# Proceedings of Conference on Engineering Reasearch, Innovation and Education (CERIE - 2011)

## January 11 - 13, 2011



*Edited by:*
*Professor Dr. Mohammad Iqbal*
*Dr. Mushtaq Ahmed*
*Dr. Md. Jasim Uddin*
*Dr. Md. Mozammel Hoque*
*Mr. Mohammad Alamgir*

*Organized by:*
*School of Applied Sciences and Technology*
*Shahjalal University of Science and Technology*
*Sylhet-3114, Bangladesh*

ISBN: 978-984-33-2140-4

Proceedings of the
Conference on Engineering Research, Innovation and Education 2011
CERIE 2011, 11-13 January 2011, Sylhet, Bangladesh

# INTERFACE DESIGN OF A CLIENTS-SERVER DISTRIBUTED SYSTEM

**Rosina Surovi Khan** *
Ahsanullah University of Science and Technology
141-142 Love Road, Tejgaon Industrial Area, Dhaka-1208, Bangladesh

This paper is based on the interface design of a distributed system that involves clients and one server. Two different write-information clients exert separate signatures to their message lines in message queues and write these messages in blocks to a single message file. The server takes note of the signatures, extracts the appropriate blocks from the file and displays them in its interface under two categories of information. Two corresponding read clients extract their appropriate information from the single file and display their particular information separately in their interfaces. The idea behind this project is to present a distributed system and the message transfers that can take place among clients and server through the usage of message queues, signed messages, threads and a message file all incorporated into one system which previous papers have not demonstrated. The designed system stands out to be a useful and knowledgeable experiment not only for academic purposes but also in respect of research as it demonstrates how all the components taken into account together can work in harmony in a step by step manner and on which further future work can be carried out.

**Keywords:** Distributed system, Threads, Message queues, Signature, Clients-Server interaction

## 1. INTRODUCTION

A distributed system is defined to be a collection of independent computers that appears to users as a single coherent system. In other words, the distributed system is organized as a middleware. The middleware layer extends over multiple machines, and offers each application the same interface. The goals of distributed systems are to connect users and resources and offer distribution transparency, openness, scalability and security. [5]

The work outlined in this paper is based on the interface design of a distributed system that has been implemented in C Sharp using Microsoft Visual C# 2008 Express Edition editor. The experimental work involves the usage of threads along with message queues, signed messages and a message file all incorporated into a clients-server system, something which previous papers have not approached. There are four clients and one server in the system. The server will receive text messages from a parts write client interface that submits parts information of company products and also receive messages from an address write client interface that submits addresses of customers. Message transfers from these clients to the server take place via signed messages with the help of message queues separately. The write clients write these messages to a single file and the server extracts the parts and addresses messages from the file via two threads, one for each type of message and displays them in its separate list boxes. Two read clients will extract parts and addresses message information in list boxes respectively on demand.

Section 2 describes the traditional way in which clients connect with the server. The next section explains the importance of threads and how the designed system uses threads to maintain synchrony among the clients and server. Section 4 goes on to demonstrate how message queues exert signatures on messages sent from the write clients, how they assist in transferring these messages to the server side and how the server seggregates the two type of messages and displays in its interface under separate categories. The next section shows how two read clients extract the content of the messages on demand. Section 6 demonstrates the snapshots of the message file, clients and server interfaces and how they all work together in harmony in a step by step manner. The paper ends on a concluding note of the work achieved in this experiment and the scope for future work.

* Corresponding Author: Rosina Surovi Khan,
E-mail: surovi99@yahoo.com

ISBN: 978-984-33-2140-4

## 2. CLIENTS-SERVER CONNECTION

The Client Server model is usually based on a simple, connection-oriented request/reply protocol. The client sends a request and gets an answer. The reply message serves as the acknowledgement to the request. [6]

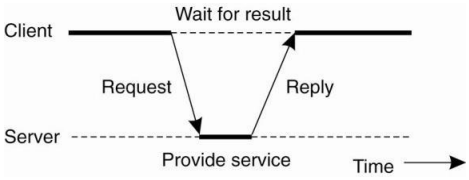


Fig. 1: General interaction between a client and a server

Each client binds to the server through a communication endpoint called socket. The socket listens and binds to a local IP address. The server waits for requests and finally accepts client connection request. Each client gets remote IP address and connects to remote server host via socket. [5]

Writing data from the server to a client's interface is equivalent to sending some data to the client and displaying in the client's interface and vice versa. Reading data from a client means receiving data from the client to the server and displaying in the server's interface and vice versa.

