./thread

Program Usage:

./thread <database file> <book order file> <categories file>

Program Description:

- Simulates a multithreaded book ordering system given a database file, orders file and categories file.

Program Overview:

- The book ordering system starts off by calling the startup() function which reads the database file and categories file, then creates a database that maps customerID to customer info and a database that maps category names to initialized buffers, respectively. Setup() is also responsible for initializing our global variables which are shared across all threads.
- For each category, a buffer hash is initialized to store the orders for that category.
- A producer thread is created and detached to process the orders given by the orders file. The producer continuously traverses the file, creates orders, and adds them to the buffer according to the category of the book. It then signals that consumer that there is an order available. It continues to add orders to the buffer until the buffer is full, or there are no more orders to read from the orders file (EOF is reached). The producer then sets a flag that it is finished keeps signaling all the threads that it is done. This is done so that consumers know that if a flag is set and the buffer is empty it is safe for them to exit. This is also done in case there are any waiting consumers (deadlock); they are safely exited.
- Multiple consumer threads are created and detached, one for each category given from categories.txt. They continuously processes orders and keep track of accepted and rejected sales by adding them to two global sorted lists, one for accepted orders and one for rejected orders. The lists are sorted by customerID, since the final report is printed in that order. A mutex is kept for each structure, both lists and the customer database are locked with their respective mutexes when a category is accessing it.
- In order to avoid a race condition (or a deadlock), both main and the producer wait until all consumers are done processing so that it is safe for them to exit. They simply

check a protected global variable that keeps track of how many consumers are done and compare it against the number of categories. When the consumers are done main then checks the global sorted lists and prints out the report accordingly.

Thread Management and Data Safety:

- The producer and consumer threads share a bounded buffer with 10 slots, there are variables that keep track of when the buffer is full or empty, and the consumer and producer are signaled (using pthread_cond_*), respectively.
- All shared data is protected by variables of type pthread_mutex_t, which makes the data available for one thread and unavailable to all other threads until the owner is done.
- The producer and consumers communicate through conditional flags (and pthread_cond_* structures) producerEndCond and consumerCountCond. Initially, the producer adds orders into the buffer until the buffer is full.
- ProducerFinished communicates to the consumers that the producer has finished reading the file, and so they can exit when they are done.
- addNewOrder() – the producer - calls pthread_cond_wait() if a buffer is full and processOrder() – any of the consumers - calls pthread_cond_wait() if the buffer is empty.
- To prevent deadlocks, for each pair of mutexes, each thread simultaneously locks them in the same order. We also didn't use any continue or break statements inside a mutex lock, and always unlocked the mutex if a thread was about to exit and owned a mutex.
- If an error occurred the cleanup() function was called before any of the exit statements so that no memory was leaked.

Data Structures:

- A hashmap was used to keep track of the customers, it maps customerID to customer info (name, address, balance, etc..)
- A hashmap was used to keep track of the categories, it maps category name to an order buffer struct. Each one kept track of its size, mutexes etc.. so that only the producer and the consumer for that category had access to it. This minimizes deadlocks since we kept access to these variables to a minimum.
- Linked lists were used to store sales by customerid numerical order (one list for accepted sales and one for rejected sales).

Outside Contributions:

- UTHash – Hash table for C structures
- Author: Troy Hanson
  Source: https://github.com/troydhanson/uthash

Authors:

- Mauricio Trajano
- Paul McNeil