

Warehouse Inventory System

ECE 4220: Real Time Embedded Systems



Abstract

This project encompasses many of the topics covered in class and demonstrates an inventory system that could be implemented on an embedded device where the user can remain mobile and would not be constrained to a workstation.

Introduction

This product is to be implemented on a mobile inventory-controlling device similar to the one on the cover of this report. These devices are very useful in large department and grocery stores because they allow the user to be actively helping customers while also being able to view their store's inventory, as well as order more products from their distribution warehouses. This project includes both a client program, to be used by employees of the inventory devices, as well as a server program (referred to as the distribution program), to be used by the distribution warehouse to accept orders from the client. To manage inventory, a database system that is supported on embedded devices and provided by Faircom was implemented.

The goals of the client program is to be able to place an order to the distribution warehouse, view their own store's current inventory, view the distribution warehouse's current inventory, view the open orders the client store currently has, and view the current invoices for their store as well as the total amount owed from those invoices. The goal of the distribution program is to be able to interpret messages from the client program and be able to update and create orders and invoices according to the orders made by the client program.

The motivation behind this project is to create a mobile inventory system that could be a cost effective, marketable product to lower-end grocery stores that may not have technology of this kind.

Background

This is a classic inventory system with a twist because it can be implemented on a mobile device with inventory data stored directly on the device using the embedded database. As stated above, this application is incredibly useful for keeping employees who would normally be restricted to a desktop computer or laptop to order or check inventory, being able to accomplish these tasks while actively helping customers, which overall increases the productivity of the employees in a store.

Proposed Method

To accomplish the goals of this project, the Faircom database server was used to provide data structures that would be able to store data for all stores in the distribution warehouse's network as well as store data for processing orders, invoices, and the warehouse's inventory. To be able to access tables and data in the database, a connection to the database server by both the client and the server programs . The next section will cover how the client program was implemented, followed by a description of how the distribution program was implemented.

Client

The client program begins with a username and password login screen where the user must provide a valid username and password combo to gain access to any of the functionalities of the programs. System login data is stored in a text file called login_Client.txt. The structure of this file is shown in Figure 1.

```
login_Client.txt
(number of entries (n))
(username_1),(password_1),(storeid_1),(store_name_1)
(username_2),(password_2),(storeid_2),(store_name_2)
...
(username_n),(password_n),(storeid_n),(store_name_n)
```

Figure : Format of login_Client.txt

Each line of the file is read and tokenized so that the username and password provided can be compared with the system data. If a valid username and password combo is found, the store id, an integer that helps differentiate stores in the database, is loaded as well as the store's name so that the client experience is customized. If a valid username and password combo is not specified, a message is displayed to the user and they are re-prompted for a username and password combo.

Once the user is logged into the system, the database is initialized, and the necessary tables are created if they are not found, or opened if they currently exist. This is accomplished by allocating table handles that allow records to be added or queried in a specific table, based on the table handle provided. If there are no errors initializing the database or opening/creating the tables, the user is directed to the Main Menu. On the Main Menu they are given 6 different choices of functionalities they can explore, and the user is able to provide a number input to navigate to the choice they would

like. If an improper number is entered, the user remains on the menu screen until a valid option is selected. The functionality of the menu options is discussed in greater detail in the Menu Options section.

Menu Options

Option 1: Place an order

If the user selects option 1, a TCP connection is established with the distribution program. This is accomplished by connecting to a socket created by the distribution program that will allow communication between the two programs. If the client is able to successfully connect to the socket, the client program immediately sends the user's store id to the distribution server and the user is then directed to a menu screen where they are prompted to provide an order string that will be sent to the distribution server. Instructions are provided to the user for the structure of the message to be sent to the distribution server, as well as a way to return to the Main Menu. If the user provides an incorrect string, the distribution server is able to recognize that an invalid request was sent and provides the user with an error message. The user is able to specify as many orders as they desire, until they type "terminate" which sends the terminate message to the distribution server so that it may terminate its connection on the server side, and then returns the user to the main menu. If the client provides a valid order request, they receive messages from the distribution server that specify whether the request could be filled, or if there was some error processing the request.