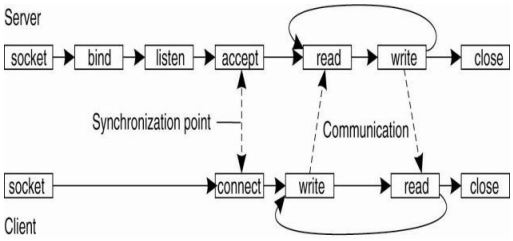


Fig. 2: Connection-oriented communication pattern using sockets

## 3. USAGE OF THREADS

Threads are like little mini-processes that were invented to allow parallelism to be combined with sequential and blocking system calls. Blocking system calls make programming easier and parallelism improves performance. [6]

In the designed system, two user threads are created. One is used so that the server can read parts message blocks from a single message file to which the parts write client writes. The other thread is for the server to read address message blocks from the same file to which the address write client submits address messages.

Using threads this way, a synchronous order is maintained in which the two types of write clients write to the message file along with the server reading from the message file and displaying their messages in its interface. Otherwise without the usage of threads this harmony is impossible to achieve.

## 4. INTERACTION BETWEEN WRITE CLIENTS AND SERVER

Asynchronous persistent communication is achieved through the support of middleware-level message queues. Queues correspond to buffers at communication servers. Basic interface to a queue in a message-queuing system is summarized in the following table. [5]

**Table 1**. Basic interface to a message queue

| Primitive | Meaning |
|---|---|
| Put | Append a message to a specified queue |
| Get | Block until the specified queue is nonempty, and remove the first message |
| Poll | Check a specified queue for messages, and remove the first. Never block |
| Notify | Install a handler to be called when a message is put into the specified queue |

In the system parts write client before sending its message to parts messge queue adds a signature "Queue1" to parts message line to differentiate it from address message blocks. Similarly address write client adds the signature "Queue2" to address message line to differentiate it from parts message blocks.

//Signature "Queue1" is added to parts message to differentiate from address message blocks

```
partsItems.Message="Queue1" +String.Format( "\r\n ")+"Part Desc " + part_desc_txt.Text + String.Format( "\r\n") + "Quantity " + qty_txt.Text + String.Format( "\r\n") + "Part No " + part_no_txt.Text + String.Format("\r\n")+ "Price per Each " + price_each_txt.Text;

mq.Send(partsItems);
```

Code 1 [2]: Addition of signature to parts queue

Parts queue writes parts messages to a message file from which the server attempts to read via a thread. Similarly address queue writes address message

blocks to the same file from which the server once more attempts to read via another thread. From this single file, the server checks if a block begins with the signature "Queue1" or Queue2". If the block begins with Queue1 the server extracts it from the file and displays it in its parts list box. If a block begins with Queue2 the server extracts and displays in its address list box. The write clients can send messages to the server side in any order.

```
//creates an instance MessageQueue, which points
//to the already existing MyQueueParts
if(MessageQueue.Exists(@".\Private$\
                                    MyQueueParts"))
     mq1=new System.Messaging.MessageQueue
                 (@".\Private$\MyQueueParts" );
else
 //creates a new private queue called MyQueueParts
     mq1=MessageQueue.Create(@".\Private$\
                                    MyQueueParts");
     mq1.Formatter = new System.Messaging.
        XmlMessageFormatter(new Type[]{typeof
                                    (WriteItems)}) ;
 while (true)
   {
     System.Messaging.Message m1;
     // checks if parts queue mq1 has messages
     m1 = mq1.Peek();
     System.Messaging.Message msg1 =
                                    mq1.Receive();
      WriteItems outpartsItems =
                        (WriteItems)msg1.Body;
      //Writing messages from parts queue to file
      using (StreamWriter sw1 = new
      StreamWriter(outpartsItems.FileName, true))

        {   sw1.WriteLine(outpartsItems.Message);
           sw1.Flush(); }

       using(StreamReader sr1 = new
       StreamReader(outpartsItems.FileName, true))
     { int queue = 0;
       string line1;
//Read from the file and display parts queue
//message blocks in server's parts list box until the
//end of the file is reached
     while ((line1 = sr1.ReadLine()) != null)
        { if (line1.StartsWith("Queue1"))
             { queue = 1;}
          else if (line1.StartsWith("Queue2" ))
             { queue = 2;}
          else if (queue == 1)
             {
           Client_Parts_Records.Items.Add(line1) ;
             }
          }
        }
        mq1.Close();
```

