Producer Consumer Bookstore Project

David DeSimone

November 30, 2014

1 Program Description

The program implements a producer/consumer system which models a bookstore. A customer database is created, and orders are read. Each customer has a credit allowance, while each book has a given cost. If a customer cannot afford a given book, their order is rejected. Each book has a category. Each category is processed by a different thread (The consumers). Once all orders have been successfully processed, the program will print a list of results to **output.txt**.

2 Implementation

This program features a single producer, and several consumers, one for each category of book. Each producer/consumer is run in a separate thread. Each consumer has a **thread-safe bounded queue** which it reads data from. When the producer reads a line from the order file, it 1) looks at the category field 2) searches for the category in the master category list 3) finds the relevant consumer for that category and 4) if the consumers bounded queue is not full, adds the order to the consumers process queue and alerts the consumer that data has been added. If the consumers queue is full, the producer will wait until the consumer alerts him of room in it's respective queue.

The consumer listens for the producer to alert it of data. When the consumer is alerted of new data, it attempts to dequeue an order from it's queue. It will 1) search for the relevant customer is the master customer list 2) if the customer exists, check to see if the customer has the funds to complete the purchase and 3) calculate the record of this transaction, and add it to the customer's list of transactions. Each customer maintains a thread safe vector of strings representing it's accepted orders, and it's rejected orders.

When the producer has finished reading the book orders, it will flag each of the consumers with a finished flag. If the consumers detect this flag it set, they will terminate after they process the entries in their current queues. After the producer joins with each of the finished consumers, it will calculate the total sales, and print the output file to output.txt

3 Data Structures

To assure thread safety, several thread-safe data structures had to be implemented. These include a thread-safe queue, a thread safe array-list (vector in c++ terms) for strings and customer objects. To assure thread-saftey, all data member's of the relevant structs are accessed via getters and setters protected by mutexs. The thread-safe queue was implemented with a circular link-list.

4 Efficiency

In terms of asymptotic running time, this program runs in O(n * (m + k)), where n is the number of orders, m is the number of people, and k is the number of categories. Each order causes a linear lookup for the consumer for the given category, and for the given customer in the master customer list. For each of the n orders, we preform these lookups, and thus O(n * (m + k)). In terms of memory, book orders are released as they are processed. The entire order list is NOT necessarily loaded into memory at once, and thus a large number of orders could be processed by this program in a space where memory is limited.