Option 2: View Distribution Warehouse Inventory

If the user selects option 2, the current state of the distribution warehouse's inventory is displayed from the "Distribution Inventory" table of the database. This can help the user determine if the warehouse currently has enough items in stock to fulfill an order. The client program queries the database server using the Distribution Inventory table handle, and displays all rows in the database table. The user can be returned to the main menu by pressing the ENTER key.

Option 3: View Store's Inventory

If the user selects option 3, the current state of the user's store inventory is displayed from the "Store Inventory" table of the database. The client program queries the store inventory table, which contains data for all stores in the distribution warehouse's network. However, since the store inventory table contains a store_id in each row of data, we are able to use the store id acquired at login to display only the rows in the table that pertain to the current user's store. This allows one distribution warehouse to do business with multiple different chains of stores. Because each user can only view data specific to their store, data security is maintained between clients of the distribution warehouse. The user can be returned to the main menu by pressing the ENTER key.

Option 4: View Open Orders

If the user selects option 4, the current state of the user's open orders is displayed from the "Orders To Process" table of the database. Using the same technique described in option 3, the user is only able to view their store's open orders. The user can be returned to the main menu by pressing the ENTER key.

Option 5: View Current Invoices

If the user selects option 5, the current state of the user's invoices is displayed from the "Invoices" table of the database. Using the same technique described in options 3 and 4, the user is only able to view their store's invoices. The user can be returned to the main menu by pressing the ENTER key.

Option 6: Exit

If the user selects option 6, all tables that are open in the database are closed, and the connection with the database server is terminated, and execution of the program is stopped.

Client FlowChart

Figure 2 shows a flowchart of the operation of the client program.