Code 2 [4]: Interaction between parts write client and server

## 5.INTERACTION OF READ CLIENTS IN THE SYSTEM

Parts read client reads four lines at a time from the single file that the parts write client has previously written to. The read client checks if this block starts with the signature "Queue1". If it does, the client extracts it and displays it in its interface's list box. The same goes for the address read client which will check if a block starts with the signature "Queue2". The read clients can read messages in any order.

```
String buffer;

using (StreamReader stream = File.OpenText
          (@"C:\rsk8332\Project2\Server\Msg.txt"))
        { buffer = stream.ReadToEnd(); }

//Stores into each index of lines array the individual
fields and their values of parts block

string[] lines = buffer.Split(new string[] { "\r\n" },
          StringSplitOptions.RemoveEmptyEntries);

string[] result = new string[4];

if (lines[i].StartsWith("Queue1" ))
{
 //Copies into result 4 lines of each parts block
  Array.Copy(lines, i + 1, result, 0, 4);
  Client_Read_Parts_Records.Items.AddRange
                                    (result);
  i = i + 5;
}
else
{ i = i + 4; }
```

Code 3 [1,3]: Interaction of parts read client in the system

## 6. INTERFACE DESIGN SNAPSHOTS

Initially the message file to which the write clients write may have information as shown in Fig. 3.

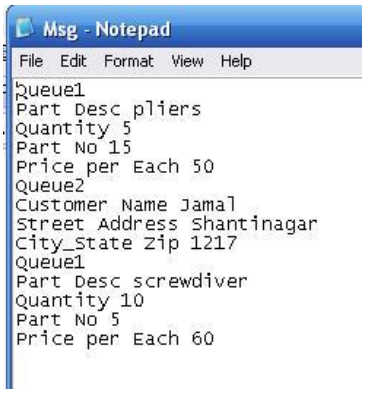


Fig. 3: Single Message File

Extracting information from the message file, the Server Graphical User Interface (GUI) when it is started shows parts and address information in separate list boxes as shown in Fig 4.

A "parts" block message is submitted via Parts Write Client GUI as shown in Fig. 5. An "address" block message is submitted via Address Write Client GUI in the same way. Accordingly, the Server GUI shows in its separate list boxes both newly submitted parts and address message blocks from the corresponding write clients by reading from the single message file as shown in Fig. 6. Parts Read Client GUI reads all the parts information from the single message file as shown in Fig. 7. Address Read Client GUI reads all the address information from the message file in the same way. The write and read clients can interact with the server via the message file in any order. In this way, message transfers take place during clients-server interaction in the designed system. This synchrony has been possible to achieve through the combined usage of a message file, threads, signed messages and message queues as explained in the earlier sections.

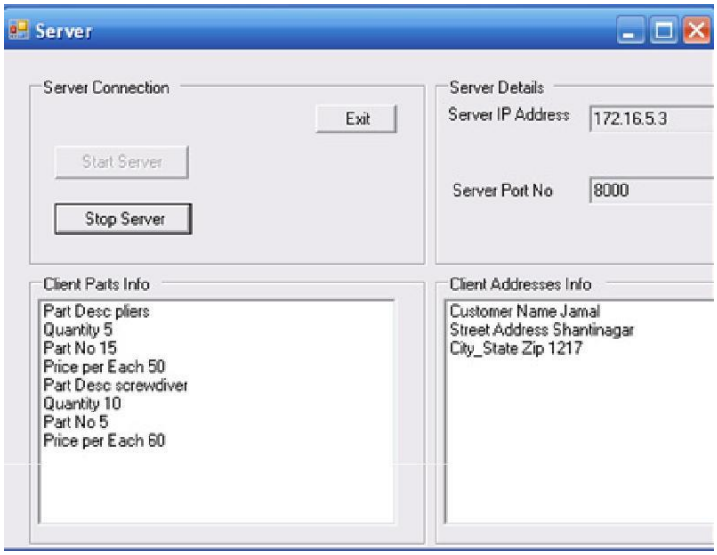


Fig. 4: Server Graphical User Interface (GUI)

