A SOFTWARE PROJECT REPORT ON

A GUI FOR REMOTE CONTROL OF HEALTH MONITORING SYSTEM

project carried out at

RAMAN RESEARCH INSTITUTE
BANGALORE 560 080

Under the Guidance of

Mr. R. GANESAN
Scientist
RRI.

Dr. D.K. RAVINDRA
Head, Radio Astronomy Lab
RRI.

By

LATHA C.V
MANJUSHREE .K
NAVITHA .R
RUPASHRI C.R

Submitted to Bangalore University in partial fulfillment of the requirements for the award of the degree of

Bachelor of Engineering
in
Computer Science and Engineering.

Internal Guide :
Mr. A. M. PRASAD
Lecturer, Computer Science & Engineering,
D.S.C.E, Bangalore.

Department of Computer Science and Engineering
Dayananda Sagar College of Engineering
BANGALORE – 560 078
1999-2000

Dayananda Sagar College of Engineering
Bangalore – 560 078

CERTIFICATE

This is to certify that the following Students

Latha C.V	**Manjushree K**
Navitha R	**Rupashri C.R**

have successfully completed the Project Work Entitled

"A GUI FOR REMOTE CONTROL OF HEALTH MONITORING SYSTEM"

at

Raman Research Institute, RRI, Bangalore

in partial fulfillment of the requirements for the award of the degree of Bachelor of Engineering in Computer Science & Engineering by the Bangalore University during the academic year 1999 - 2000

Prasad. A.M.

Mr. A.M. Prasad
Lecturer,
Dept. of Computer Science,
D.S.C.E. Bangalore

[illegible]

Dr N.Bhaskara Rao
Head of the Department,
Dept. of Computer Science,
D.S.C.E. Bangalore

Examiners:

A. M. Prasad
LECTURER
DEPARTMENT OF COMPUTER SCIENCE
DAYANANDA SAGAR COLLEGE OF ENGINEERING
BANGALORE-560 078.

RAMAN RESEARCH INSTITUTE

C.V. Raman Avenue, Sadashivanagar, Bangalore-560 080 India

July 6, 2000

Certificate

This is to certify that the project work titled

"A GUI for Remote Control of Health Monitoring System"

has been successfully completed by

Rupashri C.R

Manjushree K

Navitha R

Latha C.V

Students of the final year,

Bachelor of Engineering (Computer Science)

Dayananda Sagar College of Engineering, Bangalore

under the guidance of *Mr. R. Ganesan.*

This project is in partial fulfillment of the requirement for the award of Bachelor's Degree in Computer Science Engineering during the academic year 1999-2000.

[illegible]

Dr. D.K. Ravindra
Head, Radio Astronomy Lab.

[illegible]

R. Ganesan
Scientist

Phone : (80) 334 0122 Fax : (80) 334 0492 Telegrams : RAMANINST e-mail : root@rri.ernet.in

ACKNOWLEDMENT

This project work would be incomplete if we do not take this opportunity to thank all those who were behind us in successful completion of this project.

We would like to express our sincere gratitude to our guide **Mr.A.M.Prasad** (Lecturer, Department of Computer Science, D.S.C.E, Bangalore) for being a source of inspiration and for helping us in all matters regarding the project.

We shall for ever be indebted to **Dr. N. Bhaskara Rao**(Head of the Department,Computer Science and Engineering) for his continuous encouragement given to us in the successful completion of the project.

We owe our thanks to Principal **Mr. Puttaraju** for allowing us to do this project.

We would like to whole heartedly thank **Prof. N. Kumar**, Director and **Dr. D.K. Ravindra**, Radio Astronomy Lab of Raman Research Institute for providing us an opportunity to undertake our project work in this reputed institute and utilizing the facilities at RRI.

We would like to thank **Mr. R. Ganesan** , for his concern and co-operation given to us to complete the project successfully. Special mention has to be made of **Mrs S. Swarna** for her skillful steering and time to time assistance given to us in shaping our project.

We would like to extend our thanks to **Mr. K. Gurukiran** and **Mr. K. Ramesh** of RRI for providing assistance and guidance while developing our project .