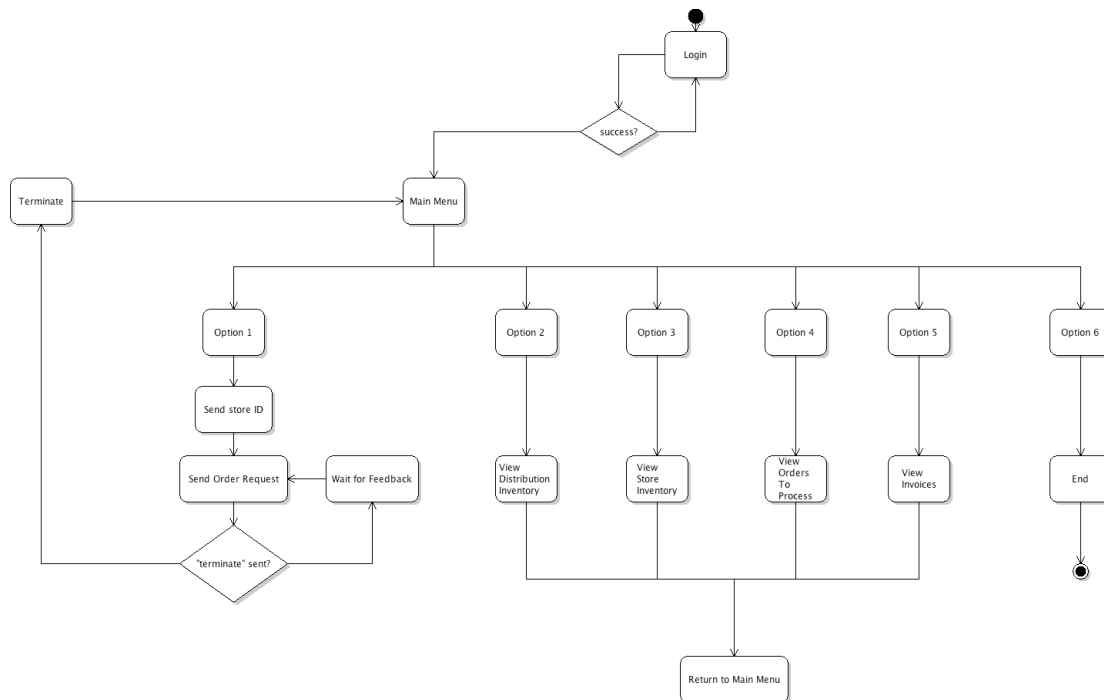


Figure : Flowchar for client program

Distribution

The distribution program begins with a username and password login screen where the user must provide a valid username and password combo to gain access to any of the functionalities of the programs. System login data is stored in a text file called login_Server.txt. The structure of this file is shown in Figure 3.

```

login_Server.txt
(number of entries (n))
(username_1),(password_1)
(username_2),(password_2),
...
(username_n),(password_n)
  
```

Figure :Format of login_Server.txt

Each line of the file is read and tokenized so that the username and password provided can be compared with the system data. If a valid username and password combo is found, the user is given access

to the program. If a valid username and password combo is not specified, a message is displayed to the user and they are re-prompted for a username and password combo.

Once the user is inside the system, a connection is established with the database server, and the necessary tables, "Distribution Inventory", "Store Inventory", "Order To Process", and "Invoices", are opened if found, or created if not found. A TCP communication structure is established and a socket is then created to communicate with clients. The server then listens for connections from clients and as soon as a connection is established, the server performs a fork, and the forked process then calls the function `handle_order()`, which handles all message passing between the clients and the server. The process must fork when a valid connection is established so that more than one client can be serviced at a time. The main process then returns to listen for more clients to connect and repeats the above steps.

As soon as the handle order function is called, the server receives the store id that tells the server which store it is communicating with based off of the store id and this value is saved. The store id is compared against all entries in a file called `clients.txt`, which contains a list of all client store id's as well as the store names that go with each id. A message is then displayed to the user that details what number connection has been made, as well as what store the connection was made with. The structure of the `clients.txt` file is shown below in Figure 4.

```
clients.txt  
(number of entries (n))  
(storeid_1),(storename_1)  
(storeid_2),(storename_2)  
...  
(storeid_n),(storename_n)
```

Figure : Format of clients.txt

The server then listens for messages from the client and acts accordingly based on the messages received. The various messages the server can receive and its reaction to those messages are discussed in greater detail in the Received Messages section.

Since this is designed to be a distribution warehouse server, the clients need to be able to connect to the server at all times. For this

reason, the server is not given an option to stop execution. The program will continue to run and listen for connections from clients until it is exited by force.

Received Messages

“order (product name) (quantity)”

If the server receives a message in this form, the string is tokenized, and the product name is queried in the “Distribution Inventory” table. If the name is found, the quantity requested is checked with the quantity of that product available, and if there are enough units in stock, an order is added to the “Orders To Process” table that includes which store ordered it, what product was ordered, how much of that product, the cost per unit, and the total cost of the order. If there are not enough units in stock to complete the request, the server sends a message to the client that states, “Not enough quantity in warehouse”. If the product is unable to be found in the warehouse, a message is sent to the client that states, “Item not found in warehouse”.

“process”

If the server receives a message that says “process”, all orders waiting to be processed in the “Orders To Process” table are processed for the connected store, and invoices are created for each order processed and placed in the “Invoice” table. The server then returns a message to the client that specifies how many orders were processed successfully.

“terminate”

If the server receives a message that says “terminate”, the socket between the client and server is destroyed, and the connection count is reduced by 1 and the `handle_orders()` function is exited. The process then terminates, with the main process still listening for connections

all other messages

If the server receives a message that says anything other than the above messages, a message is returned to the user that says, “Wrong message format”.