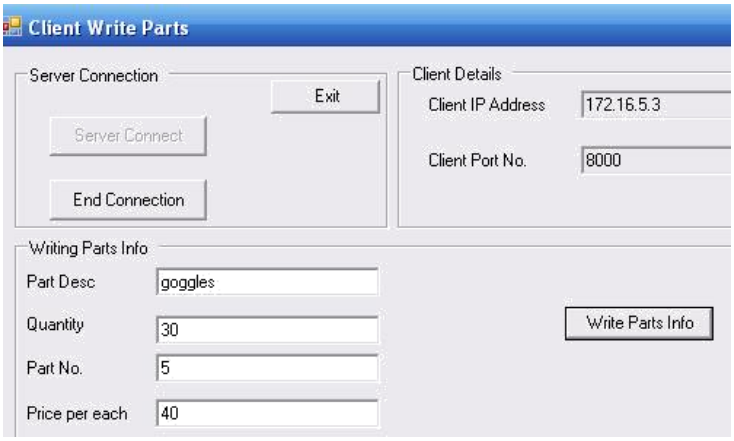


Fig. 5: Parts Write Client GUI

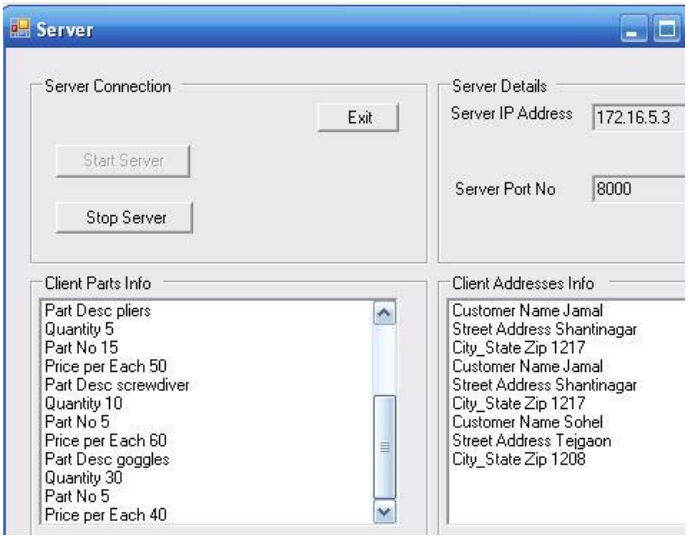


Fig. 6: Server GUI showing updated info

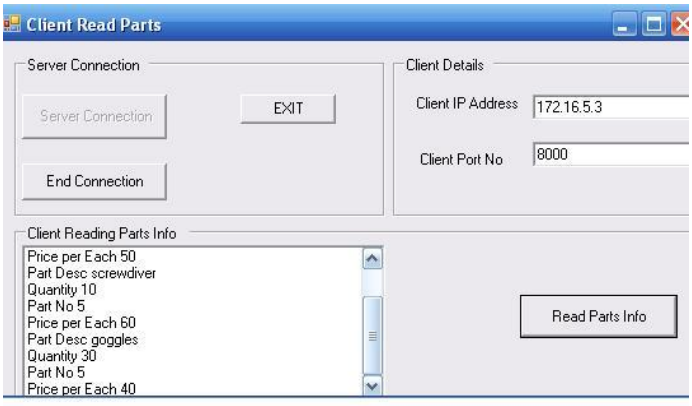


Fig. 7: Parts Read Client GUI

## 7. CONCLUSION AND FUTURE WORK

In the designed system, write clients use message queues to transfer their messages to the server end, while attaching signatures to the beginning of message lines with the server reading the messages from a file under separate threads and using the signatures to identify the message blocks while extracting them from the file, and while read clients can utilize the same signatures in the same file to extract their corresponding messages. This is the subtleness of the experimental work, the system's homogeneity details being hidden from a high level. Thus the use of signatures in message lines, a message file, message queues, the utilization of threads and overall clients-server interaction are components of the designed system which make the system a good and useful implementation, giving insights on distributed systems by taking into account the collective components working in a series of coordinated steps. The work can be extended for future scope to take into account issues of security such as encryption and authentication during transfer of messages between clients and server.

## REFERENCES


[1] Allen S. (2010), Copying Arrays, <http://dotnetperls.com/array-copy>, Accessed June 2010

[2] Microsoft Support (2007), Message Queuing, <http://support.microsoft.com/kb/815811>, Accessed June 2010

[3] Mojica J. (2003), Splitting Arrays, Peachpit Press, <http://www.peachpit.com/articles/article.aspx?p=31938&seqNum=14>, Accessed June 2010

[4] Strawmyer M. (2004), Message Queuing, Code Guru Site: <http://www.codeguru.com/Csharp/Csharp/cs_misc/article.php/c4241/>, Accessed June 2010

[5] Tanenbaum A. S. (1995), Distributed Operating Systems, Pearson Education, Inc.

[6] Tanenbaum & Steel V. (2007), Distributed Systems: Principles and Paradigms, Prentice Hall, Inc.