CS 351: Design of Large Programs
Project 5: Distributed Auction
Due: May 11th
Worth 100 pts

April 18, 2023

In this program, you will be simulating a system of multiple auction houses selling items, multiple agents buying items, and a bank to keep track of everyone's funds.

The bank will exist on one machine at a static known address, the agents and auction houses will be dynamically created on other machines.

Bank

The bank is static and at a known address. You'll start this program before either agents or auction houses. (The bank is a server and the agents and auction houses are its clients.)

Both agents and auction houses will have bank accounts. When an agent bids on or or is outbid in an auction, the bank will block or unblock the appropriate amount of funds, at the request of the auction house. When an agent wins an auction, the bank will transfer these blocked funds from the agent to the auction house account, at the request of the agent.

Auction houses provide the bank with their host and port information. The bank provides the agents with the list of the auction houses and their addresses so the agents will be able to connect directly to the auction houses.

Aside from possibly some initial configuration, the bank is not expected to interact with the user, though you may chose to have some status messages printed to verify what is happening.

You may assume the bank program will remain running throughout the simulation. You don't have to make agents and auction houses robust to the bank program terminating.

Auction House

Each auction house is dynamically created. Upon creation, it registers with the bank, opening an account with zero balance. It also provides the bank with its host and port address[1], so the bank can inform the agents of the existence of this auction house. (An auction house is a client of the bank, but also is a server with agents as its clients.)

It hosts a list of items being auctioned and tracks the current bidding status of each item. Initially, the auction house will offer at least 3 items for sale.[2] As the items are sold, new items will be listed to replace them. (The items for sale may be scripted, read in from a configuration file, programmatically generated, etc.)

Upon request, it shares the list of items being auctioned and the bidding status with agents, including for each item house id, item id, description, minimum bid and current bid.

The user may terminate the program when no bidding activity is in progress. The program should not allow exit when there are still bids to be resolved. At termination, it de-registers with the bank. An auction house terminating should not break the behavior of any other programs in the system.

Aside from possibly some initial configuration and requesting the program to exit when safe to do so, an auction house is not expected to interact with the user, though you may chose to have some status messages printed to verify what is happening.

Auction Rules

The auction house receives bids and acknowledges them with a reject or accept response.

When a bid is accepted, the bank is requested to block those funds. In fact, the bid should not be accepted if there are not enough available funds in the bank.

When a bid is overtaken, an outbid notification is sent to the agent and the funds are unblocked.

A bid is successful if not overtaken in 30 seconds.[3] When winning a bid, the agent receives a winner notification and the auction house waits for the blocked funds to be transferred into its account.

If there has been no bid placed on an item, the item remains listed for sale.[4]

[1] You can get this information for a server by asking the `ServerSocket` object. Check the API.

[2] The auction house may end up with fewer than 3 listings eventually if it was selling items from a fixed list.

[3] This is a long enough time to allow someone to outbid, but still short enough that we don't have to wait forever for the auction to end.

[4] In the past, we've had some groups time out auctions after 30 seconds regardless of bidding activity and found it led to chaos during testing, so that behavior is now forbidden.

Agent

Each agent is dynamically created. Upon creation, it opens a bank account by providing a name and an initial balance, and receives a unique account number. (The agent is a client of both the bank and the auction houses.)

The agent gets a list of active auction houses from the bank. In connects to an auction house using the host and port information sent from the bank. The agent receives a list of items being auctioned from the auction house.

When an agent makes a bid on an item, it receives back one or more status messages as the auction proceeds:

- acceptance – The bid is the current high bid
- rejection – The bid was invalid, too low, insufficient funds in the bank, etc.
- outbid – Some other agent has placed a higher bid
- winner – The auction is over and this agent has won.

The agent notifies the bank to transfer the blocked funds to the auction house after it wins a bid.

The program may terminate when no bidding activity is in progress. The program should not allow exit when there are still bids to be resolved. At termination, it de-registers with the bank. An agent terminating should not break the behavior of any other programs in the system.

User Interface

The agent user interface may be console-based or may be a graphical JavaFX display. Whatever you choose, make sure it is clear how to check the agent's bank balance (both total balance and available funds) and interact with the auction houses (viewing items, placing bids, getting current bid status).

Bear in mind that the bidding on an auction is a time sensitive activity, so however you design your user interface, it should not require typing very long commands and/or navigating overly complicated menus.

Testing

Testing should involve at least two auction houses and two agents. The bank program will be started first. Make no assumptions as to the order the agent and auction house programs will be started.

We recommend utilizing computers in the CS network, such as the B146 machines, to test your program.

Some CS machines you can use can be reached as follows:

- ssh your_cs_user_name@moons.cs.unm.edu

- ssh your_cs_user_name@trucks.cs.unm.edu
- ssh your_cs_user_name@b146-XX.cs.unm.edu (where XX is some two digit number)

The best way to transfer your files to these machines is by downloading Filezilla. You can also use the ftp command.

You may also need to transfer your JDK to these machines and set up the java environment for your user.

Grading

Your grade on this project will be determined by a demonstration given to an instructor/grader/TA. These demonstrations will take place at the date as the final for this class. That is Tuesday May 11th from 7:30-9:30. I will be sending out a sign up sheet soon with time slots.

You will need to be ready to go and know how to connect to various CS machines prior to the demo. It might help to come up with some kind of "script" to make sure you hit all of the requirements of the project. We will also ask you to show specific functionality during the demo.

Milestones

Sometimes life prevents you from getting all the requirements of an assignment done. Thus I have decided to have explicit milestones that you can try to reach in order to get a certain percentage of the points. You should not set out trying to only hit these milestones and be done with it, instead these should act as a backup plan. The milestones are as follows:

- To get 100% of the points you need to complete all the requirements given above
- To get 80% of the points you need to complete a system in which there is only 1 bank and 1 auction house but still many agents
- To get 60% of the points you need to complete a system in which there is only 1 bank, 1 auction house and 1 agent.

Note that this is the MOST amount of points that you can earn. Points can still be taken off for not following coding standards, not having design diagrams, etc.