads should have separate critical sections for reading the buffer and writing the copy.

The program should be called `encrypt.c` and will be compiled with:

```
gcc -Wall -o encrypt encrypt.c -lpthread
```

It will be invoked as follows:

```
encrypt <KEY> <nIN> <nWORK> <nOUT> <file_in> <file_out> <bufSize>
```

`<KEY>:` the key value used for encryption or decryption, and its valid value is from -127 to 127. If it is positive, WORK threads use `<KEY>` as the KEY value to encrypt each data byte. If it is negative, WORK threads use the absolute value of `<KEY>` as the key value to decrypt each data byte.
`<nIN>`: the number of IN threads to create. There should be at least 1.
`<nWORK>:` the number of WORK threads to create. There should be at least 1.
`<nOUT>`: the number of OUT threads to create. There should be at least 1.
`<file_in>`: the pathname of the file to be converted. It should exist and be readable.
`<file_out>:` the name to be given to the target file. If a file with that name already exists, it should be overwritten.
`<bufSize>:` the capacity, in terms of BufferItem's, of the shared buffer. This should be at least 1.

As a simple debugging strategy, if you use the same KEY value to run your `encrypt.c` to process any text file (e.g. test.txt) twice, one in encryption mode and one in decryption mode, you should be able to perfectly recover the original text file.

What to submit?

Submit the C program using the following command from your PRISM account:

```
submit 3221 a2 encrypt.c
```

Include the following information (please complete) as a comment at the beginning of your C program:

```
/*
Family Name:
Given Name:
Section:
Student Number:
CS Login:
```

```
*/
```

No hardcopy is needed for this assignment.