CSCE 3600: Systems Programming

Major Assignment 2 – Sockets and Synchronization

Due: 11:59 PM on Thursday, May 2, 2019

COLLABORATION

You should complete this assignment as a group assignment with the other members of your group as assigned on Canvas using our GitLab environment (i.e., the same group you had for Major 0 and Major 1, unless explicit changes were made by the instructor and communicated to affected group members). If desired, you may submit only one program per group, but all source code *must* be committed in GitLab. Also, make sure that you list the names of all group members who participated in this assignment in order for each to get credit.

PROGRAM DESCRIPTION

In this assignment, you will write two complete C programs to support a client-server model using Linux sockets for a ticketing system. Your programs will develop a two-tiered ticketing system that will consist of a ticket outlet (i.e., the server) that will provide ticket services to "BUY" and "SELL" tickets to two local ticket distributors (i.e., the clients), who will also serve as "scalpers", providing tickets at double the advertised cost to "buyers", who are also the clients.

Server

- The server program will be called with one command-line argument, the port number being used, such as `./major2svr 8001`. If the user calls the server program with too few or too many arguments, you will print out a usage statement and terminate the program.
- The server will generate N tickets, where $N = 25$, and each ticket will be a unique 5-digit number of your choice. Each ticket will cost between \$200 and \$400, inclusively, using a seeded, randomly generated integer to assign the value and initially be made available for sale. In this case, the server will therefore maintain a database of 25 tickets that contains the ticket number, the ticket value, and the availability of the ticket (i.e., whether or not it has been sold or is available). The entire database will be printed before any transactions have taken place (where all tickets are available) and after the last transaction has been completed and both clients have been disconnected.
- The server will set up an Internet domain TCP socket using the port passed in to the server program and listen to the socket for both clients to communicate on that socket. The server should wait for both clients to connect to the socket before processing any (read) requests. Once both clients have connected, however, the server should respond equitably as each request comes in by the different clients.
- The server will support two types of requests as follows:

- `BUY <user balance>`

  When a client sends a `BUY` request with the user's monetary balance, the server can respond in one of three ways:

  - `<ticket number> <ticket price>`

    If the server has tickets available for sale and the user has sufficient funds in his/her user account, then the server will respond with the unique 5-digit ticket number being purchased along with the price of the ticket. After the ticket has been sold to the client, the server will update the database to indicate that the ticket has been sold.

  - `NOFUNDS`

    If the server has tickets available for sale, but the user does not have sufficient funds in his/her user account, then the server will respond with a `NOFUNDS` message. Keep in mind that since users can `SELL` or `SCALP` tickets, the user may acquire funds to allow a future `BUY` request to be successful.

  - `SOLDOUT`

    If the server has no tickets available (i.e., they are currently all sold), then the server will respond with a `SOLDOUT` message. Keep in mind that since the server supports a `SELL` request, tickets may alternate between being sold out and not.

- `SELL <ticket number>`

  When a client sends a `SELL` request with the ticket number, then the server will respond with a `<ticket number> <ticket price>` response indicating that the `SELL` request was successful. After the ticket has been sold to the server, the server will update the database to indicate that the ticket is now available for sale.

• After both clients have disconnected, the server will print the current state of the database and close the sockets.

Client(s) (note that there are 2 clients, but 1 client source code)

• The client program will be called with three command-line arguments: the hostname of the server, the port number being used, and the remote IP address of the "other" client, such as `./major2cli cse06 8001 129.120.151.98`. If the user calls the client program with too few or too many arguments, you will print out a usage statement and terminate the program.

• The client will initially be given a user balance of $4,000 to be used in `BUY`, `SELL`, and `SCALP` requests.

• The client will maintain a database of 15 tickets that contains the ticket number and the ticket value. The entire database plus the remaining user balance will be printed after the last transaction has been completed from both clients (hint: you may want to sleep a couple seconds to allow the other client's transactions to complete as well).

- The client will connect to the Internet domain TCP socket set up by the server using the hostname and port passed in to the client program. This socket connection will be used to issue 15 automated `BUY` and/or `SELL` requests to the server (details to follow shortly).
- The client will then set up an Internet domain UDP socket using the IP address and value of one greater than the port number (i.e., port number + 1) passed in to the client program and listen to the socket for the other active client to communicate on that socket. Note that since the client must simultaneously accomplish to different tasks (i.e., sending automated requests to the server AND listening to the Internet domain UDP socket), you are encouraged to either create a child process or thread to support this functionality. This Internet domain UDP socket will allow the client to behave as a server to the other client, functioning as a "scalper", accepting `SCALP <user balance>` requests from a "buyer".

  When a "buyer" sends a `SCALP` request with the user's monetary balance, the "scalper" can respond in one of two ways:

  - `<ticket number> <two times the ticket price>`

    If the "scalper" has tickets available for sale and the user has sufficient funds in his/her user account for twice the value of the original purchase price of the ticket, then the "scalper" will respond with the unique 5-digit ticket number being purchased along with the value for two times the price of the ticket. The ticket that is selected to be "scalped" should be an existing ticket in the client's database with the minimum purchase price. After the ticket has been sold to the "buyer", the "scalper" will remove the corresponding entry from its database and update the client's user balance with the money received from the `SCALP` request.