Distribution FlowChart

Figure 5 shows a flowchart of the operation of the distribution program

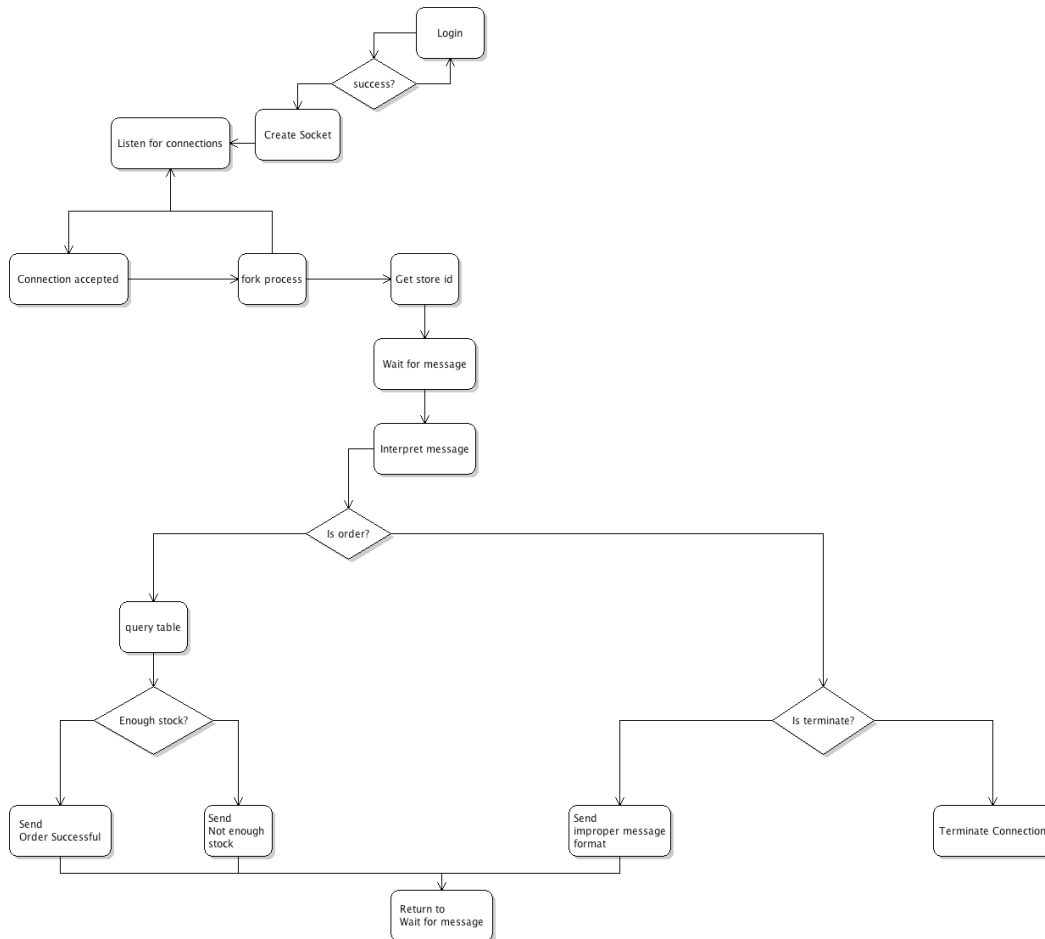


Figure :Flowchart of server program

Experiments and Results

The results of the system are not completely as expected. The functionality of the programs work quite well up until a valid order is placed to the server. When the server attempts to add the order to the "Orders to Process" table, there is some data passing that is made between standard c data types, and the Faircom field types. During this data passing, some of the data is corrupted, or is not well supported in the types of the fields in the table and when the data is inserted, it corrupts the entire table file. I have spent multiple hours trying to figure out the cause of this table being corrupted, but I have been unable to figure out why this happens.

