

# MULTIPLATFORM INFORMATION ACCESS

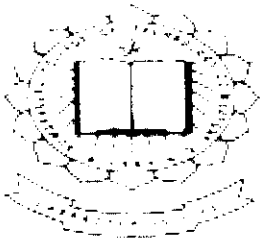
## PROJECT REPORT

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE  
AWARD OF THE DEGREE OF

## BACHELOR OF ENGINEERING

OF BHARATHIAR UNIVERSITY.

COIMBATORE.

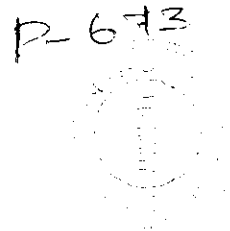


Submitted by

**M. JAYAPRABHU  
S. KARTHIK  
M.S. PRABAKARAN  
S. SRIRAM**

Guided by

**Mrs.D.Chandrakala M.E**



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

**KUMARAGURU COLLEGE OF TECHNOLOGY**

COIMBATORE 641 006

MARCH 2002

P-673

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

**KUMARAGURU COLLEGE OF TECHNOLOGY**

(Affiliated to Bharathiar University, Coimbatore)

**CERTIFICATE**

This is to certify that project report entitled

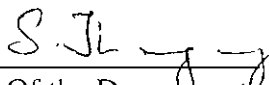
**MULTIPLATFORM INFORMATION ACCESS**

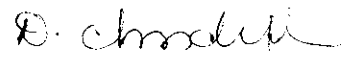
Is a bonafide record of work done by

<b>M JAYAPRABHU</b>	<b>9827KO176</b>
<b>S KARTHIK</b>	<b>9827KO179</b>
<b>M S PRABAKARAN</b>	<b>9827KO195</b>
<b>S SRIRAM</b>	<b>9827KO218</b>

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE AWARD OF  
THE DEGREE OF


**BACHELOR OF ENGINEERING**

  
Head Of the Department 12/3/02

  
Staff-in-charge

Submitted for the University Examination held on \_\_\_\_\_

  
Internal Examiner

  
External Examiner

Place : Coimbatore

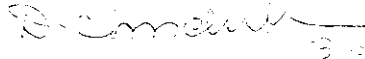
Date :

# DECLARATION

We, M.Jayaprabhu, S.Karthik, M.S Prabakaran, S.Sriram here by declare that this project entitled "MULTIPLATFORM INFORMATION ACCESS" submitted to **Kumaraguru College of Technology, Coimbatore** (Affiliated to **Bharathiar University**) is a record of original work done by us under the supervision and guidance of **Mrs.D.Chandrakala M.E.**, Senior Lecturer, Department of Computer Science & Engineering.

NAME	REGISTRATION NUMBER	SIGNATURE
M.Jayaprabhu	9827K0176	
S.Karthik	9827K0179	
M.S.Prabakaran	9827K0195	
S.Sriram	9827K0218	

Countersigned:

  
13.03.2002

Staff in charge: **Mrs.D.Chandrakala M.E.**,

Senior Lecturer,

Department of Computer Science & Engineering,

Kumaraguru College of Technology, Coimbatore.

Place : Coimbatore.

Date : 13.03.2002

*Dedicated to our beloved parents  
and parting Friends...*



## *ACKNOWLEDGEMENT*

---

# ACKNOWLEDGEMENT

---

We take this opportunity to thank everyone who contributed to our project and their efforts are commemorative.

We are extremely grateful to **Dr.K.K.Padmanaban B.Sc.(Engg), M.Tech., Ph.D.**, Principal, Kumaraguru College of Technology for having given us his valuable guidance and useful suggestions during the course of our education.

We are deeply obliged to **Dr.S.Thangaswamy Ph.D**, Prof & Head of Department of Computer Science and Engineering his valuable guidance and useful suggestions during the course of this project.

We extend our thanks to our project coordinator **Mrs.S.Devaki B.E.,M.S**, Assistant Professor, Department of computer Science and Engineering, for providing us her support which really helped us to come out with this project.

We are indebted to our project guide **Mrs.D.Chandrakala M.E**, Senior Lecturer, Department of computer Science and Engineering, for her helpful guidance and valuable support given to us throughout this project.

Above all we owe our gratitude and heart felt thanks to our parents, our friends, all teaching and non-teaching staffs for their encouraging support and Almighty for His abundant blessing for finishing this project successfully in time.



## *SYNOPSIS*

---

# SYNOPSIS

---

This project deals with the implementation of a Multiplatform Information Access. The objective of this process is to access the database created in the Linux server from Microsoft Windows machine using a front end in Windows, the operations to be performed on the database are obtained. We tackle the above problem by using sockets as the means of communication between the Linux and the Microsoft Windows machine. MySQL is the database used in Linux. This is implemented by creating a server in Linux and a client in Microsoft Window. The client accepts the request from the user and sends the request to the server that responds by accessing the database. The client provides the user interface in the Windows programming environment.





# *CONTENTS*

---

# **CONTENTS**

---

<b>1. Introduction</b>	<b>1</b>
1.1 Existing system and its limitations	2
1.2 Proposed system and its advantages	2
<b>2. System Requirements</b>	<b>25</b>
2.1 Product Definition	25
2.2 Project Plan	25
<b>3. Software Requirements Specification</b>	<b>28</b>
<b>4. Design Document</b>	<b>34</b>
<b>5. Product Testing</b>	<b>38</b>
<b>6. Future Enhancements</b>	<b>41</b>
<b>7. Conclusion</b>	<b>43</b>
<b>8. References</b>	<b>45</b>
<b>9. Appendix</b>	<b>47</b>
9.1 Sample source code	48
9.2 Sample Outputs	84



# *INTRODUCTION*

---

# 1. INTRODUCTION

---

## 1.1 Current System:

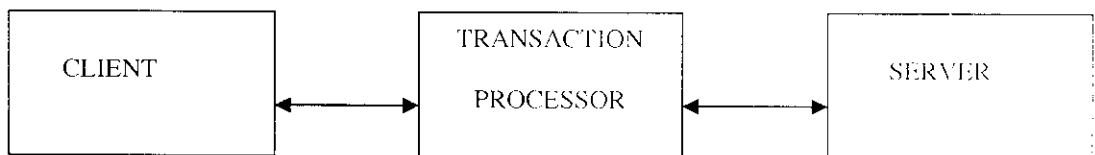
A large number of useful data has been stored in LINUX database servers around the world. But nowadays, Windows based machines are becoming more prevalent. It is out of the question to physically transfer all the data from LINUX machines to Windows systems as it involves huge manpower and computational time.

## 1.2 Proposed System:

So a means to use the databases existing in LINUX directly from Windows without any physical transfer of data is implemented. This implementation is called the transaction processor. The Berkeley sockets do the data transfer. A front end is designed to meet the scratchy thoughts of the user when user queries the database.

A large amount of useful data has been stored in LINUX database servers around the world. But nowadays, Windows based machines are becoming more prevalent. It is impossible to

physically transfer all the data from LINUX machines to Windows systems as it involves huge manpower and computational time. So a means to use the databases existing in LINUX directly from Windows without any physical transfer of data is implemented. This implementation is called the Transaction Processor (TP).



*Block Diagram of the proposed approach*

In this implementation, all the transactions are predefined as a unit of work called service. The server is a multithreaded process, waiting for the client's request. It has a listening thread, which is the main thread that will accept the client's request and dispatch it to the appropriate service. The service is a C function, which in turn invokes other C/C++, SQL functions. Thus there are two units, the Listener and the Service Dispatcher.

The implementation of the project consists of three phases

1. Creating a Server process in LINUX

2. Creating a client process in Windows
3. Creating an interface to MySQL in LINUX.

The server receives a request from the front end Graphical User Interface (GUI) based client and then processes the request by accessing one or more back end Resource managers. Generally the Resource manager is a database file server or some other form of data provider.

## **1.2 SOCKET IMPLEMENTATION IN LINUX**

### **1.2.1 SOCKETS – AN OVERVIEW**

A socket is an object used to send and receive packets of data. It describes a file handle, which is a positive integer value that identifies the endpoint for communication. The data is buffered both by server application sending the data and client application receiving the data. Each socket has a type and is associated with a running process. A socket is not the same as TCP port value. Instead it is a handle to the data that includes IP addresses as well as a port addresses doing the communication.

Socket Programming can be broken down into two parts:

- What the client process does
- What the server process does

The client creates a socket, binds the socket to the local address and the port, connects to server, sends/ receives data and then closes the socket. The server creates the socket, binds to its address and the port and sits and listens for the client to start

something. After the client connects, the server accepts by creating a new socket for the client's use. The original socket waits for a second client to connect. The new socket performs the receive / send operations with the client and then closes.

The basic concept is that two systems are communicating and needs to know the socket numbers of each other. Each sends messages with this socket number.

Sockets are of two types:

- Stream Sockets
- Datagram Sockets

A stream socket is a bi-directional stream of bytes delivered in the correct sequence and each packet is received only once. Datagram sockets are record oriented, may be duplicated and are not guaranteed to be delivered in any specific sequence.

### **1.2.2 BERKELEY SOCKETS**

The API (Application Programming Interface) is the interface to the programmer for the communication protocols. The availability of



an API depends on both the operating system being used and the programming language.

The two most prevalent communication APIs for LINUX system are

1. Berkeley sockets
2. System Vs Transport Layer Interface.

### **1.2.3 Comparison of Network I/O and File I/O:**

The six system calls open, create, close, read, write, lseek used for file I/O work with the file descriptor. It would be nice if the interface to the network facilities maintained the file descriptor semantics of the LINUX file system, but network I/O involves more details and options.

- The typical client server relationship is not symmetrical.
- A network connection can be connection oriented or connectionless
- Names are more important in networking than for file operations.