  - `NOMONEY`

    If the "scalper" has tickets available for sale, but the "buyer" does not have sufficient funds in his/her user account to purchase the ticket at twice the purchase price, then the "scalper" will respond with a `NOMONEY` message. Keep in mind that since users can `SELL` or `SCALP` tickets, the user may acquire funds to allow a future `SCALP` request to be successful.

- The client will send 15 automated `BUY <user balance>` requests to the server. The server can respond in one of three ways:

  - `<ticket number> <ticket price>`

    If the server responds with the `<ticket number> <ticket price>` message, then the client will update the client's database with the newly acquired ticket information and deduct the purchase price from his/her user balance.

  - `NOFUNDS`

  If the server responds with a `NOFUNDS` message, then the client will then select any ticket he/she chooses and send a `SELL <ticket number>` request back to the server to sell the ticket back to the server at the original purchase price to acquire additional funds. After the ticket has been sold to the server, the client will remove the corresponding entry from its database and update the client's user balance with the money received from the `SELL` request.

- `SOLDOUT`

  If the server responds with a `SOLDOUT` message, the client will then transform its behavior to a "buyer", which means that he/she will then conduct a transaction with a "scalper", which is essentially the "other" client. To accomplish this, the now "buyer" will connect to the Internet domain UDP socket set up by the other client using the IP address and value of one greater than the port number (i.e., port number + 1) passed in to the client program, although technically you will not "connect" since it is a UDP socket and send a `SCALP <user balance>` request to the "scalper" (i.e., other client).

  When a "buyer" sends a `SCALP` request with the user's monetary balance, the "scalper" can respond in one of two ways:

  - `<ticket number> <two times the ticket price>`

    If the "scalper" responds with the `<ticket number> <two times the ticket price>` message, then the "buyer" will update the client's database with the newly acquired ticket information and deduct the purchase price from his/her user balance.

  - `NOMONEY`

    If the "scalper" responds with a `NOMONEY` message, the "buyer" will then select any ticket he/she chooses and send a `SELL <ticket number>` request back to the server to sell the ticket back to the server at the original purchase price to acquire additional funds. After the ticket has been sold to the server, the client will remove the corresponding entry from its database and update the client's user balance with the money received from the `SELL` request.

- After the client has completed its 15 automated BUY requests, the client will print the current state of the database along with the remaining user balance and close the socket.

General Requirements and Comments

- There are four actors in this ticketing system: (1) the server, (2) the clients, (3) the buyers, and (4) the scalpers. All incoming (`<-`), outgoing (`->`), and important status (`<>`) messages must be printed throughout the execution of both the server and client programs. Since each of these will be interacting simultaneously with one another, you must distinguish among each actor by printing in a different color (see

SAMPLE OUTPUT). Additionally, you should identify and distinguish among incoming, outgoing, and important status messages (you don't have to use the format given here, but it is presented for your benefit).

- Other than defined constants, only the client's database and user balance may be implemented as a global variable. Otherwise, all other data structures must be passed as parameters to functions/system calls.

GROUP COLLABORATIVE PORTION

Although this assignment is somewhat coupled in its functionality, the "group" component of this assignment will consist of the program invocation (including checking for and parsing command-line arguments), initial database setup, and Internet domain TCP socket set up for the server-client as well as Internet domain UDP socket set up for the scalper-buyer, which includes creation of the child process or thread if needed.

INDIVIDUAL PORTION

In this assignment, each member of the group will implement portions of the messaging sent over the respective sockets. Unfortunately, all of the socket messaging functionality is needed, so all groups (whether having 3 or 4 members) must implement all of the functionality for this assignment. However, each team may decide on their own (with consensus from group members) which messaging each team member implements. For example, one team member may be assigned support of the `BUY` requests (i.e., the responses) on the server, while another might be assigned the handling of the `SOLDOUT` message, which includes "buyer" functionality in the support of the `SCALP` request. It is up to each team to decide which individual team members are assigned which messages, but care should be taken to ensure that this division is equitable, as contribution points will be awarded based on survey results (and verification of code on GitLab).

