EECS3221 Assignment A2

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". No late submission will be accepted. All assignment work must be done on the EECS `red` *server. 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 specifications. 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 two groups of pthreads, an `IN` group and an `OUT` group, to create an exact copy of a source file passed as a command-line argument.

The original main thread is not part of either group. The `main()` function should open the source file, and create/initialize a circular buffer, and create all `IN` and `OUT` threads. Then, the main thread waits for all these threads to finish their work. All `IN` and `OUT` threads share the circular buffer. Each buffer slot stores `2` pieces of information: one data byte read from the source file and its offset in the source file.

```
typedef  struct {
     char  data ;
     off_t offset ;
} 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 file and saves that byte and its offset in the file into the next available empty slot in the circular buffer. Then, this `IN` thread goes to sleep (use `nanosleep`) for some random time between `0` and `0.01` seconds and then goes back to read the next byte, until the end of file. Try inserting a `nanosleep` between the file reading and the buffer writing parts of the code and compare the resulting logs.

Similarly, upon being created, each OUT thread sleeps (use `nanosleep`) for some random time between 0 and 0.01 seconds and then reads a 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 then goes back to copy the next byte until nothing is left. Try inserting a `nanosleep` between the buffer reading and the file writing parts of the code and compare the resulting logs.

Before finishing the main thread can either i) sleep for a few seconds to give time to the other threads to finish their work (not guaranteed), or ii) use `pthread_join` and other

methods to make sure that all threads have finished their work (more difficult but may earn you some extra credit).

Along the way, each thread writes some information to a log file, so that the execution of the program can be traced.

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.

Pay attention to the critical sections of code. Structure your code in such a way that no thread will spend an extensive period of time in the critical section, i.e. you should make your critical sections as small as possible. For example, an `IN` thread should not have one critical section, where it does all of the following together: (a) read from the file, (b) write to the log file, (c) write to the buffer; (d) write to the log file. Organize critical sections in a way that avoids unnecessary blocking but guarantees proper order of the records in the log file. For instance, make sure that since a thread reads from a file no other record should be allowed in the log file until the thread actually writes to the log file. Similarly, writer threads should have separate critical sections and their reading from the buffer and writing to the file should be timely recorded in the log file.

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

```
cc -Wall -o cpy copy.c -lpthread
```

It will be invoked as follows:

```
./cpy <nIN> <nOUT> <file> <copy> <bufSize> <Log>
```

`<nIN>` is the number of IN threads to create. There should be at least `1`.
`<nOUT>` is the number of OUT threads to create. There should be at least `1`.
`<file>` is the pathname of the file to be copied. It should exist and be readable.
`<copy>` is the name to be given to the copy. If a file with that name already exists, it should be overwritten.
`<bufSize>` is the capacity, in terms of buffer slots, of the shared buffer. This should be at least `1`.
`<Log>` the threads write some trace information to this file. If a file with that name already exists, it should be overwritten.

For you reference, we have posted on the course site a file with some hints `(a2-hints.pdf)`, a copy file `(dataset4.txt)`, and a log file `(dataset4log.txt)`. However, your program should be able to process any copy file.

The log file

The log file creates a trace of the execution of your program.

Each of the `IN` threads should be given a different number in the range `0` … `<nIN>-1`. Each of the `OUT` threads should be given a different number in the range `0` … `<nOUT>-1`. Each thread should know its own number. (This number is different from the thread id.)

When an IN thread reads the next unread byte from the file, it can obtain the offset using the `lseek` system call. When the `IN` thread saves the byte and its offset to the buffer, it writes to a particular index in the buffer. Each time an `IN` thread number *n* reads `a` byte *b* (print it as an integer between `0` and `255`) from offset ***x*** in the file, it should write to the log file the following line:

`read_byte PT`*n* `O`*x* `B`*b* `I-1`

In general, output format should be as follows:

`<operation> <thread_type><n> O<offset> B<b> I<i>\n`

Where:

- `<operation>` is one of `read_byte`, `write_byte`, `produce` or `consume` followed by a single blank,
- `<therad_type>` is either `PT` (Producer/IN Thread) or `CT` (Consumer/OUT Thread)
- `<n>` is the thread number
- `<offset>` is the offset in the file
- `<b>` is the actual byte that is an integer between `0` and `255`
- `<i>` is ether `-1` (for `read_byte` and `write_byte` operations) or the actual index in the buffer (for `produce` and `consume` operations)

Consequently when the `IN` thread number *n* writes the byte *b* and its offset *x* in the file to index *i* in the buffer, it should write to the log file the following line:

`produce PT`*n* `Ox B`*b* `I`*i*

Similarly, each `OUT` thread writes to the log a `consume` line after reading an item from the buffer and a `write_byte` line after writing to the target file as follows:

`consume CT`*n* `O`*x* `B`*b* `I`*i*
`write_byte CT`*n* `O`*x* `B`*b* `I-1`

Examples:

```
read_byte PT0 O0 B47 I-1
produce PT0 O0 B47 I0
consume CT1 O0 B47 I0
read_byte PT1 O1 B42 I-1
produce PT1 O1 B42 I1
write_byte CT1 O0 B47 I-1
```

Do not deviate from this format and do not add headings, summary information, or user prompts as this may interfere with the proper examination of the log.

Make sure that you include the following information (please complete) as a comment in the beginning of your C programs:

/
Family Name:
Given Name:
Student Number:
EECS Login ID (the one you use to access the `red` *server):*
YorkU email address (the one that appears in eClass):
**/*

Finally prepare your report as a PDF file named `reportA2.pdf`. Your report should be structured as follows:

1. A general outline of your understanding of the assigned work.
2. A clear statement about the assigned work/components you believe you have done/completed successfully.
3. A statement about the work you believe you might have not completed successfully (feel free to comment on related problems, if any).
4. Anything else related to your work that you might wish to comment upon.

Please copy the above entries and paste into your report to use as a template when typing your report.

What to submit

Submit the following files:

```
reportA2.pdf
copy.c
```