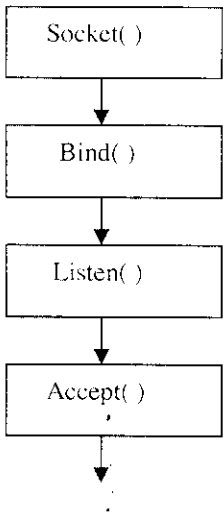
- A network association needs the following five parameters. {protocol, local\_addr, local process, foreign\_addr, foreign process}
- For some communication protocols, record boundaries have significance.
- The network interface should support multiple communication protocols.

#### **1.2.4 Overview:**

The original 4.3 BSD VAX release from 1986 supported the following communication protocols.

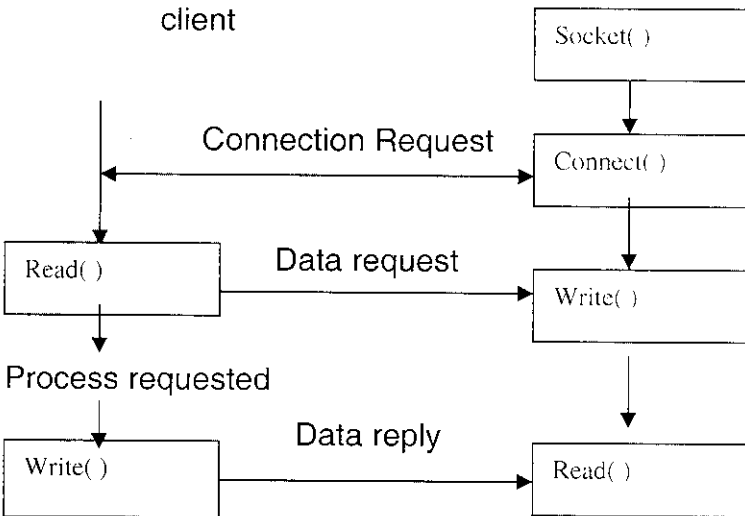
- LINUX domain
- Internet domain
- Xerox NS domain

A time line of the typical scenario that takes place for a connection-oriented transfer is shown. First the server is started, and then sometime later a client is started that connects to the server.



Blocks until connection from client

client



*Socket system calls for connection oriented protocols*

## 1.2.5 SOCKET ADDRESS:

Many of the BSD networking system calls require a pointer to a socket address structure as an argument. The definition of this structure is in `<sys/socket.h>`.

```
Struct sockaddr
{
    u_short sa_family;    /*address family: AF_XXX
value*/
    char sa_data[14]; /*upto bytes of protocol-specific
address*/
};
```

The contents of the 14 bytes of protocol specific address are interpreted according to the type of the type of address.

For the Internet family, the following structures are defined in `<netinet/in.h>`.

```
Struct in_addr
{
    u_long s_addr;    /*32 bit netid /hosted*/
                    /*network byte ordered*/
};
Struct sockaddr_in
```

```

{
    short sin_family;      /* AF_INET*/
    u-short sin_port;     /* 16 bit port number*/
    struct in_addr sin_addr; /* 32 bit netid/hosted*/
                                /* network byte ordered*/
    char sin_zero[8];     /* unused*/
};

```

## 1.2.6 ELEMENTARY SOCKET SYSTEM CALLS:

Socket system call:

To do network I/O, the first thing a process must do is call the socket system call, specifying the type of communication protocol desired (Internet TCP, Internet UDP, XNS, SPP, etc..).

```
int socket(int family, int type, int protocol);
```

The family is one of

AF_LINUX	LINUX internal protocols
AF_INET	Internet protocols
AF_NS	Xerox NS protocols
AF-IMPLINK	IMP link layer

P-673



The AF\_ prefix stands for “address family”. There is another set of terms that is defined starting with a PF\_ prefix which stands for “protocol family”: PF\_LINUX, PF\_INET, PF\_NS and PF\_IMPLINK. Either term for a given family can be used as they are equivalent.

The socket type is one of the following

SOCK_STREAM	stream socket
SOCK_DGRAM	datagram socket
SOCK_RAW	raw socket
SOCK_SEQPACKET	sequenced packet socket
SOCK_RDM	reliably delivered message
Socket(not yet completed)	

The socket system call specifies one element of this 5 tuple, the protocol. The socket system call returns a small integer value, similar to a file descriptor. This is called a socket descriptor, or a sockfd.

## Bind System Call:

The bind system call assigns a name to unnamed socket.

```
int bind(int sockfd, struct sockaddr *myaddr, int  
addrlen);
```

The second argument is a pointer to a protocol specific address and the third argument is the size of this address structure. The bind system call fills in the local process elements of the association 5 tuple.

## Connect System call:

The client process connects a socket descriptor following the socket system call to establish a connection with a server.

```
int connect(int sockfd, struct sockaddr *servaddr, int  
addrlen);
```

The sockfd is the socket descriptor that was returned by the socket system call. The second and the third arguments are a pointer to a socket address and its size. The client does not have to bind a local address before calling connect.

Listen system call:

This system call is used by a connection oriented server to indicate that it is willing to receive connection.

```
int listen(int sockfd, int backlog);
```

It is usually executed after both the socket and bind system calls and immediately before the accept system calls. The backlog argument specifies how many connection requests can be queued by the system while it waits for the server to execute the accept system call. This argument is usually specified as 5, the maximum value currently allowed.

Accept system call:

After a connection oriented server executes the listen system call described above, an actual connection from some client process is waited for, by having the server execute the accept system call.

```
int accept(int sockfd, struct sockaddr *peer, int *addrlen);
```

Accept takes the first connection request on the queue and creates another socket with the same properties as sockfd.



there are no connections requests pending, this call blocks the caller until one arrives.

The `peer` and `addrlen` arguments are used to return the address of the connected peer process( the client ), `addrlen` is called a value result argument. This system call returns three values: an integer return code that is either an error indication or a new socket descriptor, the address of the client process(`peer` )and the size of this address(`addrlen`). `Accept` automatically creates a new socket descriptor, assuming that the sever is a concurrent server.

Close system call:

The normal LINUX close system call is also used to close a socket.

```
int close(int sockfd);
```

If the socket being closed is associated with a protocol that promises reliable delivery (e.g.: TCP or SPP), the system must assure that any data within the kernel that still has to be transmitted or acknowledged is sent. Normally the system returns from the

close call immediately but the kernel still tries to send any data already queued.

### **1.2.7 BYTE ORDERING ROUTINES:**

The following four functions handle the potential byte order differences between different computer architectures and different network protocols.

```
u_long htonl(u_long hostlong);
```

```
u_short htons(u_short hostshort);
```

```
u_long ntohl(u_long netlong);
```

```
u_short ntohs(u_short netshort);
```

htonl convert host\_to\_network, long integer.

htons convert host\_to\_network, short integer.

ntohl convert network\_to\_host, long integer.

ntohs convert networkto\_host, short integer.

### **1.2.8 ADDRESS CONVERSION ROUTINES:**

An Internet address is usually written in the dotted decimal format eg 192.168.12.5. The following functions convert between the dotted decimal format and in\_addr structures.

```
unsigned long inet_addr(char *ptr);
```

```
char *inet_ntoa(struct in_addr inaddr);
```

The first of these, `inet_addr` converts a character string in dotted decimal format notation to a 32 bit Internet address. The `inet_ntoa` function does the reverse conversion.

### 1.2.9 RESERVED PORTS:

There are two ways for a process to have an Internet Port or an XNS port assigning to a socket.

- The process can request a specific port.
- The process can let the system automatically assign a port.

BSD provides a library function that assigns a reserved TCP stream socket to the caller.

```
int rresvport(int *aport);
```

The function creates an Internet stream socket and binds a reserved port to the socket.

Assignment of ports in the Internet Domain

Reserved Ports	1-1023
Ports automatically assigned by system	1024-5000
Ports assigned by <code>rresvport()</code>	512-1023.

## **1.3 SOCKET IMPLEMENTATION IN WINDOWS**

### **1.3.1 WINDOWS SOCKETS : PORTS AND SOCKET ADDRESS**

#### **Port**

A port identifies a unique process for which a service can be provided. In the present context, a port is associated with an application that supports Windows Sockets. The idea is to identify each Windows Sockets application uniquely so that it is possible to have more than one Windows Sockets application running on a machine at the same time. Certain ports are reserved for common services, such as FTP. The Windows Sockets specification details these reserved ports. The file WINSOCK.H also lists them. To let the Windows Sockets DLL select a usable port, zero is passed as the port value. MFC selects a port value greater than 1024 decimal. The port value that MFC selected by calling the `CAsyncSocket :: GetSockName` member function.

#### **Socket Address**

Each socket object is associated with an Internet Protocol(IP) address on the network. Typically, the address is a machine name, such as `ftp.microsoft.com`, or a dotted number, such as "192.168.12.6". To create a socket it is necessary to specify our

own address. It's possible that the machine has multiple network cards, each representing a different network. If so, it might be necessary to give an address to specify which network card the socket will use. This is certain to be an advanced usage and a possible issue.

### **1.3.2 WINDOWS SOCKETS: BACKGROUND**

The Windows Sockets specification defines a binary-compatible network-programming interface for Microsoft Windows. Windows Sockets are based on the LINUX sockets implementation in the Berkeley Software Distribution (BSD, release 4.3) from the University of California at Berkeley. The specification includes both BSD-style socket routines and extension specific to Windows. Using Windows Sockets permits the current application to communicate across any network that conforms to the Windows Sockets API. On Win32, Windows Sockets provide thread safety.

The Microsoft Foundation Class Library (MFC) supports programming with the Windows Sockets API by supplying two classes, Csocket, provides a high level of abstraction to simplify network communications programming.

## Definition of a Socket

A socket is a communication endpoint—an object which a Windows Sockets application sends or receives packets of data across a network. A socket has a type and is associated with a running process, and it may have a name. Currently, sockets generally exchange data only with other sockets in the same “communication domain”, which uses the Internet Protocol Suite.

Both kinds of sockets are bi-directional: they are data flows that can be communicated in both directions simultaneously (full-duplex).

