I need two complete Python programs to support a client-server model using Linux sockets for a ticketing system. The programs will develop a two-tiered ticketing system that will consist of a ticket outlet (i.e., the server) that will provide ticket services such as “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 the “buyer”, which is the other client.

* The server will:

• generate N tickets, where N = 25, and each ticket will have a unique 5-digit number. 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 (where all tickets are available), every time a transaction took place, and after the last transaction has been completed and both clients have been disconnected.

• Set up an Internet domain TCP socket using the port passed into the server program and listen to the socket for both clients to communicate on that socket. The server must 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 simultaneously comes from different clients.

• Support two types of requests:

1. 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.

1. 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.

1. 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.

1. SELL <ticket number>

When a client sends a SELL request with the ticket number, then the server will respond with a response indicating that the SELL request was successful (i.e., <ticket number> <ticket original price>). 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.
* There are 2 clients, but only 1 client source code. The clients 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 of 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 into the client program. This socket connection will be used to issue 15 automated BUY and/or SELL requests to the server (details on how to proceed are described below).
* 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 requests from a “buyer”
* The client will send 15 automated BUY requests to the server in a format: BUY <user balance>. The client will behave based on the server’s respond to each BUY request. If the server’s respond is:
  1. <ticket number> <ticket price>  
     The client will update the client’s database with the newly acquired ticket information and deduct the purchase price from his/her user balance.
  2. NOFUNDS  
     The client will then select any ticket he/she chooses and send a SELL 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.
  3. SOLDOUT  
     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. The client will send a SCALP request to the “scalper” (i.e., the other client).
* The client can send a SCALP request to the other client in the format of: SCALP <user balance>.  
  The client will need to act based on the scalper’s response, which can be one of two ways:

1. <ticket number> <two times the ticket price>  
   If the “scalper” responds with the 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.
2. NOMONEY  
   If the “scalper” responds with a NOMONEY message, the “buyer” will then select any ticket he/she chooses and send a SELL 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.

Don’t implement the code yet. We need to outline the steps that must be done for the client to function as required.

Strategy: Provide the requirements of the server and client programs separately, and ask ChatGPT to outline the steps to do before the actual implementation.

Currently: ask ChatGPT to implement the server program and complete its functionalities before moving on to implementing client program.