Because of this error, the system cannot be demonstrated as well as I would like. I can show that my logic for displaying only the rows of the tables that pertain to the current user is sound and functioning properly, but I cannot show that orders are populated into the “Orders To Process” table, and therefore I cannot process those orders and place them into the “Invoice” table. This is a very infuriating bug because it corrupts the entire database and will not allow me to work on any other parts of the project because I cannot connect to the database server.

In the next section I will briefly describe the screenshots that would have been a part of this report had the database server not caused such a problem.

Screenshots

Login

This screenshot would have been from both the client and server programs because they both use very similar login functionality. I would have shown that invalid username and password combos do not allow the user entry to the system, and the user is prompted until a valid combo occurs.

Initialization

This screenshot would have been from both the client and server programs showing that each displays messages to the user as the programs are executing necessary operations in the background. The user would be notified that the database server was being accessed, and that the necessary tables were being opened. Then on the server side, the user would also see that the program was listening for connections, and when one was made, a message was given to the user of the form “Connection (number) was established.” Followed by a message that told the user which store they had just connected with.

Main Menu

This screenshot is from the client program and is the user is directed to this screen as soon as they provide a valid username and password combo.

Sending Orders

This screenshot is from the client program and this screen is shown if the user selects option 1 from the Main Menu. This screen allows the client to send messages to the server and displays messages sent back from the server based on the message sent. This screen would have demonstrated that the server is able to decipher whether or not the message sent is of the correct syntax, and is able to send responses to the user that tell the user whether or not what they entered is valid or not.

Receiving Orders

This screenshot is from the server program and shows that the server is able to receive messages from the client program and interpret them correctly, and then send messages back to the client that tell the client whether their order was successful, whether there was enough stock in the warehouse to fulfill their order, or whether the order was of the wrong syntax and disregarded.

View Distribution Inventory

This screenshot is from the client program and this screen is shown if the user selects option 2 from the user menu. The table is printed with all rows of the "Distribution Inventory" table showing to the user. This part is fully functional and independent of the order processing issues.

View Store Inventory

This screenshot is from the client program and this screen is shown if the user selects option 3 from the user menu. The table is printed with all rows of the "Store Inventory" table where the user's store id is a match to the store_id field of the table allowing the user to only view the records that apply to their store. This part is fully functional and independent of the order processing issues.

View Orders To Process

This screenshot is from the client program and this screen would have shown all of the orders waiting to process for the user's store. I have no doubt that this screen would have displayed correctly had the table it was trying to display not been corrupted each time an order was

added to it. I believe so because it follows the logic of options 2 and 3 in the menu, which both work properly.

View Invoices

This screenshot is from the client program and would have shown all open invoices for the user's store. I have no doubt that this screen would have displayed correctly had the table it was trying to display had the opportunity to be filled with invoice data. The data was to come directly from the "Order To Process" table, which is the table that repeatedly was corrupted each time an order was added to it.

Exiting the client

This screenshot would have shown the user entering the option to exit the client program and the messages that were given to the user that signified a proper system shutdown. The user would have seen that all tables opened would have been successfully shutdown and that the connection with the database server was properly disconnected.

Discussion

It is with great displeasure that I was not able to meet the goals of this project. I feel as though I had the knowledge to finish the project and make it a robust and complete project, but was unable to because of the Faircom database system not allowing me to fully test and debug my program. I think that this project was a very good indicator that I have the skills to create full-scale applications that are user-friendly as well as efficient in accomplishing the task at hand, and given more time, I would be able to create a fully functioning product that could be marketed to real businesses.

This project also could be extended to include financial functionality, such as handling of payments from the clients and updating invoice and other financial tables accordingly, as well as provide data for stores on what items are being purchased the most, and how those businesses can price those items effectively so that the most profit can be earned.

Conclusion

In conclusion, this project did not turn out the way that I wanted, but I got some experience with the Faircom database architecture, as well as was able to develop a nice inventory system that could be implemented on an embedded device that would provide mobility and ease of use to the user.