Two socket types are available

- Stream sockets

Stream sockets provide for a data flow record boundaries – a stream of bytes that can be bi-directional (the application is full-duplex: it can both transit and receive through the socket). Streams can be relied upon to deliver sequenced, unduplicated data. (“Sequenced” means that packets are delivered in the order sent. “Unduplicated” means that a particular packet can be sent only once.) Receipt of stream messages is guaranteed, and streams are well-suited to handling large amounts of data. The network transport

layer may break up or group data into packets of reasonable size. The CSocket class handles the packing and unpacking. Stream are based on explicit connections: socket A requests a connection to socket B; socket B accepts or rejects the connection requests.

- Datagram sockets

Datagram sockets support a record-oriented data flow that is not guaranteed to be delivered and may not be sequenced as sent or unduplicated. “Sequenced” means that packets are delivered in the order sent. “Unduplicated” means that it is possible to get a particular packet only once.

Uses for sockets

Sockets are highly useful in at least three communications contexts:

- Client/Server models
- Peer-to-Peer scenarios, such as chat applications

- Making remote procedure calls (RPC) by having the receiving applications interpret a message as a function call.

## **1.4 IMPLEMENTATION**

In this project, a client running on Windows is used to access data present in the MySQL database, through a server program running on LINUX.

The implementation consists of three steps:

1. Creating a Server process in LINUX
2. Creating a Client process in Windows
3. Creating an interface to MySQL database in LINUX

### **1.4.1 CREATING A SERVER PROCESS IN LINUX:**

The Server program is a multi-threaded program, which performs three major functions.

First, it creates a socket, which is an endpoint of connection. The socket created is bound to the address of the local host. After this is done, the server listens for requests from the clients. Secondly, it formats the message, which is to be sent through the



socket to the client. Concatenating the data items received from the database along with their respective lengths forms this message. This concatenated message is sent through the socket to the client. The Server Program also receives the request from the client through the socket, determines what service is necessary and accordingly calls the function used to access the database. Thirdly, the server performs the important function of accessing the database. The MySQL database is accessed from the server program using the SQL commands. The SQL commands are included as part of the program.

#### **1.4.2.CREATING THE CLIENT PROCESS IN WINDOWS:**

The client is created in Windows 2000 using VC++ platform SDK.

The operations performed by the client are

- A socket is created and bound to the host address of the machine.
- Using the connect system call the client tries to connect to the server.

- After the connection is established, the client sends the requests obtained from the user to the server through the socket.



# *SYSTEM REQUIREMENTS*

---

## **2. SYSTEM REQUIREMENTS**

---

### **2.1 Product Definition**

When data transfer comes into picture oriented network it consumes a lot of time when the database from the LINUX platform to Windows. This scenario is replaced by accessing the database from the LINUX platform to Windows using queries using MYSQL.

To be beneficiary, space requirements are reduced and time for data transfer is also reduced. In order to avoid the confusion over the queries provided by the user to access the database, a suitable front end is designed which effectively defines controls as well as retrieves the information.

The transfer is not edged only for peers but multiclients are comprised in.

### **2.2 Project Plan**

In the analysis phase requirements for Multiplatform Information Access like functions, user interface prototype and certain aspects that eliminate the ambiguity were identified.

From the results obtained from the analysis phase system design was established where the complete system was depicted in the form of an event flow diagram.

In the implementation phase coding is written for each related module and finally they are integrated.

In the testing phase all the necessary tests are performed to test the validity of the product developed.

Multiplatform Information Access will be implemented as a tool where it must be running in the client machine as well as the server to obtain the functionality of the product developed.



*SOFTWARE REQUIREMENTS  
SPECIFICATION*

---

# 3. SOFTWARE REQUIREMENTS SPECIFICATION

---

## 3.1 Introduction

### 3.1.1 Purpose

The purpose of this SRS is to elaborate the requirements of Multiplatform Information Access. Multiplatform Information Access would bring out the interface between two platforms namely Linux and Microsoft Windows, access of the database in Linux from the later with a newly designed front end. The Front end is designed in Microsoft Visual C++ and the server program is written in Linux based C and executed PCs.

### 3.1.2 Scope

This SRS focuses mainly on the requirement to be met by the Multiplatform Information Access. The well-structured format and descriptive nature of the SRS is aimed at aiding the developers in developing an efficient Information Access.

### **3.1.3 Overview**

Exploring further the Software Requirement Specification gives information about the specific requirements like functionality, usability and supportability of the Multiplatform Information Access.

### **3.1.4 Overall Description**

The Multiplatform Information Access is a descriptive software in which the user can access the database in a different platform namely Linux effectively with many constraints.

The database mentioned above is a Linux based database called MySQL. The objective of the Multiplatform Information Access ends with listing out the menus and it is up to the user to effectively use the database.

## **3.2 Specific Requirements**

### **3.2.1 Functionality**

The functionality of the Multiplatform Information Access can be explained using the enclosed functional diagram.



### **3.2.2 Data Definition Language (DDL):**

DDL commands allow modification of the database structure by creating, replacing, altering or removing database objects such as tables.

### **3.2.3 Data Manipulation Language (DML):**

DML commands allow the user to manipulate data. Manipulation of data refers to insertion, updating and deletion of data.

### **3.2.4 Data Retrieval Command:**

This category contains only one command-SELECT. The command is self explanatory in what it does. It selects the requested data from the table and displays it to the user on the screen. It's one the most widely used command and therefore considered as a category itself.

### **3.2.5 Supportability**

Throughout the coding Java standard naming and coding conventions are followed. The runtime files generated is platform dependant and a version for every platform must be released.

### **3.2.6 Design Constraints**

- Languages having packages, to analyze the running instances, are imminent.
- Runtime Environment is mandatory for the Multiplatform Information Access to operate.
- Queries are avoided by the customization of the user interface.

## **3.3 Interfaces**

### **3.3.1 User Interfaces**

User interface is Graphical based for better interpretation. GUI is customized to the user such that the remembrance of the commands is not needed.

### **3.3.2 Software Interfaces**

The API (Application Program Interface) is the interface available to the programmer for the communication protocols. The availability of an API depends on both the Operating System being used and the programming language.

The two most prevalent communication APIs for Linux system are

- Berkeley Sockets
- System Vs Transport Layer Interface



# *DESIGN DOCUMENTS*

---

# 4. DESIGN DOCUMENTS

---

## 4.1 Introduction

The Software Architecture Document lists the various issues concerning the architectural design phase of the software lifecycle process. It provides a detailed explanation of the purpose, scope, definitions, abbreviations, concepts and references encountered in the design phase. The beneficiaries of this document are intended to be:

- a) The Design team which uses this document as a guide for the design process.
- b) The Implementation team which carries out all implementation tasks and development of the user interface based on the guidelines provided in this document.
- c) The Testing team which tests the developed software system and checks if it conforms to the original specifications and design motives.

## **4.2 Purpose**

This document provides a comprehensive architectural overview of the system, using a number of different architectural views to depict different aspects of the system. It is intended to capture and convey the significant architectural decisions, which have been made on the system and which are to be implemented. The reader of this document is equipped with a number of object and sequence diagrams which depict the control flow in the designed system and the various state transitions which the system passes through.

## **4.3 Scope**

The contents of this document cover the entire spectrum of design related issues and they influence the implementation process, the data structures used in the source code and the various classes, methods and functions present in the source code. This manuscript serves as a formal guideline for the design process and methodology.

## **4.4 Overview**

The remaining portion of the Software Architecture Document contains details about the architectural representation and also detailed explanations about the various views of the designed system. References to the quality measures adopted are also made available.



# *PRODUCT TESTING*

---



# 5. PRODUCT TESTING

---

## 5.1 Unit Testing

Unit testing focuses verification effort on the smallest unit of software product developed – software component or module. The different modules that were developed in MIA are GUI, user interface, login, client – server communication and client – server database operations were the modules that were developed to implement MIA.

In the unit testing following were examined to ensure the reliability and correctness of each module

- Local data structure.
- Boundary conditions.
- Data flow.

Some of common errors were detected and corrected namely

- Incorrect arithmetic precedence.
- Incorrect initialization.
- Casting.
- Improperly modified loop variables.

## 5.2 Integration Testing

Integration testing is a systematic approach for constructing the program structure while at the same time conducting tests to uncover errors associated with interfacing.

In MIA testing was done by top – down integration where the each unit created is integrated with its previously created unit and tested for its validity.

First GUI was created then user interface is integrated with login module. Client – Server communication was done which then integrated with graphical user interface. After well-established communication data transfer operation was implemented where dynamic updation was validated.



*FURTHER ENHANCEMENTS*

---

## **6. FURTHER ENHANCEMENTS**

---

There is scope for increasing the number of operations that can be performed on the database.

Incase of other than peer networks mutually exclusive operations can be performed, provided the client is permitted to do so by the server.

Ciphers are included with the raw data to enhance the secured data transmission between the nodes.



*CONCLUSION*

---

## 7. CONCLUSION

---

The objective of this project is the implementation of a transaction processor to access the database present in the Linux server from a Windows machine. Three phases of the implementation of the project are

1. Creating a server process in Linux.
2. Creating a client process in Windows.
3. Creating a interface to a database in MYSQL.

All the three phases have been completed. The connection was established between the Linux and Windows machines and running the server and client process on respective machines did the operation of accessing the database from the Linux machine.





## *REFERENCES*


---

## 8. REFERENCES

---

 W.Richard Stevens, "UNIX NETWORK PROGRAMMING",  
Prentice Hall of India Pvt. Ltd, Aug 1997.

 MySQL by Paul DuBois,  
Techmedia Pvt. Ltd, Jan 2000.

 Windows 98 Programming by Herbert Schildt.  
Tata McGraw Hill Edition, Aug 1999.





