Anna Genke & Zack Colello

Assignment 5: Multithreaded Book Order System

Our Multithreaded Book Order System includes 4 files: bookorder.c, bookorder.h, queue.c, and queue.h. We've included a makefile that creates an executable called bookorder. The syntax for bookorder is as follows:

./bookorder <database file> <orders file> <categories file>

where database file is a text file of a user database, orders file is a text file of orders to be handled, and categories file is a text file of different categories.

Thread Synchronization

Our program spawns a producer thread from main that traverses the orders file line by line. The Producer then creates a node called ordernode to hold the order information. Producer attempts to place the ordernode in the corresponding category queue. If the queue contains five orders already in the queue, the Producer waits for that category queue's Consumer to dequeue an ordernode. When a space signal is given to Producer from a Consumer, Producer places the order node in the queue.

In our **bookorder** program, we create a Consumer thread for each category, which has exclusive access to only that one category. The Consumer threads check their corresponding category queues to see if there are any order nodes placed into the queue by Producer that need dequeuing. If there is an order node, the Consumer thread will look at that order node's customer ID and find it in the database structure. If that customer has enough funds for that order, it will subtract that book's price from the customer's current balance, and add it to that customer's

success queue. Otherwise, if the customer does not have enough funds, it places it in the customer's **failure** queue.

The Producer and Consumer threads both work synchronously, using mutex locks and pthread_cond_wait calls to ensure threads are not changing/accessing the same queue or database at the same time.

Memory Requirements

Our **bookorder** program reads in the user files in O(*n*) time. We then create an array of structures called **customerDatabase** in O(*n*) time that we use to store each customer's information that has been read in from the database file. We create an array of queues to hold one queue per category given in the category file in O(*n*) time. We then create our Producer and Consumer threads (ensuring they have successfully been created).

Our Producer thread traverses the orders text file in O(*n*) time, creating an order node for each line. Producer traverses the array of queues to find a matching category. If a matching category is found, we use a mutex lock and enter a while loop that will wait until space is freed by a consumer (category queues can hold up to 5 order nodes). After space is available, we enqueue the new order node and unlock the mutex. If the category is not found, we print an error. Overall, Producer runs in O(n*l) time, where n is our number of orders, and l is the number of categories.

Our Consumer threads entire a while loop that repeats as long as there are orders to be taken from producer. We use a mutex lock when we know there is an order, and another while loop that waits until it gets an order. If the order received is not null, the Consumer thread finds the matching id number in O(*n*) time in the database structure. If found, Consumer checks the customer's balance, and places it into the appropriate queue (failure queue or success queue). Then, consumer unlocks the mutex.

Back in main, we've waited for the Producer and various Consumer threads to finish, and then we traverse the database in O(n) time to print out the customer's information, as well as their successful and failure queues in O(n) time for each customer.

Finally, we free the queues, database, and nodes we've created to prevent any memory leaks, and exit our program.