Multithreaded Book Order System

Alexander Tang, Craig Perkins

11-25-13

Input:

Our executable file is named `bookorder`. To run our program, type the following into the command line:

```
./bookorder <database file> <book order file> <categories>
```

The `<database file>` and `<book order file>` both use the format specified by the given "database.txt" and "orders.txt" file, respectively. Note: in the `<database file>` and `<book order file>`, there should not be whitespace after the "|" delimiter. `<categories>` takes either a list of categories names (i.e. alphanumeric strings separated by blanks in a single- or double-quoted string) or a file which contains a category on each line as in the given "categories.txt" file.

Implementation:

To simulate a database of customers, we use a hashtable called "customers" with customer ids as keys and Customer structures as values to store as our database. The Customer structures each contain a field for name, id, address, state, zipcode, balance, successful orders, and rejected orders. To ensure consistency of our database, we use a mutex on our customers hashtable.

Our implementation of the book order system also makes use of pthreads. In total, we have one producer thread and n consumer threads, where n is the number of given book categories. Each consumer thread corresponds to one book category.

For our buffers, we used multiple buffers and implemented a thread-safe queue structure located in "queue.h" and "queue.c". Each queue contains its own mutex, which it locks whenever it calls its enqueue and dequeue functions. Each queue corresponds to one category, and we referred to these queues as "category queues", for which there are n of them.

Our producer thread reads in from the `<book order file>` and produces book orders which are placed in the corresponding category queue. To do this efficiently, we used a hash table to map category strings (keys) to category queues (values). While the producer is producing orders, each consumer thread waits on its corresponding category queue. If a consumer thread sees that its category queue is not empty, it attempts to process the order by dequeueing an order (which is thread safe because the queue contains its own mutex) and acquiring the mutex for the customers hashtable. If it acquires the mutex for the customers hashtable, depending on whether the customer's balance is sufficient, it either adds the order to the customer's list of successful or rejected orders. If the order is successful, the customer's balance is reduced by the book order's price. The consumer threads also print an Order Summary of each transaction to `stdout` as they occur. After the producer finishes producing book orders and the consumers finish processing their queues, the program prints out each customer's history to `stdout` and exits.