*APPENDIX*

---

## **9. APPENDIX**

---

The Appendix discusses briefly the following

1. Source code
2. Sample output

## 9.1 SOURCE CODE

---

### SERVER CODE:

```
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/time.h>
#include <time.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <errno.h>
#include <fcntl.h>
#include <netdb.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/uio.h>
#include <unistd.h>
#include <sys/wait.h>
#include <sys/un.h>
#include <time.h>
#include <mysql.h>

#define def_host_name NULL
#define def_user_name NULL
#define def_password NULL
#define SERV_PORT 1300
#define LISTENQ 1024
#define MAXLINE 100
#define SA struct sockaddr

MYSQL *conn;
MYSQL_RES *res_set;
MYSQL_ROW row,col;

int main(void)
{
    int listenfd,connfd,l,i,k,j,d=0,p,m,aa;
    unsigned long f;
    FILE * fp;
    FILE * fd;
    FILE * fe;
```

```

        FILE * fz;
        int n,pid;
        pid=fork();
        if(pid==0)
        {
            char buffer1[50],*ans="";

            struct sockaddr_in servaddr;
char
incom[MAXLINE],buff[MAXLINE],inc[10],incomes[100],table[30
]="samp_db",num[10],que[50];
            char numa[30];
            char nu[50]="select * from as";
            listenfd = socket(AF_INET, SOCK_STREAM, 0);

            bzero(&servaddr, sizeof(servaddr));
            servaddr.sin_family      = AF_INET;
            servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
            servaddr.sin_port        = htons(1300);

/*****MYSQL CONNECTION ESTABLISH*****/

            conn=mysql_init(NULL);

            if(conn==NULL)
            {
                printf("aserror");
            }
            if(mysql_real_connect(conn,def_host_name,def_user_name
,def_password,table,0,NULL,0)==NULL)
            {
                printf("error2");
            }

/*****PORT CONNECTION*****/

            bind(listenfd, (SA *) &servaddr, sizeof(servaddr));
            listen(listenfd, LISTENQ);

            for ( ; ; )
            {
                connfd = accept(listenfd, (SA *) NULL, NULL);
                n=read(connfd,incom,MAXLINE);
                incom[n]='\0';
                printf("%s",incom);

```

```

fp=fopen("a.txt", "w+");
while(incom[0]!='#')
{

    if(incom[0]=='#')
    {
        printf("exiting");
        exit(0);
    }
    fseek(fp, 0, SEEK_END);
    fputs(incom, fp);
}

```

/\*\*\*\*\*\*DESC table name only TABLE FUNCTION\*\*\*\*\*\*/

```

    if(incom[0]=='d')
    {
        printf("%s the command is", incom);
        if(mysql_query(conn, incom)!=0)
            printf("query error");
        else
            {
                res_set =mysql_store_result(conn);
                if(res_set==NULL)
                    printf("res set error");

                else
                    {
                        k=0;
                        m=0;
                        n=0;
                        fe = fopen("c.txt", "w+");

while((row=mysql_fetch_row(res_set))!=NULL)
{

for(i=0;i<mysql_num_fields(res_set);i++)
{
if(i>0)
fputc('\t', stdout);
if(i==0)
{

fprintf(fe, "%s", row[i]!=NULL ? row[i]:"NULL");
printf("%s", row[i]!=NULL ? row[i]:"NULL");
fprintf(fe, ", ");
}
}
}
}

```

```

        fseek(fe, 0, SEEK_END);

    }

    k++;
    fputc('\n', stdout);

    }
    fseek(fe, 0, SEEK_SET);
    fscanf(fe, "%s", inc);
    fclose(fe );
    write(connfd, inc, 100);

    printf("%d", k);
    printf("%s", col);
    fputc('\n', stdout);
    if(mysql_errno(conn) != 0)
        printf("fetch error");

    else
    {
        printf("%lu fields returned\n", (unsigned
            long)mysql_num_rows(res_set));
    }
    }
    mysql_free_result(res_set);

    }

}

/*****NO OF RECORDS*****/

else
    if(incom[0]=='D')
    {

        j=0;
        for(i=5; i<n; i++)
        {
            numa[j]=incom[i];
            j++;
        }
        i=14;
        for(k=0; k<j; k++)
        {
            nu[i]=numa[k];
            i++;
        }
    }
}

```

```

    }
    nu[i]='\0';
    printf("%s",nu);

    if(mysql_query(conn,nu)!=0)
printf("query error");
else
    {
        res_set = mysql_store_result(conn);
        if(res_set==NULL)
            printf("res set error");

        else
            {
                k=0;
                fe = fopen("c.txt","w+");

while((row=mysql_fetch_row(res_set))!=NULL)
    {

for(i=0;i<mysql_num_fields(res_set);i++)
    {
        if(i>0)
            fputc('\t',stdout);
    }
        k++;

    }
printf("NO of records is:%d",k);
    }
        fprintf(fe,"%d",k);
        fprintf(fe,";");
        mysql_free_result(res_set);
    }

printf("%s the command is",incom);
if(mysql_query(conn,incom)!=0)
printf("query error");
else
    {
        res_set = mysql_store_result(conn);
        if(res_set==NULL)
            printf("res set error");

        else
            {
                k=0;

```

```

        m=0;
        n=0;

while((row=mysql_fetch_row(res_set))!=NULL)
{

for(i=0;i<mysql_num_fields(res_set);i++)
    {
        fseek(fe,0,SEEK_END);
        if(i>0)
            fputc('\t',stdout);
if(i==0)
{
    fprintf(fe,"%s",row[i]!=NULL ?
        row[i]:"NULL");

        printf("%s",row[i]!=NULL ?
row[i]:"NULL");
        fprintf(fe,",");
    }
    //fseek(fe,0,SEEK_END);

    }

    k++;
    fputc('\n',stdout);

    }
    fseek(fe,0,SEEK_SET);
    fscanf(fe,"%s",inc);
    fclose(fe );
    write(connfd,inc,100);

    printf("%d",k);
    printf("%s",col);
    fputc('\n',stdout);
    if(mysql_errno(conn)!=0)
        printf("fetch error");

    else
    {
        printf("%lu fields
returned\n",(unsigned long)mysql_num_rows(res_set));
    }
}

mysql_free_result(res_set);

```



```

        }
    }

    /*****SELECT * FROM FUNCTION*****/

    else
    if(incom[0]=='s')
    {

if(mysql_query(conn,incom)!=0)
        printf("query error");
    else
    {
        res_set =
mysql_store_result(conn);

        if(res_set==NULL)
            printf("res set error");

        else
        {
            k=0;
            m=0;
            n=0;
            fe = fopen("c.txt","w+");

while((row=mysql_fetch_row(res_set))!=NULL)
            {

for(i=0;i<mysql_num_fields(res_set);i++)
                {
                    if(i>0)
                        fputc('\t',stdout);

                    {

fprintf(fe,"%s",row[i]!=NULL ? row[i]:"NULL");
                    printf("%s",row[i]!=NULL ?
row[i]:"NULL");

                    fprintf(fe,",");
                }
                fseek(fe,0,SEEK_END);
            }

            k++;
            fputc('\n',stdout);

```

```

    }
    fseek(fe, 0, SEEK_SET);
    fscanf(fe, "%s", inc);
    fclose(fe );
    write(connfd, inc, 100);

    printf("%d", k);
    printf("%s", col);
    fputc('\n', stdout);
    if(mysql_errno(conn) != 0)
        printf("fetch error");

    else
    {
        printf("%lu fields
returned\n", (unsigned long)mysql_num_rows(res_set));
    }
}

```

```
mysql_free_result(res_set);
```

```
}
```

```
}
```

```
/******USE DATABASE******/
```

```
else
```

```
if(incom[0]=='U')
```

```

{
    printf("fucked");
    mysql_query(conn, incom);
    for(i=0; i<n; i++)
    {
        table[i]='\0';
        incomes[i]='\0';
    }
    j=0;
    for(i=4; i<=n; i++)
    {
        incomes[j]=incom[i];
        j++;
    }
    j=0;
    i=0;
    while(incomes[j]!='\0')
    {

```

```
        table[i]=incomes[j];
        i++;
        j++;
    }
    printf("%s",table);
```

```
if(mysql_real_connect(conn,def_host_name,def_user_name,def
_password,table,0,NULL,0)==NULL)
```

```
    {
        printf("error2");
    }
}
```

```
/******OTHERS******/
```

```
else
```

```
    if(mysql_query(conn,incom)!=0)
```

```
        connfd = accept(listenfd, (SA *) NULL,
NULL);
```

```
        n=read(connfd,incom,MAXLINE);
        fz=fopen("command.txt","w+");
        fprintf(fz,"%s",incom);
        fclose(fz);
        incom[n]='\0';
        printf("%s",incom);
```

```
    }
    fclose(fp);
```

```
    }
    close(connfd);
    mysql_close(conn);
}
```

```
}
```