SAMPLE OUTPUT (user input shown in **bold**):

==> SERVER on cse06

```
mat0299@cse06:~$ ./major2svr
usage: ./major2svr port
mat0299@cse06:~$ ./major2svr 8001
[<> SERVER  ]: Database Table:
TICKET NUMBER  PRICE  STATUS
----------------------------
[Tkt# 10000]:  $ 369  AVAIL
[Tkt# 10001]:  $ 208  AVAIL
[Tkt# 10002]:  $ 254  AVAIL
[Tkt# 10003]:  $ 311  AVAIL
[Tkt# 10004]:  $ 222  AVAIL
[Tkt# 10005]:  $ 249  AVAIL
[Tkt# 10006]:  $ 244  AVAIL
[Tkt# 10007]:  $ 388  AVAIL
[Tkt# 10008]:  $ 349  AVAIL
[Tkt# 10009]:  $ 349  AVAIL
[Tkt# 10010]:  $ 330  AVAIL
[Tkt# 10011]:  $ 298  AVAIL
[Tkt# 10012]:  $ 364  AVAIL
```

```
[Tkt# 10013]:  $ 253  AVAIL
[Tkt# 10014]:  $ 383  AVAIL
[Tkt# 10015]:  $ 340  AVAIL
[Tkt# 10016]:  $ 241  AVAIL
[Tkt# 10017]:  $ 229  AVAIL
[Tkt# 10018]:  $ 273  AVAIL
[Tkt# 10019]:  $ 279  AVAIL
[Tkt# 10020]:  $ 329  AVAIL
[Tkt# 10021]:  $ 243  AVAIL
[Tkt# 10022]:  $ 292  AVAIL
[Tkt# 10023]:  $ 221  AVAIL
[Tkt# 10024]:  $ 387  AVAIL
----------------------------
0 clients connected. Waiting on CLIENT 1 to connect.
1 client  connected. Waiting on CLIENT 2 to connect.
2 clients connected. Ready for incoming requests...
[<- CLIENT 1]: BUY 4000
[-> SERVER  ]: CLIENT 1 BUY 10000 $369 OK
[<- CLIENT 2]: BUY 4000
[-> SERVER  ]: CLIENT 2 BUY 10001 $208 OK
[<- CLIENT 2]: BUY 3792
[-> SERVER  ]: CLIENT 2 BUY 10002 $254 OK
[<- CLIENT 1]: BUY 3631
[-> SERVER  ]: CLIENT 1 BUY 10003 $311 OK
[<- CLIENT 2]: BUY 3538
[-> SERVER  ]: CLIENT 2 BUY 10004 $222 OK
[<- CLIENT 1]: BUY 3320
[-> SERVER  ]: CLIENT 1 BUY 10005 $249 OK
[<- CLIENT 2]: BUY 3316
[-> SERVER  ]: CLIENT 2 BUY 10006 $244 OK
[<- CLIENT 1]: BUY 3071
[-> SERVER  ]: CLIENT 1 BUY 10007 $388 OK
[<- CLIENT 2]: BUY 3072
[-> SERVER  ]: CLIENT 2 BUY 10008 $349 OK
[<- CLIENT 1]: BUY 2683
[-> SERVER  ]: CLIENT 1 BUY 10009 $349 OK
[<- CLIENT 2]: BUY 2723
[-> SERVER  ]: CLIENT 2 BUY 10010 $330 OK
[<- CLIENT 1]: BUY 2334
[-> SERVER  ]: CLIENT 1 BUY 10011 $298 OK
[<- CLIENT 2]: BUY 2393
[-> SERVER  ]: CLIENT 2 BUY 10012 $364 OK
[<- CLIENT 1]: BUY 2036
[-> SERVER  ]: CLIENT 1 BUY 10013 $253 OK
[<- CLIENT 2]: BUY 2029
[-> SERVER  ]: CLIENT 2 BUY 10014 $383 OK
[<- CLIENT 1]: BUY 1783
[-> SERVER  ]: CLIENT 1 BUY 10015 $340 OK
[<- CLIENT 2]: BUY 1646
[-> SERVER  ]: CLIENT 2 BUY 10016 $241 OK
[<- CLIENT 1]: BUY 1443
[-> SERVER  ]: CLIENT 1 BUY 10017 $229 OK
[<- CLIENT 2]: BUY 1405
[-> SERVER  ]: CLIENT 2 BUY 10018 $273 OK
[<- CLIENT 1]: BUY 1214
[-> SERVER  ]: CLIENT 1 BUY 10019 $279 OK
[<- CLIENT 1]: BUY 935
```

```
[-> SERVER  ]: CLIENT 1 BUY 10020 $329 OK
[<- CLIENT 2]: BUY 1132
[-> SERVER  ]: CLIENT 2 BUY 10021 $243 OK
[<- CLIENT 1]: BUY 606
[-> SERVER  ]: CLIENT 1 BUY 10022 $292 OK
[<- CLIENT 2]: BUY 889
[-> SERVER  ]: CLIENT 2 BUY 10023 $221 OK
[<- CLIENT 1]: BUY 314
[-> SERVER  ]: CLIENT 1 BUY NOFUNDS
[<- CLIENT 2]: BUY 668
[-> SERVER  ]: CLIENT 2 BUY 10024 $387 OK
[<- CLIENT 1]: SELL 10000
[-> SERVER  ]: CLIENT 1 SELL 10000 OK
[<- CLIENT 2]: BUY 281
[-> SERVER  ]: CLIENT 2 BUY NOFUNDS
[<- CLIENT 1]: BUY 683
[-> SERVER  ]: CLIENT 1 BUY 10000 $369 OK
[<- CLIENT 2]: SELL 10001
[-> SERVER  ]: CLIENT 2 SELL 10001 OK
[<- CLIENT 1]: BUY 314
[-> SERVER  ]: CLIENT 1 BUY 10001 $208 OK
[<- CLIENT 2]: BUY 489
[-> SERVER  ]: CLIENT 2 BUY SOLDOUT
Both clients disconnected. Preparing to shut down...
[<> SERVER  ]: Database Table:
TICKET NUMBER  PRICE  STATUS
----------------------------
[Tkt# 10000]:  $ 369  SOLD
[Tkt# 10001]:  $ 208  SOLD
[Tkt# 10002]:  $ 254  SOLD
[Tkt# 10003]:  $ 311  SOLD
[Tkt# 10004]:  $ 222  SOLD
[Tkt# 10005]:  $ 249  SOLD
[Tkt# 10006]:  $ 244  SOLD
[Tkt# 10007]:  $ 388  SOLD
[Tkt# 10008]:  $ 349  SOLD
[Tkt# 10009]:  $ 349  SOLD
[Tkt# 10010]:  $ 330  SOLD
[Tkt# 10011]:  $ 298  SOLD
[Tkt# 10012]:  $ 364  SOLD
[Tkt# 10013]:  $ 253  SOLD
[Tkt# 10014]:  $ 383  SOLD
[Tkt# 10015]:  $ 340  SOLD
[Tkt# 10016]:  $ 241  SOLD
[Tkt# 10017]:  $ 229  SOLD
[Tkt# 10018]:  $ 273  SOLD
[Tkt# 10019]:  $ 279  SOLD
[Tkt# 10020]:  $ 329  SOLD
[Tkt# 10021]:  $ 243  SOLD
[Tkt# 10022]:  $ 292  SOLD
[Tkt# 10023]:  $ 221  SOLD
[Tkt# 10024]:  $ 387  SOLD
----------------------------
```

