EEL 4930/5934 Advanced Systems Programming

Assignment 2

Single thread version due: Wednesday, January 28th by midnight

Multithreaded version due: Monday, February 2nd by midnight

In this assignment you are going to write two (one single threaded and one multithreaded) versions of a program that gets five command line arguments (in order): `instructionFileName`, `inputDataFileName`, `outputDataFilename`, `blockSize`, and `bufferSize`. The idea is to copy `inputDataFileName` to `outputDataFilename` by applying the transformations specified in file `instructionFileName`.

`instructionFileName` contains a sequence of actions. Each line has a separate action. An action can be a `revert` or a `zero` action.

- `revert ni no b0 b1 ... bm`: Reverses the bits b0, b1, ..., bm, where $0 \leq m \leq 7$, on every byte of the most recent copy of the nith block from `inputDataFileName` and writes the processed block to `outputDataFilename` as the noth block.
- `zero ni no b0 b1 ... bm`: Zeros bits b0, b1, ..., bm, where $0 \leq m \leq 7$, on every byte of the most recent copy of the block ni from `inputDataFileName` and writes the processed block to `outputDataFilename` at the noth block.

Please note that if a block in `inputDataFileName` has not been read yet, i.e., no previous action has referred to it, then the most recent copy refers to the copy on disk and needs to be read from the file into the memory. However, if it already has been read to perform a previous action then it has to be the copy resulting from the most recent action performed on it.

The actions that are dependent on each other are to be executed according to the specification order. For instance, for the following instruction sequence

```
revert 5 1 2
revert 4 9 3
zero 5 7 0
zero 5 9 1 5 6
revert 4 12 0
```

`zero 5 7 0` cannot be executed before `revert 5 1 2` because `zero 5 7 0` needs the processed block resulting from `revert 5 1 2`. Also, `zero 5 9 1`

`5 6` cannot be executed before `revert 4 9 3` as they both write to block 9 of `outputDataFilename` and when the program terminates we should have on block 9 the copy that has been generated by `zero 5 9 1 5 6`. Please note that although `zero 5 9 1 5 6` overwrites the block written by `revert 4 9 3`, `revert 4 9 3` still needs to be performed as `revert 4 12 0` is based on the processed version of block 4 generated by that action. So your program should figure out the dependencies based on the input and output block no's of the actions to reflect all the transformations at the same time providing sequential consistency. As a special case if file `instructionFileName` is empty then `outputDataFilename` should be an exact copy of file `inputDataFileName`.

The multithreaded version of your program should have three threads in addition to the main thread. The responsibilities of the threads are as follows:

- **Main thread:** Reads in `instructionFileName`, initializes shared data, creates the other threads, and waits until all the threads to terminate.
- **Reader thread:** Performs all reading from file `inputDataFileName`.
- **Writer thread:** Performs all writing to file `outputDataFilename`.
- **Processing thread:** Performs the `revert` and `zero` operations.

You are expected to use Pthreads library to create the threads and ensure the necessary synchronization. The goal is to give each thread as much data as possible to keep it busy and minimize wait time due to waiting for data to arrive by another thread. You should make sure that threads gets blocked when the data is not available instead of busy waiting. To get full credit, your multithreaded solution should be free of deadlocks and race conditions.

Your program should use `blockSize` as the block size and should create buffers of size `bufferSize` to keep at maximum `bufferSize` of read/processed blocks in your data structures.

Please note that we have two deadlines for this assignment: Wednesday, January 28th for the single thread version and Monday, February 2nd for the multithreaded version. Each version weighs 50% of the total points for the assignment. Please submit the source code(s) and the Makefile for your solutions on E-learning CANVAS.