## CLIENT CODE:

```
#include<windows.h>
#include<string.h>
#include<commctrl.h>
#include<stdio.h>
#include"ids.h"
HWND hwndclient =NULL;
HINSTANCE hinst;
SOCKET sock1;
FILE *fp;
HWND
hwnd,e1,e2,e3,e4,e5,s1,s2,s3,s4,s5,f[9],d[9],fl[9],dt[9];
int status,zz=0,i=0,j=0,k=0,l=0,m=0,z=0;
char num[2];

char str11[9][15]={"", "", "", "", "", "", "", "", ""};
char *str1=new char[];
char *recv1=new char[];
char *mod=new char[];
char *mod1=new char[];
char *dbname=new char[];
char *token;
char sep[]=", ";
char sep1[]=" ";
char sep2[] = "; ";

LRESULT CALLBACK WindProc(HWND, UINT, UINT, LONG);
LRESULT CALLBACK CreatedBProc (HWND,UINT,WPARAM,LPARAM);
LRESULT CALLBACK ShowDBProc (HWND,UINT,WPARAM,LPARAM);
LRESULT CALLBACK UseDBProc (HWND,UINT,WPARAM,LPARAM);
LRESULT CALLBACK DropDBProc (HWND,UINT,WPARAM,LPARAM);
LRESULT CALLBACK ShowTBProc (HWND,UINT,WPARAM,LPARAM);
LRESULT CALLBACK AlterTBProc (HWND,UINT,WPARAM,LPARAM);
char WinName[]="MyWin";
int WINAPI WinMain(HINSTANCE hthis,HINSTANCE hPrev,LPSTR lps,int r
{

    HWND hwnd;
    MSG msg;
    WNDCLASS wc;
    HACCEL hac;
    wc.style = 0;
    wc.lpfnWndProc = (WNDPROC)WindProc;
    wc.cbClsExtra = 0;
    wc.cbWndExtra = 0;
    wc.hIcon = LoadIcon (NULL, IDI_INFORMATION);
```

```

    wc.hCursor = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground =
(HBRUSH)GetStockObject(WHITE_BRUSH);
    wc.lpszMenuName = "MyMenu";
    wc.lpszClassName = WinName;
    wc.hInstance = hthis;

    RegisterClass(&wc);

    wc.lpfnWndProc=(WNDPROC)ChildWndProc;
    wc.hIcon =LoadIcon(NULL,IDI_APPLICATION);
    wc.hCursor = LoadCursor(NULL,IDC_ARROW);
    wc.hbrBackground
=(HBRUSH)GetStockObject(LTGRAY_BRUSH);
    wc.lpszMenuName =NULL;
    wc.lpszClassName ="Child";

    RegisterClass(&wc);

    wc.lpfnWndProc=(WNDPROC)CreateDBProc;
    wc.hIcon =LoadIcon(NULL,IDI_APPLICATION);
    wc.hCursor = LoadCursor(NULL,IDC_ARROW);
    wc.hbrBackground
=(HBRUSH)CreateSolidBrush(RGB(180,180,180));
    wc.lpszMenuName =NULL;
    wc.lpszClassName ="Create Database";

    hwnd =
CreateWindow(WinName,"OurWin",WS_OVERLAPPEDWINDOW|WS_VISIB
LE,0,0,1000,1000,NULL,NULL,hthis,NULL);
    hac=LoadAccelerators(hthis,"MyMenu");
    ShowWindow(hwnd,n);
    UpdateWindow(hwnd);

    while (GetMessage(&msg,NULL,0,0))

    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
    return (msg.wParam);
}

/***** SOCKET CREATION *****/

BOOL SocketCreation(HWND hwnd)
{

```

```

WSADATA WSADATA;
SOCKADDR_IN psin;
WSAStartup(MAKEWORD(1,1), &WSADATA);
sock1=socket(AF_INET,SOCK_STREAM,0);
psin.sin_addr.s_addr=inet_addr("192.168.12.5");
psin.sin_family =AF_INET;
psin.sin_port =htons(1300);
if (connect( sock1, (PSOCKADDR) &psin, sizeof( psin)) <
0)
{
    closesocket( sock1 );
    MessageBox(hWnd, "connect() failed", "Error",
MB_OK);
    return FALSE;
}
else
{
    MessageBox(hWnd, "CONNECTED TO SERVER ", "Error",
MB_OK);
    return TRUE;
}
}

```

/\*\*\*\*\*\* MAIN WINDOW PROCEDURE \*\*\*\*\*/

```

LRESULT CALLBACK WindProc(HWND hWnd,UINT mess,UINT wp, LONG
lp)
{
    HWND hchild;
    CLIENTCREATESTRUCT ccs;
    switch(mess)
    {
    case WM_COMMAND:
        switch(LOWORD(wp))
        {
            case ID_CREATEDB:
                hchild=CreateMDIWindow("Create
Database","Create
DB",NULL,200,140,300,250,hwndclient,hinst,0L);
                break;
            case ID_SHOWDB:
                hchild=CreateMDIWindow("Show
Database","Show
DB",NULL,200,140,300,250,hwndclient,hinst,0L);
                break;
            case ID_USEDDB:

```

```

        hchild=CreateMDIWindow("Use
Database", "Use
DB", NULL, 200, 140, 300, 250, hwndclient, hinst, 0L);
        break;
        case ID_SHOWTB:
            hchild=CreateMDIWindow("Show
Table", "Show
TB", NULL, 200, 140, 300, 350, hwndclient, hinst, 0L);
            break;
        case ID_EXIT:
            PostQuitMessage(0);
            break;
        case ID_START:

            hchild=CreateMDIWindow("Child", "hai", NULL, 200, 140, 300,
250, hwndclient, hinst, 0L);
            ShowWindow(hchild, SW_SHOW);
            UpdateWindow(hchild);
            break;
        case ID_DROPDB:
            hchild=CreateMDIWindow("Drop
Database", "Drop
DB", NULL, 200, 140, 300, 250, hwndclient, hinst, 0L);
            break;
    }
    break;
    case WM_CREATE:
        hwndclient
=CreateWindowEx(WS_EX_CLIENTEDGE, "MDICLIENT", NULL, WS_CHILD
|WS_CLIPCHILDREN, 200, 200, 300, 190, hwnd, NULL, hinst, &ccs);
        ShowWindow(hwndclient, SW_SHOW);
        UpdateWindow(hwndclient);
        break;
    case WM_DESTROY:
        PostQuitMessage(0);
        break;
    default:
        return
(DefFrameProc(hwnd, hwndclient, mess, wp, lp));
    }
    return(0L);
}

```

```

/***** DATABASE CREATION *****/

```

```

LRESULT CALLBACK CreateDBProc(HWND hWnd,UINT ums,WPARAM
wpm,LPARAM lpm)
{
    HWND b1;

    switch(ums)
    {
    case WM_CREATE:

        InitCommonControls();

        e1=CreateWindow("edit","",WS_CHILD|WS_VISIBLE|WS_BORDER,100,25,80,25,hWnd,(HMENU)2,hinst,NULL);

        b1=CreateWindow("button","DATABASE",WS_CHILD|WS_VISIBLE|WS_BORDER,100,80,80,30,hWnd,(HMENU)101,hinst,NULL);

        SetFocus(e1);

        break;
    case WM_COMMAND:
        switch(wpm)
        {
        case ID_DATABASE:
            strcpy(str,"");
            strcpy(buff1,"");

            if(!SocketCreation( hWnd))
            {
                MessageBox(hWnd,"ERROR111",0,0);
            }
            strcat(str,"create database ");
            strcat(str,buff1);
            send(sock1,str,50,0);
            break;
        }
        break;
    case WM_QUIT:
        PostQuitMessage(0);
        break;
    default:
        return (DefMDIChildProc(hWnd,ums,wpm,lpm));
    }
    return(0);
}

```



```
/****** SHOW TABLES *****/
```

```
LRESULT CALLBACK ShowTBProc (HWND hWnd, UINT ums, WPARAM  
wpm, LPARAM lpm)  
{  
    HWND b1;  
    switch(ums)  
    {  
    case WM_CREATE:  
  
        InitCommonControls();  
  
        if(!SocketCreation( hWnd))  
        {  
            MessageBox(hWnd, "ERROR111", 0, 0);  
        }  
        strcpy(str, "");  
        strcpy(szTemp, "");  
        for(i=0; i<9; i++)  
        {  
            strcpy(str31[i], "");  
        }  
        InitCommonControls();  
  
        e1=CreateWindow("listbox", "", WS_CHILD|WS_VISIBLE|WS_BO  
RDER|LBS_NOTIFY|WS_VSCROLL, 100, 25, 80, 50, hWnd, (HMENU)2, hins  
t, NULL);  
        b1=CreateWindow("button", "SHOW  
TABLES", WS_CHILD|WS_VISIBLE|WS_BORDER, 100, 100, 100, 25, hWnd,  
(HMENU)141, hinst, NULL);  
        strcpy(str, "Show tables ");  
        send(sock1, str, 50, 0);  
        status = recv(sock1, szTemp, 100, 0 );  
        szTemp[ status ] = '\0';  
        token=strtok(szTemp, sep);  
  
        k=0;  
        i=0;  
        while(token!=NULL)  
        {  
            strcpy(str31[i], token);  
            k++;  
            token=strtok(NULL, sep);  
            i++;  
        }  
    }  
}
```

```

    }
    for(i=0;i<k;i++)
    {

SendMessage(e1, LB_ADDSTRING, 0, (DWORD)str31[i]);
    }

    break;
case WM_COMMAND:
    switch(wpm)
    {
    case ID_SHOWTAB:
        strcpy(str, "");
        strcpy(buff1, "");
        strcpy(szTemp, "");
        strcpy(recv1, "");

        LV_ITEM item;
        LV_COLUMN col;

        if(!SocketCreation( hWnd))
        {
            MessageBox(hWnd, "ERROR111", 0, 0);
        }

        e2=CreateWindow(WC_LISTVIEW, "", WS_CHILD|WS_VISIBLE|WS_
        BORDER|LVS_EDITLABELS|LVS_LIST, 100, 130, 100, 150, hWnd, (HMENU
        )2, hinst, NULL);

        strcat(str, "Show tables ");
        i=0;
        i=SendMessage(e1, LB_GETCURSEL, 0, 0);

        SendMessage(e1, LB_GETTEXT, i, (DWORD) (LPSTR) dbname);
        send(sock1, str, 100, 0);
        status = recv(sock1, szTemp, 50, 0 );
        szTemp[ status ] = '\0';
        token=strtok(szTemp, sep);

        k=0;
        i=0;
        while(token!=NULL)
        {
            strcpy(str21[i], token);
            k++;
            token=strtok(NULL, sep);

```

```

        i++;
    }

col.mask=LVCF_FMT|LVCF_WIDTH|LVCF_TEXT|LVCF_SUBITEM;
col.fmt=LVCFMT_LEFT;
col.cx=100;

col.pszText="Tables";
col.iSubItem=0;
ListView_InsertColumn(e2,0,&col);

for(i=0;i<k;i++)
{
    item.mask=LVIF_TEXT;
    item.pszText=(LPTSTR)str21[i];
    item.iItem=i;
    item.iSubItem=0;
    ListView_InsertItem( e2,&item);
}
break;
}
break;
case WM_QUIT:
    PostQuitMessage(0);
    break;
default:
    return (DefMDIChildProc(hWnd,ums,wpm,lpm));
}
return(0);
}