We wish to acknowledge the co-operation provided to us by all the staff members of the Telescope building during the course of our project.

Finally, we would like to thank all the staff members of Department of Computer Science and our friends for the kind co-operation rendered towards us .

ABSTRACT

The Telescope Health Monitoring System (HMS) monitors various parameters of the Telescope sub system. This is now done by a PC and three 80186 based single board computers (SBC). SBCs are Monitor Modules (MM) that monitor various voltages and currents from sub systems. Any critical parameter failure is reported to the PC.

The HMS-PC has to communicate any failure to a Master Coordination Computer (SUN Work-station).A GUI at the workstation has to be developed to configure the MM's through the PC. Therefore a network communication software, using TCP/IP has to be developed in the workstation and the PC side.

The GUI is to be developed using TCL/TK scripting language on SUN Solaris environment. The GUI should have the following features:

1. A click-and-enter user interface to feed configuration data and save to a disk file.
2. Send these configuration data to the PC via network.
3. Update the display periodically in the workstation by communicating to the PC.

The PC software has provisions to configure the MM through RS232 interface. Apart from this, the software should have:

- A network communication module to connect to workstation. FTP Software Inc. Software Development Kit on DOS platform will be used for this module.

The following modules are developed:

1. Design and Develop code in C for network communication.
2. Implement GUI using TCL/TK

SYSTEM REQUIREMENTS

SOFTWARE REQUIREMENTS

PLATFORM	SUN SOLARIS Operating System 2.5.7 & DOS 6.2
LANGUAGE Used for Coding	'C' Language & TCL/TK 8.2

TCP/IP Protocol suite used for computer communication.

FTP Libraries , from FTP software(PC/TCP 3.2 for DOS), SDK

TCL/TK Libraries is used for designing GRAPHICAL USER INTERFACE

CONTENTS

CONTENTS

CHAPTER 1

ORGANISATION PROFILE

RAMAN RESEARCH INSTITUTE

BANGALORE 560 080

HISTORY AND PRESENT STRUCTURE OF RAMAN RESEARCH INSTITUTE

The Raman Research Institute was founded by Nobel laureate Sir C.V.Raman in 1948. The main activity of the institute was basic research in selected areas of physics which were of particular interest to Prof. Raman. The institute owes its origin to action of government of Mysore in gifting to the Indian Academy of Sciences a plot of land in Bangalore in December 1934. In the year 1956, Prof. Raman made an irrevocable gift to the Indian Academy of Sciences, of various movable and immovable properties for the use and the benefit of the Raman Research Institute.

AREAS OF RESEARCH:

ASTRONOMY AND ASTROPHYSICS

The Raman Research Institute has been active in the area of Astronomy/Astrophysics research for over two decades. The observational programmes have focussed mainly on Radio Astronomy. The Institute has been involved in the construction and operation of several major radio telescopes. Scientists at the Institute observe with these telescopes, as well as many other facilities across the world.

AREAS OF RESEARCH

- Liquid Crystals
- Theoretical Physics
- Optics

CHAPTER 2

SOFTWARE ENVIRONMENT AND TOOLS

- TCP/IP PROTOCOL SUITE
- TCL/TK 8.2
- FTP SOFTWARE
- UNIX & 'C' LANGUAGE
- CLIENT-SERVER ARCHITECTURE

TRANSMISSION CONTROL PROTOCOL & INTERNET PROTOCOL

Overview of TCP/IP :-

TCP/IP is one of the software packages that currently dominates UNIX data communications. It is the leading communications software for UNIX local area networks. The name TCP/IP refers to an entire set of data communications protocols. The suite gets its name from two of the protocols that belong to it : Transmission Control Protocol.

TCP/IP and the Internet :-

In 1969 the Defense Advanced Research Project Agency(DARPA) funded a research and development project to create an experimental packet switching network called ARPANET.

The experimental ARPANET was so successful that many of the organizations attached to it began to use it for daily data communications. The basic TCP/IP protocols were developed after ARPANET was operational. The network protocol in ARPANET is IP(Internet Protocol) which is connectionless and was designed to handle interconnection of vast number of WAN and LAN networks comprising the ARPA Internet. The ARPANET Transport Protocol is a connection oriented protocol called TCP(Transmission Control Protocol) and is used in Berkeley UNIX.