==> CLIENT 1 on cse05 (first client to connect to server)

```
mat0299@cse05:~$ ./major2cli
```

```
usage ./major2cli <hostname> <port> <IP address>
mat0299@cse05:~$ ./major2cli cse06 8001 129.120.151.97
[-> CLIENT ]: BUY 4000
[<- SERVER ]: 10000 369
[<> CLIENT ]: BUY 10000 $369 OK
[-> CLIENT ]: BUY 3631
[<- SERVER ]: 10003 311
[<> CLIENT ]: BUY 10003 $311 OK
[-> CLIENT ]: BUY 3320
[<- SERVER ]: 10005 249
[<> CLIENT ]: BUY 10005 $249 OK
[-> CLIENT ]: BUY 3071
[<- SERVER ]: 10007 388
[<> CLIENT ]: BUY 10007 $388 OK
[-> CLIENT ]: BUY 2683
[<- SERVER ]: 10009 349
[<> CLIENT ]: BUY 10009 $349 OK
[-> CLIENT ]: BUY 2334
[<- SERVER ]: 10011 298
[<> CLIENT ]: BUY 10011 $298 OK
[-> CLIENT ]: BUY 2036
[<- SERVER ]: 10013 253
[<> CLIENT ]: BUY 10013 $253 OK
[-> CLIENT ]: BUY 1783
[<- SERVER ]: 10015 340
[<> CLIENT ]: BUY 10015 $340 OK
[-> CLIENT ]: BUY 1443
[<- SERVER ]: 10017 229
[<> CLIENT ]: BUY 10017 $229 OK
[-> CLIENT ]: BUY 1214
[<- SERVER ]: 10019 279
[<> CLIENT ]: BUY 10019 $279 OK
[-> CLIENT ]: BUY 935
[<- SERVER ]: 10020 329
[<> CLIENT ]: BUY 10020 $329 OK
[-> CLIENT ]: BUY 606
[<- SERVER ]: 10022 292
[<> CLIENT ]: BUY 10022 $292 OK
[-> CLIENT ]: BUY 314
[<- SERVER ]: NOFUNDS
[-> CLIENT ]: SELL 10000
[<- SERVER ]: 10000 369
[<> CLIENT ]: SELL 10000 $369 OK
[-> CLIENT ]: BUY 683
[<- SERVER ]: 10000 369
[<> CLIENT ]: BUY 10000 $369 OK
[-> CLIENT ]: BUY 314
[<- SERVER ]: 10001 208
[<> CLIENT ]: BUY 10001 $208 OK
[<- BUYER  ]: SCALP 489
[-> SCALPER]: SCALP 10001 $416
[<> CLIENT ]: Database Table:
TICKET NUMBER  PRICE
-----------------------------
[Tkt# 10000]:  $ 369
[Tkt# 10003]:  $ 311
[Tkt# 10005]:  $ 249
```

```
[Tkt# 10007]:  $ 388
[Tkt# 10009]:  $ 349
[Tkt# 10011]:  $ 298
[Tkt# 10013]:  $ 253
[Tkt# 10015]:  $ 340
[Tkt# 10017]:  $ 229
[Tkt# 10019]:  $ 279
[Tkt# 10020]:  $ 329
[Tkt# 10022]:  $ 292
[Tkt#     0]:  $   0
[Tkt#     0]:  $   0
[Tkt#     0]:  $   0
---------------------------
BALANCE    :   $ 522
```