```

/\*\*\*\*\*\* INSERTION \*\*\*\*\*/

```

LRESULT CALLBACK Insertion(HWND hWnd,UINT ums1,WPARAM
wpm1,LPARAM lpm1)

```

```

{
    HWND b1,b2;
    int fid=70,did=80,y=230;

    switch(ums1)
    {
    case WM_CREATE:
        strcpy(str,"");
        strcpy(buff1,"");

        b1=CreateWindow("button","OK",WS_CHILD|WS_VISIBLE|WS_B

```

```
ORDER|BS_PUSHBUTTON,200,80,80,30,hWnd,(HMENU)301,hinst,NU  
L);
```

```
    s5=CreateWindow("static","Table  
Name:",WS_CHILD|WS_VISIBLE|WS_BORDER|SS_CENTER,135,40,120,  
20,hWnd,(HMENU)48,hinst,NULL);
```

```
    e5=CreateWindow("listbox","",LBS_NOTIFY|WS_VSCROLL|WS_  
CHILD|WS_VISIBLE|WS_BORDER,275,40,120,20,hWnd,(HMENU)49,hi  
nst,NULL);
```

```
        if(!SocketCreation( hWnd))  
        {  
            MessageBox(hWnd,"ERROR111",0,0);  
        }
```

```
        strcat(str,"Show tables ");  
        send(sock1,str,100,0);  
        status = recv(sock1,szTemp,50,0 );  
        szTemp[ status ] = '\0';  
        token=strtok(szTemp,sep);  
        k=0;  
        i=0;  
        while(token!=NULL)
```

```
        {  
            strcpy(str21[i],token);  
            k++;  
            token=strtok(NULL,sep);  
            i++;  
        }
```

```
        for(i=0;i<k;i++)  
        {
```

```
            SendMessage(e5,LB_ADDSTRING,0,(DWORD)str21[i]);
```

```
        }  
        break;
```

```
case WM_COMMAND:
```

```
    switch(LOWORD(wpm1))
```

```
    {  
        case ID_INOK:  
            strcpy(str,"");  
            strcpy(buff1,"");  
            strcpy(szTemp,"");  
            strcpy(recv1,"");
```

```
            if(!SocketCreation( hWnd))  
            {  
                MessageBox(hWnd,"ERROR111",0,0);  
            }
```

```
            i=0;
```

```

        i=SendMessage(e5, LB_GETCURSEL, 0, 0);

SendMessage(e5, LB_GETTEXT, i, (DWORD) (LPSTR)buff1);
    strcat(str, "desc ");
    strcat(str, buff1);
    send(sock1, str, 100, 0);
    status = recv(sock1, szTemp, 50, 0 );
    szTemp[ status ] = '\0';
    token=strtok(szTemp, sep);

    k=0;
    i=0;
    while(token!=NULL)
    {
        strcpy(str31[i], token);
        k++;
        token=strtok(NULL, sep);
        i++;
    }

    for(i=0; i<k; i++)
    {

        f[i]=CreateWindow("static", NULL, WS_CHILD|WS_VISIBLE|WS
_BORDER, 100, y, 100, 20, hWnd, (HMENU)fid+i, hinst, NULL);

        d[i]=CreateWindow("edit", NULL, WS_CHILD|WS_VISIBLE|WS_B
ORDER, 250, y, 100, 20, hWnd, (HMENU)did+i, hinst, NULL);
            y=y+30;
        }

        b2=CreateWindow("button", "INSERT", WS_CHILD|WS_VISIBLE|
WS_BORDER|BS_PUSHBUTTON, 215, y+2, 80, 30, hWnd, (HMENU)302, hins
t, NULL);

        for(i=0; i<k; i++)
        {
            SetWindowText(f[i], str31[i]);
        }
        for(i=0; i<9; i++)
        {
            strcpy(str11[i], "");
            strcpy(str31[i], "");
        }
        break;

case ID_INSERT:

```

```

        strcpy(str, "");
        strcpy(buff1, "");
        strcpy(szTemp, "");
        strcpy(buff2, "");

        if(!SocketCreation( hWnd))
        {
            MessageBox(hWnd, "ERROR111", 0, 0);
        }
        i=0;
        i=SendMessage(e5, LB_GETCURSEL, 0, 0);

SendMessage(e5, LB_GETTEXT, i, (DWORD) (LPSTR)buff1);
        strcat(str, "Insert into ");
        strcat(str, buff1);
        strcat(str, " values(");
        for(i=0; i<k; i++)
        {
            strcat(buff2, "'");
            GetWindowText(d[i], buff3, 15);
            strcat(buff2, buff3);
            strcat(buff2, "'");
            if(i!=k-1)
                strcat(buff2, ",");
        }
        strcat(str, buff2);
        strcat(str, ")");
        send(sock1, str, 200, 0);
        break;
    }
    break;
case WM_QUIT:
    PostQuitMessage(0);
    break;
default:
    return (DefMDIChildProc(hWnd, ums1, wpm1, lpm1));
}
return(0L);
}

```

/\*\*\*\*\*\* RETRIEVAL \*\*\*\*\*/

```

LRESULT CALLBACK Retrieval(HWND hw2, UINT ums2, WPARAM
wpm2, LPARAM lpm2)
{
    HWND b11, b2;
    int y=230, fid=70, did=80;

```

```

switch(ums2)
{
case WM_CREATE:
    strcpy(str, "");

    b11=CreateWindow("button", "OK", WS_CHILD|WS_VISIBLE|WS_
BORDER|BS_PUSHBUTTON, 200, 80, 80, 30, hw2, (HMENU) 351, hinst, NUL
L);

    s5=CreateWindow("static", "Table
Name:", WS_CHILD|WS_VISIBLE|WS_BORDER|SS_CENTER, 135, 40, 120,
20, hw2, (HMENU) 70, hinst, NULL);

    e5=CreateWindow("listbox", "", LBS_NOTIFY|WS_VSCROLL|WS_
CHILD|WS_VISIBLE|WS_BORDER, 275, 40, 120, 20, hw2, (HMENU) 49, hin
st, NULL);
    if(!SocketCreation( hw2))
    {
        MessageBox(hw2, "ERROR111", 0, 0);
    }
    strcat(str, "show tables ");
    send(sock1, str, 100, 0);
    status = recv(sock1, szTemp, 50, 0 );
    szTemp[ status ] = '\0';
    token=strtok(szTemp, sep);
    k=0;
    i=0;
    while(token!=NULL)
    {
        strcpy(str21[i], token);
        k++;
        token=strtok(NULL, sep);
        i++;
    }
    for(i=0; i<k; i++)
    {
        SendMessage(e5, LB_ADDSTRING, 0, (DWORD)str21[i]);
    }
    break;
case WM_COMMAND:
    switch(LOWORD(wpm2))
    {
case ID_RETOK:
        strcpy(str, "");
        strcpy(buff1, "");
        strcpy(szTemp, "");
        strcpy(recv1, "");

```

```

strcpy(buff2, "");
strcpy(buff3, "");

for(i=0;i<9;i++)
{
    strcpy(str31[i], "");
}

if(!SocketCreation( hw2))
{
    MessageBox(hw2, "ERROR111", 0, 0);
}
i=0;
i=SendMessage(e5, LB_GETCURSEL, 0, 0);

SendMessage(e5, LB_GETTEXT, i, (DWORD) (LPSTR)buff1);
strcat(str, "desc ");
strcat(str, buff1);
send(sock1, str, 100, 0);
szTemp[0]=NULL;
status = recv(sock1, szTemp, 50, 0 );
szTemp[ status ] = '\0';
token=strtok(szTemp, sep);

k=0;
i=0;
while(token!=NULL)
{
    strcpy(str31[i], token);
    strcpy(str11[i], token);
    k++;
    token=strtok(NULL, sep);
    i++;
}

for(i=0;i<k;i++)
{

    f[i]=CreateWindow("static", NULL, WS_CHILD|WS_VISIBLE|WS
_BORDER, 100, y, 100, 20, hw2, (HMENU)fid+i, hinst, NULL);

    d[i]=CreateWindow("edit", NULL, WS_CHILD|WS_VISIBLE|WS_B
ORDER, 250, y, 100, 20, hw2, (HMENU)did+i, hinst, NULL);
    y=y+30;
}

b2=CreateWindow("button", "NEXT", WS_CHILD|WS_VISIBLE|WS

```



```
_BORDER|BS_PUSHBUTTON, 215, y+2, 80, 30, hw2, (HMENU) 352, hinst, N  
ULL);
```

```
    for(i=0; i<k; i++)  
    {  
        SetWindowText(f[i], str11[i]);  
    }  
    break;  
  
case ID_RETRIEVE:  
    for(i=0; i<9; i++)  
    {  
        strcpy(str11[i], "");  
    }  
    if(!SocketCreation( hw2))  
    {  
        MessageBox(hw2, "ERROR111", 0, 0);  
    }  
    strcpy(str, "");  
    strcpy(szTemp, "");  
    zz++;  
    fp=fopen("project.txt", "w+");  
    fprintf(fp, "%d", zz);  
    fseek(fp, 0, SEEK_SET);  
    fscanf(fp, "%s", buff2);  
    fclose(fp);  
    strcpy(str, "v");  
    strcat(str, buff2);  
    strcat(str, " select * from ");  
    strcat(str, buff1);  
    send(sock1, str, 100, 0);  
    status=recv(sock1, szTemp, 100, 0);  
    szTemp[status]='\0';  
    token=strtok(szTemp, sep);  
  
    k=0;  
    i=0;  
    while(token!=NULL)  
    {  
        strcpy(str11[i], token);  
        k++;  
        token=strtok(NULL, sep);  
        i++;  
    }  
    for(i=0; i<k; i++)  
    {  
        SetWindowText(d[i], str11[i]);
```

```

        }
        break;
    }
    break;
case WM_QUIT:
    PostQuitMessage(0);
    break;
default:
    return (DefMDIChildProc(hw2,ums2,wpm2,lpm2));
}
return(0);
}

