CS 428/528 Lab 2: Concurrent Server Fun

March 22 7:59pm, other than Part 1 which is due March 13

Overview and Description

In this project, we will implement a simulation of your startup GreenTomatoes, best known for multi-colored ketchup. You will write one program to simulate the sales server of GT and another to generate orders to it.

The server will have two UDP sockets from which it could receive purchase orders, one for retail customers, and another from bulk customers like restaurants and grocery shops. You dont know when each will receive orders. Retail customers pay $8 per case, while bulk customers pay $6. The cost of a case to you is $5. The individual customer orders are for a number of cases uniformly distributed (i.e., use a random number generator to get the order size) between 1 and 20. The bulk order sizes are uniformly distributed between 10 and 100. A transaction is initiated by sending a message to the sales socket, including the number of cases to be bought.

Assume that your intital inventory of ketchup cases is 400 cases (that is the size of your warehouse). Every time you receive an order, decrement your inventory. It takes 50msecs to process individual orders, and 100 msecs to process bulk orders; after waiting this time, you can and send a message back to the buyer delivering the cases. If not enough inventory is available, you have to wait for more inventory to arrive. When the inventory gets low, you should order more cases (remember that the cost of a case is $5). It should take 5 seconds between the time that you order the extra cases and the time that they are delivered. Since reordering takes time, typically we reorder when we run into an order we cannot fullfil.

GreenTomatoes has a delivery guarantee; if it takes more than 250msecs to deliver the ketchup back they will refund 50% of the order value. If it takes more than 500 msec, the ketchup is 90% off. The delay is measured at the buyer from when they send the order, until they receive the reply that it has been processed (assume they dont lie). The buyer will reply confirming the receipt of the cases and possibly making a claim for a refund. If there is a refund claim, you have to deduct that money from your total.

Each buyer can continuously send orders in a loop. One condition is that a buyer can have only one open order at a time (stop-and-wait). Moreover, after an open order is fulfilled, the buyer takes (100 +5(order size)) milliseconds to savor the ketchup (or to sell it in the case of bulk orders). If the buyer is multi-threaded, then these conditions apply to each thread separately without coordination with the other threads.

Note that you will have to design the message formats for the messages exchanged between buyers and sellers.

Implementation 1–Due 10/17, 9pm

Implement the server as a sequential programs. Implement the buyer as a two threaded process, one generating orders for bulk and one generating orders for individual ketchup cases. Help with multi-threaded programming will provided this week.

Run your implementation and collect the following information: (1) The value of the company after 1 minute of operation measured starting from the first order (cash on hand+ value of ketchup in stock at $5 a case). If you have the value for more than one minute, prorate it; (2) number of orders and total number of cases sold for bulk and for individual; and (3) percentage of orders late (50% refund) and very late (90% refund).

Do you think that your company will stay afloat based on the numbers you collected above?

Implementation 2 – Use select

Use the `select()` system call to poll the sockets before receiving on them. This allows you to avoid receiving on an empty socket and getting blocked there, potentially missing an order on the other socket. The server implementation remains sequential. Collect the same information as requested in the first implementation above. Help with select will be provided in a help session this week.

Implementation 3 – Multi-threaded server

Reimplement the server using two threads, one for each socket. Note that the two servers must be careful not to access shared variables such as the inventory and the total cash at the same time (you need to use a lock or some other synchronization mechanism to organize that). Collect the same information as requested in the first implementation above.

Bonus – Make the most money!

In this part, we relax most of the requirements on the implementation (but the rules of the behavior of buyers and seller remain the same), but apply on each thread individually. You can for example increase the number of buyer and/or server threads beyond two. Its all up to you.

The owner of the most successful company (that makes the most money while living up to all the design requirements), will receive a 3% bonus to their overall grade in the class. Two runner ups will receive 2% and 1% each. For purposes of the competition, fix individual order sizes to 5 and bulk order sizes to 50.

Some ideas on what to do:

- You can increase the number of buyer sockets, but you may only have two server sockets. If you have multiple buyers at the same time, you have to be careful not to mix up their orders.
- Make the re-order happen speculatively when you run low instead of waiting until you run out. Make that the responsibility of a separate thread that re-orders when low and restocks instead of having to wait for the restock shipment to arrive. Alternatively, you can use the alarm signal to avoid the wait. When reordering speculatively, if you order more than you have room to store, the extra cases will have to be thrown away.
- Even though you have only two sockets for the server side, you may want to have more than two threads to handle them. Be careful that having too many threads may introduce lock contention.

Help

I will provide help material and code snippets under the resources tab on the class website to help with different aspects of the assignment. Some tips and pointers for now:

- Use `nanosleep()`, `usleep()` or something similar (google for help on them) to implement the various waits.
- Use `setitimer()` or `alarm()` if you want to use the hardware timer (you shouldnt need to).
- There is an example of using `select()` in the sockets tutorial on the class homepage.
- For timing help, lookup the section in the previous lab document on the different options to time operation.
- We will be using the pthreads (POSIX threads) library for the multi-threaded implementation. Some help on using that and example code will be provided shortly.