==> CLIENT 2 on cse04 (second client to connect to server)

```
mat0299@cse04:~$ ./major2cli cse06 8001 129.120.151.98
[-> CLIENT ]: BUY 4000
[<- SERVER ]: 10001 208
[<> CLIENT ]: BUY 10001 $208 OK
[-> CLIENT ]: BUY 3792
[<- SERVER ]: 10002 254
[<> CLIENT ]: BUY 10002 $254 OK
[-> CLIENT ]: BUY 3538
[<- SERVER ]: 10004 222
[<> CLIENT ]: BUY 10004 $222 OK
[-> CLIENT ]: BUY 3316
[<- SERVER ]: 10006 244
[<> CLIENT ]: BUY 10006 $244 OK
[-> CLIENT ]: BUY 3072
[<- SERVER ]: 10008 349
[<> CLIENT ]: BUY 10008 $349 OK
[-> CLIENT ]: BUY 2723
[<- SERVER ]: 10010 330
[<> CLIENT ]: BUY 10010 $330 OK
[-> CLIENT ]: BUY 2393
[<- SERVER ]: 10012 364
[<> CLIENT ]: BUY 10012 $364 OK
[-> CLIENT ]: BUY 2029
[<- SERVER ]: 10014 383
[<> CLIENT ]: BUY 10014 $383 OK
[-> CLIENT ]: BUY 1646
[<- SERVER ]: 10016 241
[<> CLIENT ]: BUY 10016 $241 OK
[-> CLIENT ]: BUY 1405
[<- SERVER ]: 10018 273
[<> CLIENT ]: BUY 10018 $273 OK
[-> CLIENT ]: BUY 1132
[<- SERVER ]: 10021 243
[<> CLIENT ]: BUY 10021 $243 OK
[-> CLIENT ]: BUY 889
[<- SERVER ]: 10023 221
[<> CLIENT ]: BUY 10023 $221 OK
[-> CLIENT ]: BUY 668
[<- SERVER ]: 10024 387
[<> CLIENT ]: BUY 10024 $387 OK
```

```
[-> CLIENT ]: BUY 281
[<- SERVER ]: NOFUNDS
[-> CLIENT ]: SELL 10001
[<- SERVER ]: 10001 208
[<> CLIENT ]: SELL 10001 $208 OK
[-> CLIENT ]: BUY 489
[<- SERVER ]: SOLDOUT
[-> BUYER  ]: SCALP 489
[<- SCALPER]: 10001 416
[<> BUYER  ]: SCALP 10001 $416 OK
[<> CLIENT ]: Database Table:
TICKET NUMBER  PRICE
-----------------------------
[Tkt# 10001]:  $ 416
[Tkt# 10002]:  $ 254
[Tkt# 10004]:  $ 222
[Tkt# 10006]:  $ 244
[Tkt# 10008]:  $ 349
[Tkt# 10010]:  $ 330
[Tkt# 10012]:  $ 364
[Tkt# 10014]:  $ 383
[Tkt# 10016]:  $ 241
[Tkt# 10018]:  $ 273
[Tkt# 10021]:  $ 243
[Tkt# 10023]:  $ 221
[Tkt# 10024]:  $ 387
[Tkt#     0]:  $   0
[Tkt#     0]:  $   0
-----------------------------
BALANCE    :   $  73
```

Since there are random aspects to this assignment, a second sample is being provided here to show how different runs of the program can yield different results:

==> SERVER on cse06

```
mat0299@cse06:~$ ./major2svr 8001
[<> SERVER  ]: Database Table:
TICKET NUMBER  PRICE  STATUS
----------------------------
[Tkt# 10000]:  $ 350  AVAIL
[Tkt# 10001]:  $ 301  AVAIL
[Tkt# 10002]:  $ 224  AVAIL
[Tkt# 10003]:  $ 300  AVAIL
[Tkt# 10004]:  $ 254  AVAIL
[Tkt# 10005]:  $ 354  AVAIL
[Tkt# 10006]:  $ 390  AVAIL
[Tkt# 10007]:  $ 288  AVAIL
[Tkt# 10008]:  $ 335  AVAIL
[Tkt# 10009]:  $ 398  AVAIL
[Tkt# 10010]:  $ 337  AVAIL
[Tkt# 10011]:  $ 316  AVAIL
[Tkt# 10012]:  $ 324  AVAIL
[Tkt# 10013]:  $ 378  AVAIL
[Tkt# 10014]:  $ 294  AVAIL
[Tkt# 10015]:  $ 271  AVAIL
[Tkt# 10016]:  $ 398  AVAIL
```

```
[Tkt# 10017]:  $ 254  AVAIL
[Tkt# 10018]:  $ 365  AVAIL
[Tkt# 10019]:  $ 369  AVAIL
[Tkt# 10020]:  $ 296  AVAIL
[Tkt# 10021]:  $ 322  AVAIL
[Tkt# 10022]:  $ 237  AVAIL
[Tkt# 10023]:  $ 386  AVAIL
[Tkt# 10024]:  $ 311  AVAIL
----------------------------
0 clients connected. Waiting on CLIENT 1 to connect.
1 client  connected. Waiting on CLIENT 2 to connect.
2 clients connected. Ready for incoming requests...
[<- CLIENT 1]: BUY 4000
[-> SERVER  ]: CLIENT 1 BUY 10000 $350 OK
[<- CLIENT 2]: BUY 4000
[-> SERVER  ]: CLIENT 2 BUY 10001 $301 OK
[<- CLIENT 1]: BUY 3650
[-> SERVER  ]: CLIENT 1 BUY 10002 $224 OK
[<- CLIENT 2]: BUY 3699
[-> SERVER  ]: CLIENT 2 BUY 10003 $300 OK
[<- CLIENT 1]: BUY 3426
[-> SERVER  ]: CLIENT 1 BUY 10004 $254 OK
[<- CLIENT 2]: BUY 3399
[-> SERVER  ]: CLIENT 2 BUY 10005 $354 OK
[<- CLIENT 1]: BUY 3172
[-> SERVER  ]: CLIENT 1 BUY 10006 $390 OK
[<- CLIENT 2]: BUY 3045
[-> SERVER  ]: CLIENT 2 BUY 10007 $288 OK
[<- CLIENT 1]: BUY 2782
[-> SERVER  ]: CLIENT 1 BUY 10008 $335 OK
[<- CLIENT 2]: BUY 2757
[-> SERVER  ]: CLIENT 2 BUY 10009 $398 OK
[<- CLIENT 1]: BUY 2447
[-> SERVER  ]: CLIENT 1 BUY 10010 $337 OK
[<- CLIENT 2]: BUY 2359
[-> SERVER  ]: CLIENT 2 BUY 10011 $316 OK
[<- CLIENT 1]: BUY 2110
[-> SERVER  ]: CLIENT 1 BUY 10012 $324 OK
[<- CLIENT 2]: BUY 2043
[-> SERVER  ]: CLIENT 2 BUY 10013 $378 OK
[<- CLIENT 1]: BUY 1786
[-> SERVER  ]: CLIENT 1 BUY 10014 $294 OK
[<- CLIENT 2]: BUY 1665
[-> SERVER  ]: CLIENT 2 BUY 10015 $271 OK
[<- CLIENT 1]: BUY 1492
[-> SERVER  ]: CLIENT 1 BUY 10016 $398 OK
[<- CLIENT 2]: BUY 1394
[-> SERVER  ]: CLIENT 2 BUY 10017 $254 OK
[<- CLIENT 1]: BUY 1094
[-> SERVER  ]: CLIENT 1 BUY 10018 $365 OK
[<- CLIENT 2]: BUY 1140
[-> SERVER  ]: CLIENT 2 BUY 10019 $369 OK
[<- CLIENT 1]: BUY 729
[-> SERVER  ]: CLIENT 1 BUY 10020 $296 OK
[<- CLIENT 2]: BUY 771
[-> SERVER  ]: CLIENT 2 BUY 10021 $322 OK
[<- CLIENT 1]: BUY 433
```

```
[-> SERVER  ]: CLIENT 1 BUY 10022 $237 OK
[<- CLIENT 2]: BUY 449
[-> SERVER  ]: CLIENT 2 BUY 10023 $386 OK
[<- CLIENT 1]: BUY 196
[-> SERVER  ]: CLIENT 1 BUY NOFUNDS
[<- CLIENT 1]: SELL 10000
[-> SERVER  ]: CLIENT 1 SELL 10000 OK
[<- CLIENT 2]: BUY 63
[-> SERVER  ]: CLIENT 2 BUY NOFUNDS
[<- CLIENT 2]: SELL 10001
[-> SERVER  ]: CLIENT 2 SELL 10001 OK
[<- CLIENT 1]: BUY 546
[-> SERVER  ]: CLIENT 1 BUY 10000 $350 OK
[<- CLIENT 2]: BUY 364
[-> SERVER  ]: CLIENT 2 BUY 10001 $301 OK
[<- CLIENT 1]: BUY 196
[-> SERVER  ]: CLIENT 1 BUY NOFUNDS
[<- CLIENT 2]: BUY 63
[-> SERVER  ]: CLIENT 2 BUY NOFUNDS
[<- CLIENT 1]: SELL 10000
[-> SERVER  ]: CLIENT 1 SELL 10000 OK
[<- CLIENT 2]: SELL 10001
[-> SERVER  ]: CLIENT 2 SELL 10001 OK
Both clients disconnected. Preparing to shut down...
[<> SERVER  ]: Database Table:
TICKET NUMBER  PRICE  STATUS
----------------------------
[Tkt# 10000]:  $ 350  AVAIL
[Tkt# 10001]:  $ 301  AVAIL
[Tkt# 10002]:  $ 224  SOLD
[Tkt# 10003]:  $ 300  SOLD
[Tkt# 10004]:  $ 254  SOLD
[Tkt# 10005]:  $ 354  SOLD
[Tkt# 10006]:  $ 390  SOLD
[Tkt# 10007]:  $ 288  SOLD
[Tkt# 10008]:  $ 335  SOLD
[Tkt# 10009]:  $ 398  SOLD
[Tkt# 10010]:  $ 337  SOLD
[Tkt# 10011]:  $ 316  SOLD
[Tkt# 10012]:  $ 324  SOLD
[Tkt# 10013]:  $ 378  SOLD
[Tkt# 10014]:  $ 294  SOLD
[Tkt# 10015]:  $ 271  SOLD
[Tkt# 10016]:  $ 398  SOLD
[Tkt# 10017]:  $ 254  SOLD
[Tkt# 10018]:  $ 365  SOLD
[Tkt# 10019]:  $ 369  SOLD
[Tkt# 10020]:  $ 296  SOLD
[Tkt# 10021]:  $ 322  SOLD
[Tkt# 10022]:  $ 237  SOLD
[Tkt# 10023]:  $ 386  SOLD
[Tkt# 10024]:  $ 311  AVAIL
----------------------------
```