```

/\*\*\*\*\*\* CREATION \*\*\*\*\*/

```

LRESULT CALLBACK Creation(HWND hw2,UINT ums2,WPARAM
wpm2,LPARAM lpm2)
{
    HWND b1,b2;
    int fid=50,did=60,y=230;
    switch(ums2)
    {
    case WM_CREATE:
        s1=CreateWindow("static","Table
Name:",WS_CHILD|WS_VISIBLE|WS_BORDER|SS_CENTER,135,60,120,
20,hw2,(HMENU)37,hinst,NULL);

        e1=CreateWindow("edit",NULL,WS_CHILD|WS_VISIBLE|WS_BOR
DER|WS_GROUP|WS_TABSTOP,275,60,120,20,hw2,(HMENU)38,hinst,
NULL);

        s2=CreateWindow("static","No:of
Fields:",WS_CHILD|WS_VISIBLE|WS_BORDER|WS_TABSTOP|SS_CENTE
R,135,100,120,20,hw2,(HMENU)39,hinst,NULL);

        e2=CreateWindow("edit",NULL,WS_CHILD|WS_VISIBLE|WS_BOR
DER|WS_TABSTOP,275,100,120,20,hw2,(HMENU)40,hinst,NULL);

        b1=CreateWindow("button","OK",WS_CHILD|WS_VISIBLE|WS_B
ORDER|BS_PUSHBUTTON|WS_TABSTOP,215,150,80,30,hw2,(HMENU)45
1,hinst,NULL);
        break;
    case WM_COMMAND:
        switch(LOWORD(wpm2))
        {
        case ID_CREATEOK:
            strcpy(buff1,"");

```

```

        GetWindowText(e2, buff1, 5);
        m=atoi(buff1);
        s3=CreateWindow("static", "Field
Name", WS_CHILD|WS_VISIBLE|WS_BORDER|SS_CENTER, 100, 200, 120,
20, hw2, (HMENU)42, hinst, NULL);
        s4=CreateWindow("static", "Data
Type", WS_CHILD|WS_VISIBLE|WS_BORDER|SS_CENTER, 250, 200, 120,
20, hw2, (HMENU)43, hinst, NULL);
        for(i=0; i<m; i++)
        {

            f[i]=CreateWindow("edit", NULL, WS_CHILD|WS_VISIBLE|WS_B
ORDER, 100, y, 100, 20, hw2, (HMENU)fid, hinst, NULL);

            d[i]=CreateWindow("combobox", NULL, CBS_AUTOHSCROLL|CBS_
DISABLENOSCROLL|CBS_DROPDOWN|CBS_DROPDOWNLIST|CBS_SORT|WS_
CHILD|WS_VISIBLE|WS_BORDER|WS_VSCROLL|WS_TABSTOP, 250, y, 100
, 20, hw2, (HMENU)did, hinst, NULL);

            SendDlgItemMessage(hw2, did, CB_ADDSTRING, 0, (LPARAM)"var
char(15)");

            SendDlgItemMessage(hw2, did, CB_ADDSTRING, 0, (LPARAM)"int
");

            SendDlgItemMessage(hw2, did, CB_ADDSTRING, 0, (LPARAM)"dat
e");

                did=did+1;
                fid=fid+1;
                y=y+30;
        }

        b2=CreateWindow("button", "CREATE", WS_CHILD|WS_VISIBLE|
WS_BORDER|BS_PUSHBUTTON, 215, y+2, 80, 30, hw2, (HMENU)452, hinst
, NULL);

        break;
    case ID_CREATE:
        strcpy(str, "");
        strcpy(buff1, "");
        strcpy(buff2, "");
        strcpy(buff3, "");
        strcpy(buff4, "");

        if(!SocketCreation( hw2))
        {
            MessageBox(hw2, "ERROR111", 0, 0);

```

```

    }

    strcat(str,"create table ");
    GetWindowText(e1,buff1,10);
    strcat(buff1,"(");
    strcat(str,buff1);
    for(i=0;i<m;i++)
    {
        GetWindowText(f[i],buff2,15);
        strcat(buff4,buff2);
        strcat(buff4," ");
        GetWindowText(d[i],buff3,15);
        strcat(buff4,buff3);
        if(i!=m-1)
            strcat(buff4,",");
    }
    strcat(str,buff4);
    strcat(str,")");
    send(sock1,str,200,0);
    break;

}
break;

case WM_QUIT:
    PostQuitMessage(0);
    break;
default:
    return (DefMDIChildProc(hw2,ums2,wpm2,lpm2));
}

return(0L);
}

```

/\*\*\*\*\*\* DROP TABLE \*\*\*\*\*/

LRESULT CALLBACK Dropping(HWND hw2,UINT ums2,WPARAM wpm2,LPARAM lpm2)

```

{
    HWND b1;

    switch(ums2)
    {
    case WM_CREATE:
        strcpy(str,"");
        strcpy(szTemp,"");
    }
}

```

```
    s1=CreateWindow("static","Table
Name:",WS_CHILD|WS_VISIBLE|WS_BORDER|SS_CENTER,135,60,120,
20,hw2,(HMENU)45,hinst,NULL);
```

```
    b1=CreateWindow("button","DROP",WS_CHILD|WS_VISIBLE|WS
_BORDER|BS_PUSHBUTTON,215,150,80,30,hw2,(HMENU)501,hinst,N
ULL);
```

```
    e5=CreateWindow("listbox","",LBS_NOTIFY|WS_VSCROLL|WS_
CHILD|WS_VISIBLE|WS_BORDER,275,40,120,20,hw2,(HMENU)49,hin
st,NULL);
```

```
    if(!SocketCreation( hw2))
    {
        MessageBox(hw2,"ERROR111",0,0);
    }
```

```
    strcat(str,"show tables ");
    send(sock1,str,100,0);
    status = recv(sock1,szTemp,50,0 );
    szTemp[ status ] = '\0';
    token=strtok(szTemp,sep);
    k=0;
    i=0;
    while(token!=NULL)
    {
        strcpy(str21[i],token);
        k++;
        token=strtok(NULL,sep);
        i++;
    }
    for(i=0;i<k;i++)
    {
```

```
        SendMessage(e5, LB_ADDSTRING, 0, (DWORD)str21[i]);
    }
    break;
case WM_COMMAND:
```

```
    switch(LOWORD(wpm2))
    {
    case ID_DROPTAB:
        strcpy(str,"");
        strcpy(buff1,"");
```

```
        if(!SocketCreation( hw2))
        {
            MessageBox(hw2,"ERROR111",0,0);
        }
```

```

        i=0;
        strcat(str,"Drop table ");
        i=SendMessage(e5, LB_GETCURSEL, 0, 0);

SendMessage(e5, LB_GETTEXT, i, (DWORD) (LPSTR)buff1);

        strcat(str,buff1);
        send(sock1, str, 200, 0);
        break;
    }
    break;
case WM_QUIT:
    PostQuitMessage(0);
    break;
default:
    return (DefMDIChildProc(hw2, ums2, wpm2, lpm2));
}
return(0L);
}

/***** ALTERATION *****/

LRESULT CALLBACK Alteration(HWND hw2, UINT ums2, WPARAM
wpm2, LPARAM lpm2)
{
    HWND b1,b2,b3,halter;

    switch(ums2)
    {
    case WM_CREATE:
        strcpy(str, "");

        b1=CreateWindow("button", "RENAME", WS_CHILD|WS_VISIBLE|
WS_BORDER|BS_PUSHBUTTON, 150, 80, 100, 20, hw2, (HMENU) 551, hinst
, NULL);

        b2=CreateWindow("button", "ALTER", WS_CHILD|WS_VISIBLE|W
S_BORDER|BS_PUSHBUTTON, 250, 80, 100, 20, hw2, (HMENU) 552, hinst,
NULL);

        e5=CreateWindow("listbox", "", LBS_NOTIFY|WS_VSCROLL|WS_
CHILD|WS_VISIBLE|WS_BORDER, 200, 40, 120, 20, hw2, (HMENU) 49, hin
st, NULL);
        if(!SocketCreation( hw2))
        {
            MessageBox(hw2, "ERROR111", 0, 0);
        }
    }
}

```

```

    strcat(str,"show tables ");
    send(sock1,str,100,0);
    status = recv(sock1,szTemp,50,0 );
    szTemp[ status ] = '\0';
    token=strtok(szTemp,sep);
    k=0;
    i=0;
    while(token!=NULL)
    {
        strcpy(str21[i],token);
        k++;
        token=strtok(NULL,sep);
        i++;
    }
    for(i=0;i<k;i++)
    {
        SendMessage(e5,LB_ADDSTRING,0,(DWORD)str21[i]);
    }
    break;
case WM_COMMAND:
    switch(LOWORD(wpm2))
    {
        case ID_RENAMEOK:

            e1=CreateWindow("edit",NULL,WS_CHILD|WS_VISIBLE|WS_BORDER,
120,120,100,20,hw2,(HMENU)49,hinst,NULL);

            e2=CreateWindow("edit",NULL,WS_CHILD|WS_VISIBLE|WS_BORDER,
200,120,100,26,hw2,(HMENU)50,hinst,NULL);

            b3=CreateWindow("button","OK",WS_CHILD|WS_VISIBLE|WS_BORDER|
BS_PUSHBUTTON,200,160,100,20,hw2,(HMENU)553,hinst,NU
LL);

                break;

        case ID_ALTEROK:
            strcpy(buff1,"");
            i=0;
            i=SendMessage(e5,LB_GETCURSEL,0,0);

            SendMessage(e5,LB_GETTEXT,i,(DWORD)(LPSTR)buff1);

            halter=CreateMDIWindow("AlterTab","AlterTable",NULL,13
5,0,500,520,hwndclient,hinst,NULL);
                SetFocus(e1);

