Assignment 2

EEL4732/5733 Advanced Systems Programming

Due: Monday, February 10th by midnight

In this assignment, you are going to implement the Combiner program from Assignment 1 using multiple threads instead of multiple processes. One difference from Assignment 1 is that the tuples may not be sorted according to user ids. Your program will use the Pthreads library to create one thread for the Mapper and multiple threads for the Reducer, where each Reducer thread handles tuples related to a unique user.

Note that this is an instance of the Producer Consumer problem, where the Mapper thread is the producer and the Reducer threads are the consumers. You will implement a dynamic data structure to represent the buffer that enables communication between the mapper and a reducer. The buffer will have a certain number of slots and each slot in the buffer can hold exactly one tuple. The size of the buffer will be determined by the command line parameter. There will be as many buffers as the number of Reducer threads. You should assume that the Mapper thread will be reading the input from the standard input. So, as an example, your multithreaded Combiner program will be executed as follows:

```
$ ./combiner 10  5 < inputFile
```

where 10 denotes the number of slots in each communication buffer, 5 denotes the number of users (reducer threads), and inputFile contains the tuples with user actions on specific topics as it was in Assignment 1. Your code does not need to double check to see if the given number of reducers is equal to the number of users in the inputFile.

Please make sure that the Mapper thread waits if there are no available slots in the buffer and the Reducer thread waits if there are no items in the buffer. Also, once the Mapper thread processes all the tuples (enters them in the communication buffers), it should let the Reducer threads know that the producer will not be providing any more data. **To get full credit, your solution should be free of race conditions and deadlocks and produce the correct output in every run of your code.** You can use mutexes and condition variables and/or semaphores for synchronizing your threads. Please submit your files (source, README, and a Makefile) on CANVAS by the due date.