==> CLIENT 1 on cse05 (first client to connect to server)

```
mat0299@cse05:~$ ./major2cli cse06 8001 129.120.151.97
```

```
[-> CLIENT ]: BUY 4000
[<- SERVER ]: 10000 350
[<> CLIENT ]: BUY 10000 $350 OK
[-> CLIENT ]: BUY 3650
[<- SERVER ]: 10002 224
[<> CLIENT ]: BUY 10002 $224 OK
[-> CLIENT ]: BUY 3426
[<- SERVER ]: 10004 254
[<> CLIENT ]: BUY 10004 $254 OK
[-> CLIENT ]: BUY 3172
[<- SERVER ]: 10006 390
[<> CLIENT ]: BUY 10006 $390 OK
[-> CLIENT ]: BUY 2782
[<- SERVER ]: 10008 335
[<> CLIENT ]: BUY 10008 $335 OK
[-> CLIENT ]: BUY 2447
[<- SERVER ]: 10010 337
[<> CLIENT ]: BUY 10010 $337 OK
[-> CLIENT ]: BUY 2110
[<- SERVER ]: 10012 324
[<> CLIENT ]: BUY 10012 $324 OK
[-> CLIENT ]: BUY 1786
[<- SERVER ]: 10014 294
[<> CLIENT ]: BUY 10014 $294 OK
[-> CLIENT ]: BUY 1492
[<- SERVER ]: 10016 398
[<> CLIENT ]: BUY 10016 $398 OK
[-> CLIENT ]: BUY 1094
[<- SERVER ]: 10018 365
[<> CLIENT ]: BUY 10018 $365 OK
[-> CLIENT ]: BUY 729
[<- SERVER ]: 10020 296
[<> CLIENT ]: BUY 10020 $296 OK
[-> CLIENT ]: BUY 433
[<- SERVER ]: 10022 237
[<> CLIENT ]: BUY 10022 $237 OK
[-> CLIENT ]: BUY 196
[<- SERVER ]: NOFUNDS
[-> CLIENT ]: SELL 10000
[<- SERVER ]: 10000 350
[<> CLIENT ]: SELL 10000 $350 OK
[-> CLIENT ]: BUY 546
[<- SERVER ]: 10000 350
[<> CLIENT ]: BUY 10000 $350 OK
[-> CLIENT ]: BUY 196
[<- SERVER ]: NOFUNDS
[-> CLIENT ]: SELL 10000
[<- SERVER ]: 10000 350
[<> CLIENT ]: SELL 10000 $350 OK
[<> CLIENT ]: Database Table:
TICKET NUMBER  PRICE
-----------------------------
[Tkt#     0]:  $   0
[Tkt# 10002]:  $ 224
[Tkt# 10004]:  $ 254
[Tkt# 10006]:  $ 390
[Tkt# 10008]:  $ 335
```

```
[Tkt# 10010]:  $ 337
[Tkt# 10012]:  $ 324
[Tkt# 10014]:  $ 294
[Tkt# 10016]:  $ 398
[Tkt# 10018]:  $ 365
[Tkt# 10020]:  $ 296
[Tkt# 10022]:  $ 237
[Tkt#     0]:  $   0
[Tkt#     0]:  $   0
[Tkt#     0]:  $   0
-----------------------------
BALANCE     :  $ 546
```

==> CLIENT 2 on cse04 (second client to connect to server)

```
mat0299@cse04:~$ ./major2cli cse06 8001 129.120.151.98
[-> CLIENT ]: BUY 4000
[<- SERVER ]: 10001 301
[<> CLIENT ]: BUY 10001 $301 OK
[-> CLIENT ]: BUY 3699
[<- SERVER ]: 10003 300
[<> CLIENT ]: BUY 10003 $300 OK
[-> CLIENT ]: BUY 3399
[<- SERVER ]: 10005 354
[<> CLIENT ]: BUY 10005 $354 OK
[-> CLIENT ]: BUY 3045
[<- SERVER ]: 10007 288
[<> CLIENT ]: BUY 10007 $288 OK
[-> CLIENT ]: BUY 2757
[<- SERVER ]: 10009 398
[<> CLIENT ]: BUY 10009 $398 OK
[-> CLIENT ]: BUY 2359
[<- SERVER ]: 10011 316
[<> CLIENT ]: BUY 10011 $316 OK
[-> CLIENT ]: BUY 2043
[<- SERVER ]: 10013 378
[<> CLIENT ]: BUY 10013 $378 OK
[-> CLIENT ]: BUY 1665
[<- SERVER ]: 10015 271
[<> CLIENT ]: BUY 10015 $271 OK
[-> CLIENT ]: BUY 1394
[<- SERVER ]: 10017 254
[<> CLIENT ]: BUY 10017 $254 OK
[-> CLIENT ]: BUY 1140
[<- SERVER ]: 10019 369
[<> CLIENT ]: BUY 10019 $369 OK
[-> CLIENT ]: BUY 771
[<- SERVER ]: 10021 322
[<> CLIENT ]: BUY 10021 $322 OK
[-> CLIENT ]: BUY 449
[<- SERVER ]: 10023 386
[<> CLIENT ]: BUY 10023 $386 OK
[-> CLIENT ]: BUY 63
[<- SERVER ]: NOFUNDS
[-> CLIENT ]: SELL 10001
[<- SERVER ]: 10001 301
[<> CLIENT ]: SELL 10001 $301 OK
```

```
[-> CLIENT ]: BUY 364
[<- SERVER ]: 10001 301
[<> CLIENT ]: BUY 10001 $301 OK
[-> CLIENT ]: BUY 63
[<- SERVER ]: NOFUNDS
[-> CLIENT ]: SELL 10001
[<- SERVER ]: 10001 301
[<> CLIENT ]: SELL 10001 $301 OK
[<> CLIENT ]: Database Table:
TICKET NUMBER  PRICE
-----------------------------
[Tkt#     0]:  $   0
[Tkt# 10003]:  $ 300
[Tkt# 10005]:  $ 354
[Tkt# 10007]:  $ 288
[Tkt# 10009]:  $ 398
[Tkt# 10011]:  $ 316
[Tkt# 10013]:  $ 378
[Tkt# 10015]:  $ 271
[Tkt# 10017]:  $ 254
[Tkt# 10019]:  $ 369
[Tkt# 10021]:  $ 322
[Tkt# 10023]:  $ 386
[Tkt#     0]:  $   0
[Tkt#     0]:  $   0
[Tkt#     0]:  $   0
-----------------------------
BALANCE    :   $ 364
```

GRADING

Your C program file(s), `README`, and makefile should be committed to our GitLab environment as follows:

- Your C program file(s). Your code should be well documented in terms of comments. For example, good comments in general consist of a header (with your name, course section, date, and brief description), comments for each variable, and commented blocks of code.
- A **`README`** file with some basic documentation about your code. This file should contain the following four components:
  - Your name(s).
  - Organization of the Project. Since there are multiple components in this project, you will describe how the work was organized and managed, including which team members were responsible for what components – there are lots of ways to do this, so your team needs to come up with the best way that works based on your team's strengths. Note that this may be used in assessment of grades for this project.
  - Design Overview: A few paragraphs describing the overall structure of your code and any important structures.
  - Complete Specification: Describe how you handled any ambiguities in the

    specification. For example, for this project, explain how your shell will handle lines that have no commands between semi-colons.
    - Known Bugs or Problems: A list of any features that you did not implement or that you know are not working correctly.

- A completed group assessment evaluation (given at a later date) for each team member. *Please be aware that a student receiving a poor evaluation with regards to their performance on the team will have his/her grading marks reduced by an appropriate amount, based on the evaluation. In addition, the rubric for this assignment allows modification of each individual's portion of the group grade to account for individual contribution to the group's submission, which may result in a member of the group receiving a much higher or much lower grade than other members of the group. This implies that in the individual portion, each group member is responsible for "committing" his or her own code. Additionally, there are points allocated for participation and contribution to the overall project.*
- A `Makefile` for compiling your source code, including a clean directive. When using `gcc` to compile your code, use the `–Wall` switch to ensure that all warnings are displayed. Do not be satisfied with code that merely compiles – it should compile with no warnings! You will lose points if your code produces warnings when compiled.
- Your program will be graded based largely on whether it works correctly on the CSE machines (e.g., `cse01`, `cse02`, …, `cse06`), so you should make sure that your program compiles and runs on a CSE machine.

SUBMISSION

- Each team may electronically submit all applicable components to the **Major 2** dropbox in Canvas, but all work must be submitted to our GitLab environment by the due date.