```

```

        ShowWindow(halter, SW_SHOW);
        UpdateWindow(halter);
        break;

    case ID_RENAME:
        strcpy(str, "");
        strcpy(buff2, "");
        strcpy(buff3, "");

        if(!SocketCreation( hw2))
        {
            MessageBox(hw2, "ERROR111", 0, 0);
        }
        strcat(str, "Alter table ");
        i=0;
        i=SendMessage(e5, LB_GETCURSEL, 0, 0);

    SendMessage(e5, LB_GETTEXT, i, (DWORD) (LPSTR)buff2);

        strcat(str, buff2);
        GetWindowText(e2, buff3, 10);
        strcat(str, " rename as ");
        strcat(str, buff3);
        send(sock1, str, 100, 0);
        break;
    }
    break;
case WM_QUIT:
    PostQuitMessage(0);
    break;
default:
    return (DefMDIChildProc(hw2, ums2, wpm2, lpm2));
}
return(0L);
}

/*****ALTER TABLE *****/

LRESULT CALLBACK AlterTBProc(HWND hWnd, UINT ums, WPARAM
wpm, LPARAM lpm)
{
    HWND b1, b2, b3, f[9], d[9];
    char spr[]=" ,";
    int fid=50, did=60, y=80;
    int fd=70, dd=80;
    switch(ums)
    {

```



```

case WM_CREATE:

    InitCommonControls();

    e1=CreateWindow("edit","",WS_CHILD|WS_VISIBLE|WS_BORDER|WS_GROUP,160,25,80,20,hWnd,(HMENU)2,hinst,NULL);

    b1=CreateWindow("button","DESC",WS_CHILD|WS_VISIBLE|WS_BORDER|WS_TABSTOP,260,25,80,20,hWnd,(HMENU)601,hinst,NULL);

    SetWindowText(e1,buff1);
    SetFocus(e1);
    break;
case WM_COMMAND:
    switch(LOWORD(wpm))
    {
    case ID_DESC:

        strcpy(str,"");
        strcpy(buff1,"");
        strcpy(szTemp,"");
        strcpy(buff2,"");
        if(!SocketCreation( hWnd))
        {
            MessageBox(hWnd,"ERROR111",0,0);
        }

        strcat(str,"Besc ");
        GetWindowText(e1,buff1,15);
        strcat(str,buff1);
        send(sock1,str,100,0);
        status = recv(sock1,szTemp,50,0 );
        szTemp[ status ] = '\0';
        token=strtok(szTemp,spr);

        k=0;
        i=0;
        while(token!=NULL)
        {
            strcpy(str31[i],token);
            k++;
            token=strtok(NULL,spr);
            i++;
        }

        i=0;
        for(j=0;j<k/2;j++)

```

```

    {

        f[j]=CreateWindow("static",NULL,WS_CHILD|WS_VISIBLE|WS_
_BORDER|WS_TABSTOP,120,y,100,20,hWnd,(HMENU)fid,hinst,NULL
);

        SetWindowText(f[j],str31[i]);

        d[j]=CreateWindow("edit",NULL,WS_CHILD|WS_VISIBLE|WS_B
ORDER|WS_TABSTOP,270,y,100,20,hWnd,(HMENU)did,hinst,NULL);

        SetWindowText(d[j],str31[i=i+1]);
        did=did+1;
        fid=fid+1;
        y=y+30;
        i++;
    }

    e2=CreateWindow("edit","",WS_CHILD|WS_VISIBLE|WS_BORDE
R|WS_TABSTOP,120,y,80,20,hWnd,(HMENU)3,hinst,NULL);

    b2=CreateWindow("button","ADD",WS_CHILD|WS_VISIBLE|WS_
_BORDER|WS_TABSTOP,270,y,80,20,hWnd,(HMENU)602,hinst,NULL);

    break;
case ID_ADDOK:
    GetWindowText(e2,buff2,5);
    m=atoi(buff2);
    y=y+90;
    for(i=0;i<m;i++)
    {

        fl[i]=CreateWindow("edit",NULL,WS_CHILD|WS_VISIBLE|WS_
_BORDER|WS_TABSTOP,120,y+30,100,20,hWnd,(HMENU)fd,hinst,NUL
L);

        dt[i]=CreateWindow("combobox",NULL,CBS_AUTOHSCROLL|CBS_
_DISABLENOSCROLL|CBS_DROPDOWN|CBS_DROPDOWNLIST|CBS_SORT|WS_
_CHILD|WS_VISIBLE|WS_BORDER|WS_VSCROLL|WS_TABSTOP,270,y+30
,100,20,hWnd,(HMENU)dd,hinst,NULL);

        SendDlgItemMessage(hWnd,dd,CB_ADDSTRING,0,(LPARAM)"var
char(15)");

        SendDlgItemMessage(hWnd,dd,CB_ADDSTRING,0,(LPARAM)"int
");
    }
}

```

```
SendDlgItemMessage(hWnd, dd, CB_ADDSTRING, 0, (LPARAM) "dat  
e");
```

```
    dd=dd+1;  
    fd=fd+1;  
    y=y+30;
```

```
    }
```

```
    b3=CreateWindow("button", "OK", WS_CHILD|WS_VISIBLE|WS_B  
ORDER|BS_PUSHBUTTON|WS_TABSTOP, 215, y+30, 80, 30, hWnd, (HMENU)  
603, hinst, NULL);
```

```
        break;
```

```
case ID_ADD:
```

```
    if(!SocketCreation( hWnd))
```

```
    {
```

```
        MessageBox(hWnd, "ERROR111", 0, 0);
```

```
    }
```

```
    strcpy(str, "");
```

```
    strcpy(buff1, "");
```

```
    strcpy(buff2, "");
```

```
    strcpy(mod, "");
```

```
    strcpy(mod1, "");
```

```
    strcat(str, "alter table ");
```

```
    GetWindowText(e1, buff1, 15);
```

```
    strcat(str, buff1);
```

```
    strcat(str, " add ");
```

```
    for(i=0; i<m; i++)
```

```
    {
```

```
        GetWindowText(f1[i], mod, 15);
```

```
        strcat(buff2, mod);
```

```
        strcat(buff2, " ");
```

```
        GetWindowText(dt[i], mod1, 15);
```

```
        strcat(buff2, mod1);
```

```
        if(i!=m-1)
```

```
            strcat(buff2, ",");
```

```
    }
```

```
    strcat(str, buff2);
```

```
    send(sock1, str, 100, 0);
```

```
    break;
```

```
    }
```

```
break;
```

```
case WM_QUIT:
```

```
    PostQuitMessage(0);
```

```
    break;
```

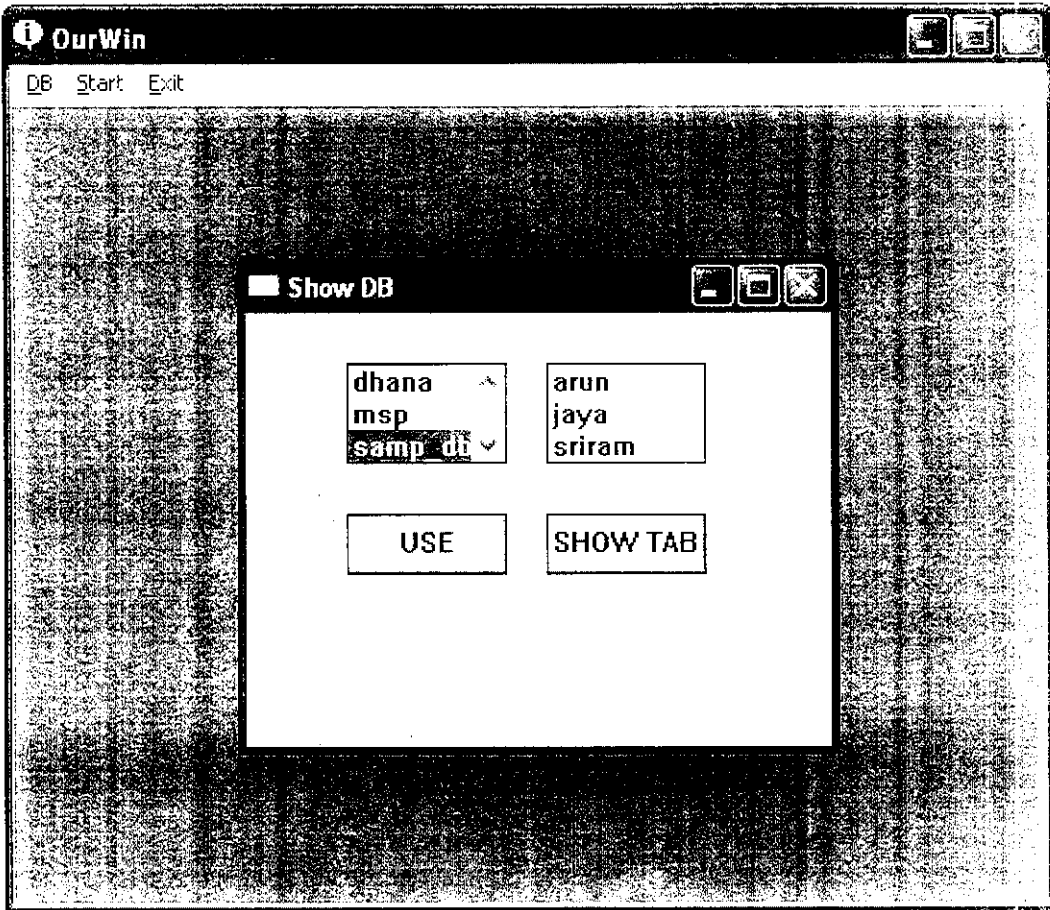
```
default:
```

```
    return (DefMDIChildProc(hWnd, ums, wpm, lpm));  
}  
return(0);  
}
```

## 9.2 SAMPLE OUTPUTS

---

### VIEW DATABASE:



# TABLE CREATION:

The image shows a 'Create' dialog box with the following fields and controls:

- Table Name:** SAMPLE
- No:of Fields:** 3
- OK** button
- Field Name** and **Data Type** columns:

Field Name	Data Type
NAME	varchar(15) ▾
AGE	int ▾
DEPT	varchar(15) ▾

**CREATE** button

DROP TABLE:

