Project 2 Spring 2021

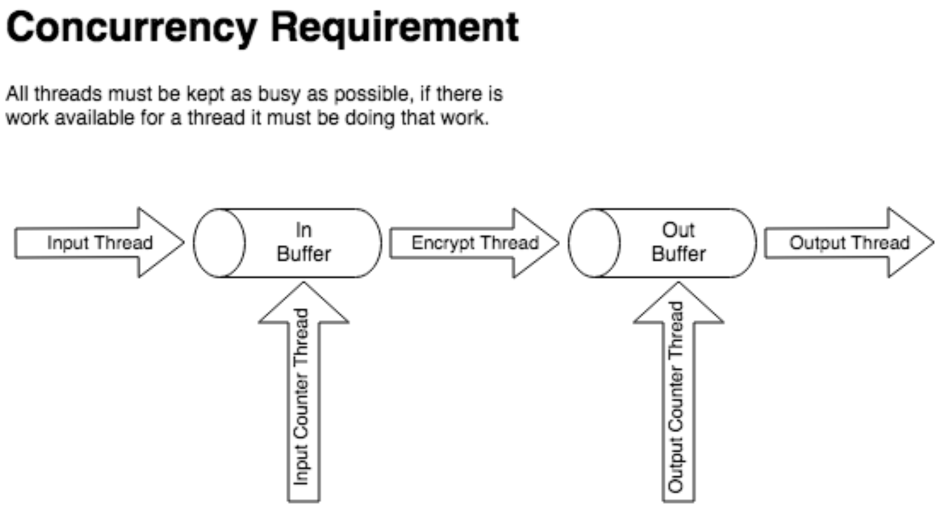
In this project you will synchronize multiple threads. The project must be written in C, and must use the Pthreads library. As you will see, some aspects of the project are not specified; it will be your responsibility to take whatever steps you feel are appropriate to complete the task.

Overview

The purpose of this project is to implement a multi-threaded text file encryptor. Conceptually, the function of the program is simple: to read characters from an input source, encrypt the text, and write the encrypted characters to an output. Also, the encryption program counts the number of occurrences of each letter in the input and output files. All I/O and encryption is performed by a module (encrypt-module.c) that is supplied by the project. Here is the single threaded version that uses the methods of the encrypt module.

```
init("in.txt", "out.txt");
while ((c = read_input()) != EOF) {
       count_input(c);
       c = caesar_encrypt(c);
       count_output(c);
       write_output(c);
}
printf("End of file reached.\n");
display_counts();
```

The major challenge of the project is that all processing must be performed in five concurrent threads and those threads must be kept as busy as possible. The threads are a reader, input counter, encryptor, output counter and writer. Since multiple threads will be accessing the same data structures, you will need to synchronize the threads to avoid race conditions (therein lies the difficulty).



Synchronization **must** be done via Pthreads constructs such as semaphores, condition variables, and/or mutexes (i.e., your code cannot contain spinlocks). For more information, consult the man pages for the following.

- `pthread_create`
- `pthread_join`
- `sem_init`
- `sem_wait`
- `sem_post`
- `sem_destroy`
- `pthread_mutex_init`
- `pthread_mutex_lock`
- `pthread_mutex_unlock`
- `pthread_mutex_destroy`
- `pthread_cond_init`
- `pthread_cond_wait`
- `pthread_cond_signal`
- `pthread_cond_destroy`

You have been provided with the files encrypt-module.h and enrypt-module.c which implement all of the file I/O, encryption and counting operations you will need. You **must** use these functions to perform I/O, encryption and counting. You should not make any assumptions about the timing of these functions. The init( ) function must be called in main( ). Portions of this project are graded using a testing framework, failure to follow these rules may result in a very low score. Here are the functions.

- `init - must be called from main()`
- `read_input - get the next input`
- `write_output - write an encrypted character`
- `caesar_encrypt - encrypt a character`
- `count_input - count an input (plaintext) character`
- `count_output - count an output (cyphertext) character`
- `get_input_count - get the input count`
- `get_output_count - get the output count`
- `reset_requested - this one you implement`
- `reset_finished - this one you implement`

Required Features

5 points: makefile

You get 5 points for simply using a makefile. Name your source files whatever you like. Please make the name of your executable be **encrypt352**. Be sure that "make" will build your executable on pyrite.

10 points: Documentation

If you have more than one source file, then you must submit a **Readme** file containing a brief explanation of the functionality of each source file. Each source file must also be well-documented. There should be a paragraph or so at the beginning of each source file describing the code; functions and data structures should also be explained. Basically, another programmer should be able to understand your code from the comments.

35 points: Main Thread

Main does the following.

1. Obtain the input and output files as **command line arguments**. If the number of command line arguments is incorrect, exit after displaying a message about correct usage.
2. Call init( ) with the input and output file names.
3. Prompt the user for the input and output buffer sizes *N* and *M*. The buffers may be any size >1.
4. Initialize shared variables. This includes allocating appropriate data structures for the input and output buffers. You may use any data structure capable of holding exactly *N* and *M* characters for the input and output buffers respectively.
5. Create the other threads.
6. Wait for all threads to complete.
7. Display the number of occurrences of each letter in the input and output files.

10 points: Reader Thread

The reader thread is responsible for reading from the input file one character at a time, and placing the characters in the input buffer. It must do so by calling the provided function `read_input()`. Each buffer item corresponds to a character. Note that the reader thread may need to block until other threads have consumed data from the input buffer. Specifically, a character in the input buffer cannot be overwritten until the encryptor thread and the input counter thread have processed the character. The reader continues until the end of the input file is reached.

5 points: Input Counter Thread

The input counter thread simply counts occurrences of each letter in the input file by looking at each character in the input buffer. It must call the provided function `count_input()`. Of course, the input counter thread will need to block if no characters are available in the input buffer.

10 points: Encryption Thread

The encryption thread consumes one character at a time from the input buffer, encrypts it, and places it in the output buffer. It must do so by calling the provided function `caesar_encrypt()`. Of course, the encryption thread may need to wait for an item to become available in the input buffer, and for a slot to become available in the output buffer. Note that a character in the output buffer cannot be overwritten until the writer thread and the output counter thread have processed the character. The encryption thread continues until all characters of the input file have been encrypted.

5 points: Output Counter Thread

The output counter thread simply counts occurrences of each letter in the output file by looking at each character in the output buffer. It must call the provided function `count_output()`. Of course, the output counter thread will need to block if no characters are available in the output buffer.

5 points: Writer Thread

The writer thread is responsible for writing the encrypted characters in the output buffer to the output file. It must do so by calling the provided function `write_output()`. Note that the writer may need to block until an encrypted character is available in the buffer. The writer continues until it has written the last encrypted character. (Hint: the special character EOF will never appear in the middle of an input file.)

15 points: Encryption Module Reset

For security reasons, the encryption module occasionally needs to reset itself with a new encryption key. When this happens, the input and output counts are reset to zero. To reduce predictability, the module may decide to perform a reset at any time. Before it performs a reset, it will inform your application by calling `reset_requested()`, do not return from `reset_requested()` until it is safe for the reset to occur. After a reset is complete it will call `reset_finished()`. Use these function calls to print the input and output counts before the reset and to ensure the correct behavior of the system during and after the reset. Only the characters encrypted using a particular key should be counted together and there should be no characters that go uncounted. You may not simply blocking `reset_requested()` until the entire file is read.

Synchronization Requirement

Your program should achieve maximum concurrency. That is, you should allow different threads to operate on different buffer slots concurrently. For example, when the reader thread is placing a character in slot 5 of the input buffer, the encryption thread may process the character in slot 3 of the input buffer and the input counter thread may process the character in slot 2 of the input buffer.

Your program **MUST NOT** do the following: first let the reader thread put *N* characters in the input buffer, and then let the input counter thread and the encryption thread consume all the characters in the input buffer. This does not provide maximum concurrency.

Example

Here are some example input files and their corresponding output files.

```
> encrypt infile outfile
Enter input buffer size: 10
Enter output buffer size: 10

Reset requested.
Total input count with current key is 488.
A:5 B:1 C:24 D:17 E:29 F:7 G:1 H:12 I:33 J:0 K:1 L:21 M:6 N:33 O:21 P:14 Q:0 R:4 S:20 T:46
U:37 V:1 W:0 X:0 Y:6 Z:2
Total output count with current key is 488.
A:2 B:5 C:1 D:24 E:17 F:29 G:7 H:1 I:12 J:33 K:0 L:1 M:21 N:6 O:33 P:21 Q:14 R:0 S:4 T:20
U:46 V:37 W:1 X:0 Y:0 Z:6
Reset finished.

Reset requested.
Total input count with current key is 592.
A:21 B:0 C:15 D:15 E:44 F:14 G:1 H:6 I:31 J:0 K:1 L:18 M:7 N:31 O:18 P:22 Q:1 R:23 S:12
T:46 U:27 V:3 W:3 X:0 Y:2 Z:0
Total output count with current key is 592.
A:31 B:18 C:22 D:1 E:23 F:12 G:46 H:27 I:3 J:3 K:0 L:2 M:0 N:21 O:0 P:15 Q:15 R:44 S:14
T:1 U:6 V:31 W:0 X:1 Y:18 Z:7
Reset finished.

Reset requested.
Total input count with current key is 509.
A:6 B:0 C:31 D:2 E:14 F:3 G:3 H:0 I:18 J:0 K:2 L:4 M:0 N:27 O:25 P:19 Q:0 R:10 S:5 T:50
U:34 V:2 W:0 X:0 Y:2 Z:6
Total output count with current key is 509.
A:14 B:3 C:3 D:0 E:18 F:0 G:2 H:4 I:0 J:27 K:25 L:19 M:0 N:10 O:5 P:50 Q:34 R:2 S:0 T:0 U:2
V:6 W:6 X:0 Y:31 Z:2
Reset finished.

End of file reached.
Total input count with current key is 96.
A:3 B:0 C:3 D:0 E:3 F:0 G:1 H:0 I:2 J:0 K:0 L:3 M:0 N:7 O:8 P:3 Q:0 R:4 S:0 T:18 U:10 V:0
W:0 X:0 Y:0 Z:0
Total output count with current key is 96.
A:0 B:0 C:0 D:0 E:3 F:0 G:3 H:0 I:3 J:0 K:1 L:0 M:2 N:0 O:0 P:3 Q:0 R:7 S:8 T:3 U:0 V:4 W:0
X:18 Y:10 Z:0
```

Submitting Your Project

You will submit your project on Canvas. Your program must compile and run without errors on pyrite.cs.iastate.edu.

Put all your source files (including the makefile and the README file) in a folder. Then use command `zip -r <your ISU Net-ID> <src_folder>` to create a .zip file. For example, if your Net-ID is ksmith and project2 is the name of the folder that contains all your source files, then you will type `zip -r ksmith project2` to create a file named ksmith.zip and then submit this file.