The Internet is the world wide collections of interconnected networks, which grew out of the original ARPANET, that uses Internet Protocol(IP) to link various physical networks into a single logical network. As TCP/IP is required for Internet connection, the large number of diverse new organizations recently added to the

Internet have spurred interest in TCP/IP. As more organizations became familiar with TCP/IP, they see that its power can be applied in other network applications. It was common for a site to use TCP/IP for communication over a local Ethernet, which UUCP for communication with remote computer sites.

TCP/IP Features :-

The popularity of the TCP/IP protocols on the Internet grew rapidly as they met an important need(world wide data communication) at the right time and they had several important features that allowed them to meet this need. These are :

- Open Protocol Standards, freely available and developed independently from any specific computer hardware or operating system. As it is so widely supported, TCP/IP is ideal for uniting different hardware and software even if there is no communication over the Internet.
- Independence from specific physical network hardware. This allows TCP/IP to integrate many different kinds of networks. TCP/IP can run over an Ethernet, Token Ring, a Dialup Line, an X.25 Net and virtually any other kind of physical transmission media. A
  common addressing scheme that allows any TCP/IP device to uniquely address any other device in the entire network, even if the network is as large as the world wide network
- Standardised high level protocols for consistent, widely available user services.

Protocol Standards :-

The open nature of TCP/IP requires publicly available standards documents. Information about TCP/IP protocols is published as Requests For Comments(RFC). RFCs contain the latest versions of the specifications of all standard TCP/IP protocols. RFCs contain a wide range of interesting and useful information, and are not limited to the formal specification of data communications protocols.

Data Communication Model :-

An architectural model developed by the International Standards Organization(ISO) is frequently used to describe the structure and function of data communication protocols. This architectural model is called Open Systems Interconnection(OSI) Reference model.

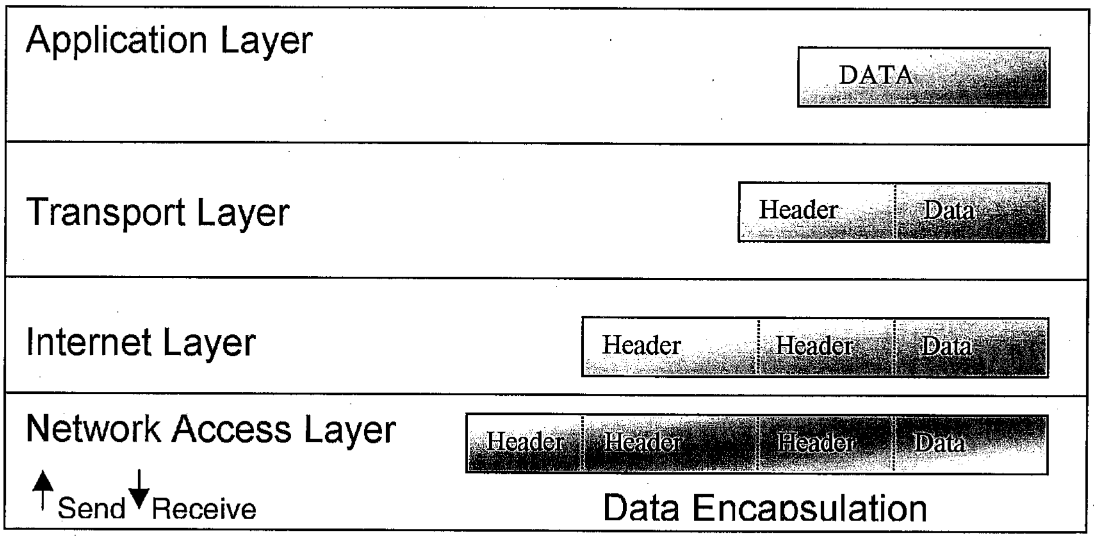
Although the OSI model is useful, the TCP/IP protocols don't match its structure exactly. The layered mode of TCP/IP is generally viewed as being composed of fewer layers than the OSI model. The four layer model of TCP/IP is as shown :

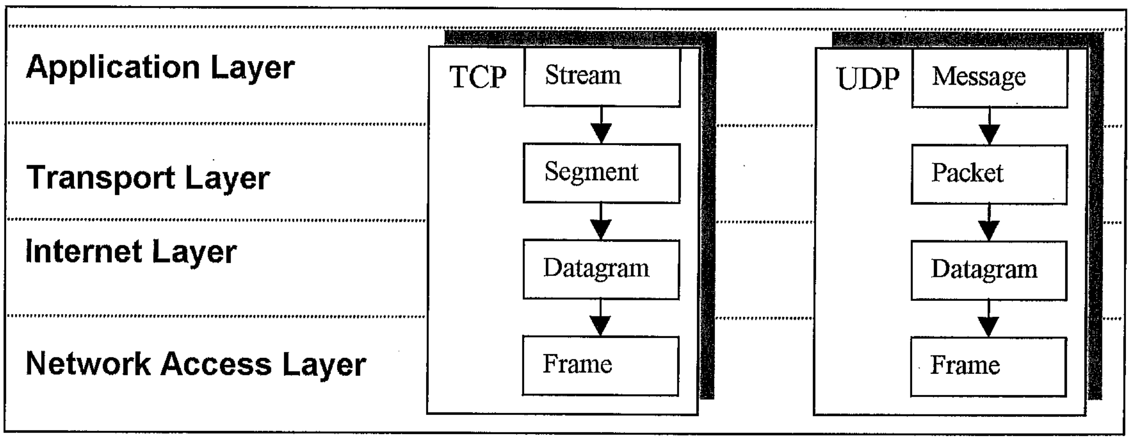
4. Application Layer consists of application and processes that use the Network.
3. Host-to-Host Transport Layer provides end-to-end data delivery services.
2. Internet Layer defines the datagram and handles the routing of data.
1. Network Access Layer consists of routines for accessing physical networks.

The four layered structure of TCP/IP is seen in the way data is handled as it passes down the protocol stack from application layer to the underlying physical network. Each layer adds its control information called the Header(to the data received from the previous layer) to ensure proper delivery. This addition of delivery information at each layer is called **Encapsulation.**

When data is received, opposite happens wherein each layer strips off its header before passing the data to the layer above.

Each layer has its own independent data structure and its own terminology to describe that structure. The figure below shows the terms used in different layers to TCP/IP to refer to the data being transmitted.





The functionality of each layer is as follows :-

- ***Network Access Layer*** **:** This is the lowest layer of the TCP/IP protocol hierarchy. The functions performed by protocols in this level include encapsulation of IP datagrams into frames transmitted by the network, and the mapping of the IP addresses to physical addresses used by the network. One of the TCP/IP 's strengths is its addressing scheme that uniquely identifies every host on the Internet. This IP address must be converted into whatever address is appropriate for the physical network over which the datagram is transmitted.

- ***Internet Layer*** **:** The layer above the Network Access layer in the protocol hierarchy is the Internet Layer. This layer provides the 'Virtual Network' image of Internet(i.e.., this layer shields the higher levels from the typical network architecture below it). The Internet Protocol is the heart of the TCP/IP and most important protocol in Internet layer. The other useful protocols of this layer are ARP, RARP, etc.

INTERNET PROTOCOL

The Internet protocol is a connectionless protocol and it relies on protocols in other layers to provide error detection and recovery, and is some times called unreliable protocol. The Internet Protocol is the building block of the Internet . Its functions include :

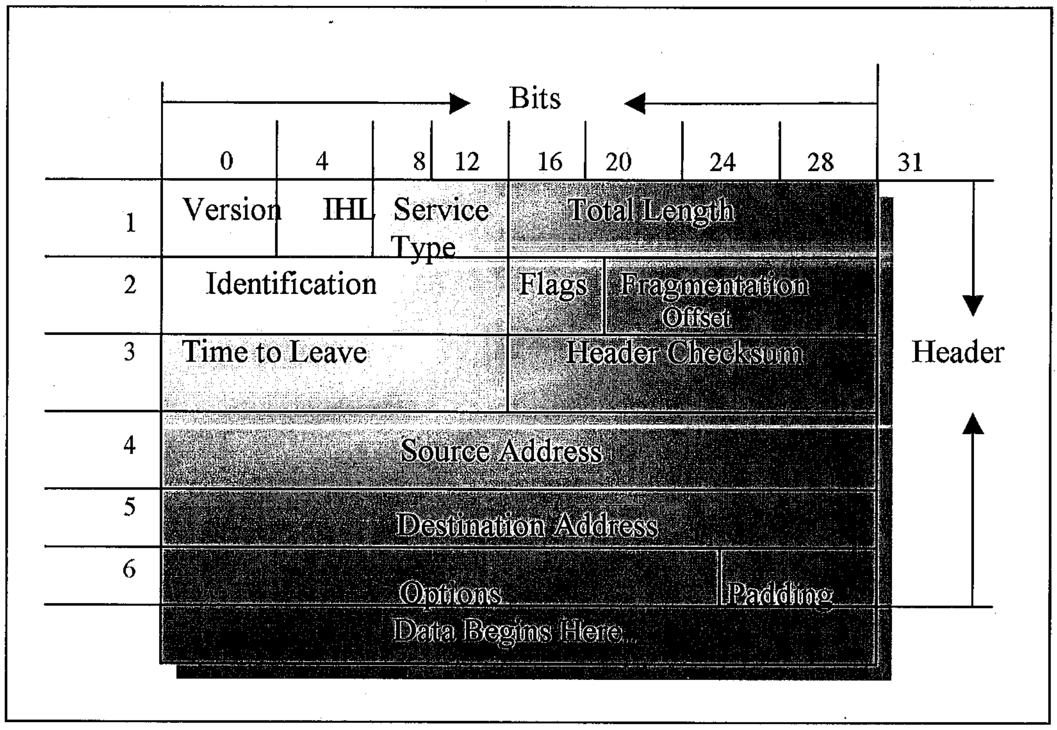
- defining the datagram, which is the basic unit of transmission in the Internet
- defining the Internet addressing scheme
- moving data between the Network Access Layer and the Host-to-Host Transport Layer
- routing datagrams to remote hosts
- performing fragmentation and re-assembly of datagrams

IP DATAGRAM

The Internet datagram(IP Datagram) is the base transfer packet in the Internet Protocol suite. It has a header containing information for IP and data that is only relevant to the higher level protocols. The IP datagram format is as shown below.

The five or six 32-bit words of the datagram are the control information called the Header. As header's length is variable it includes a field Internet Header Length(IHL), which indicates header's length in words.

The IP delivers the datagram by checking the destination address(32-bit IP address) in word 5 of the header. If destination address is address of a host on the local network, the packet is delivered directly to the destination, else the packet is passed to a Gateway for delivery.



Gateways are devices that switch packets between different physical networks. They move data between different protocols. Deciding which gateway to use is called Routing. A Router moves data between different networks.

The Hosts(or End-Systems) process packets through all four protocol layers, while the gateways(or Intermediate systems) process packets only up to the Internet Layer where the routing, decisions are made.

When an IP datagram travels from one host to another, it can cross different physical networks. Some physical networks may have a maximal frame size, called **Maximum Transmission Unit(MTU)** which will allow long IP datagrams to be placed in one physical frame. If the datagram received from one network is longer than the other network's MTU, it is necessary to divide the datagram into fragments for transmission. This process is called Fragmentation. The format of each fragment is the same as the format of any normal datagram.

IP ADDRESSING

Internet Protocol uses IP addresses to specify source and target hosts on the Internet. The Network address of the IP address is unique and is assigned by Network Information Center(NIC). They are 32-bit address, usually represented in dotted decimal form.

There are two logical addresses in each IP address : a Network address representing the physical network within the Internet, and a Host address specifying an individual host or gateway within the network.

IP address = <Network address><Host address>

The first bits of the IP address specify how the rest of the address should be separated in its network and host part. Four classes of IP addresses exists:

- Class A addresses provide for 128(infact 126) networks, each of which can have upto 2^{24}.
- Class B addresses allocate 14 bits for the physical network and 16 bits for the hosts in each of these networks.

- Class C address allocate 21 bits of the network ID and each can have upto 256 hosts.
- Class D addresses are reserved for multicasting(a sort of broadcasting, but in a limited area, and only to hosts having Class D address)

Thus Class A addresses will only be assigned to networks with a huge number of hosts, and Class C addresses to networks with a small number of hosts.

IP Routing

An important function of the IP layer is IP routing. It forms the basic mechanism for IP gateways, for interconnecting different physical networks. There are two kinds of routing: Direct Routing and Indirect Routing.

Direct Routing : If the destination host is attached to a network to which the source host is also attached, an IP datagram can be sent directly, by simply encapsulating the IP datagram in the physical network frame. The only addresses an individual host must know are the hosts attached to the 'direct networks'. This is determined by the IP network address part of the total IP address of the hosts.

Indirect Routing : This occurs when the destination host is not on a network directly attached to the source host. The only way to reach the destination is via one or more IP gateways. The address of the first of these gateways(first hop) is called indirect route. The address of the first gateway is the only information needed for the source host.

Each host keeps the set of mappings between:

- destination IP network address
- route to next gateways

in a table called **IP Routing Table.**

The three types of mapping found in the table are : Direct route, Indirect route and Default route.

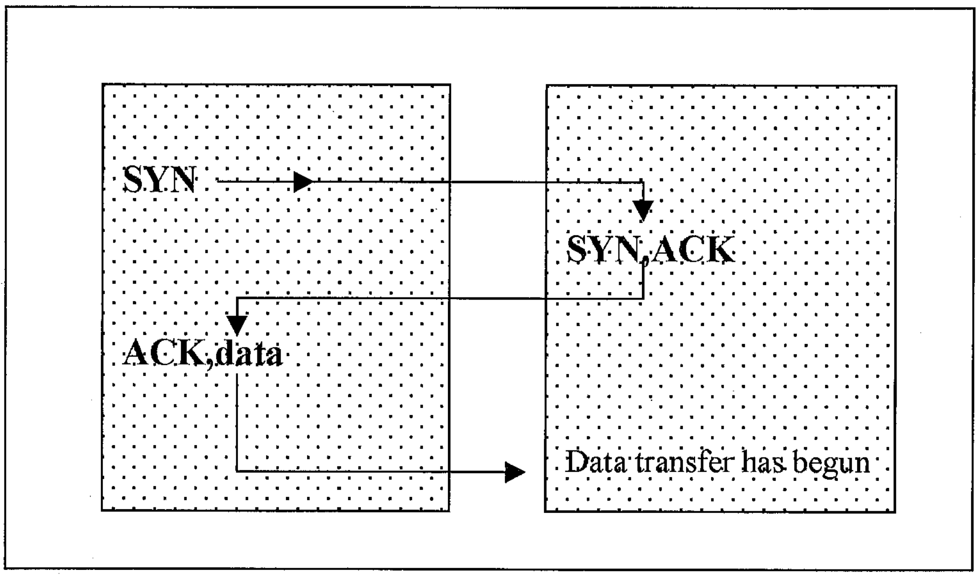
The default route specifies the route taken in case the destination IP network is not found in the direct or indirect mappings.

The steps that IP must follow in order to determine the route for an outgoing datagram is given by the IP Routing Algorithm. This is an iterative process. It is applied to every host handling the datagram, except for the host to which the datagram is finally delivered.

Thus IP is an important protocol of the Internet layer. However there are other useful protocols in the Internet layer such as Address Resolution Protocol(ARP), Reverse Address Resolution Protocol(RARP) etc.

- ***Transport Layer*** **:** This layer is just above the Internet Layer. This layer is also called Host-to-Host Transport Layer. The two important protocols in this layer are **Transmission Control Protocol(TCP) and the User Datagram Protocol(UDP).**

TCP is a connection oriented protocol. It establishes a end-to-end connection between the two communicating hosts. Control information called a hand shake is exchanged between the two end points to establish a dialog, before data is transmitted. The type of hand shake used by TCP is called a 3-way hand shake, because three segments are exchanged. The figure below shows the simplest form of the 3-way hand shake.

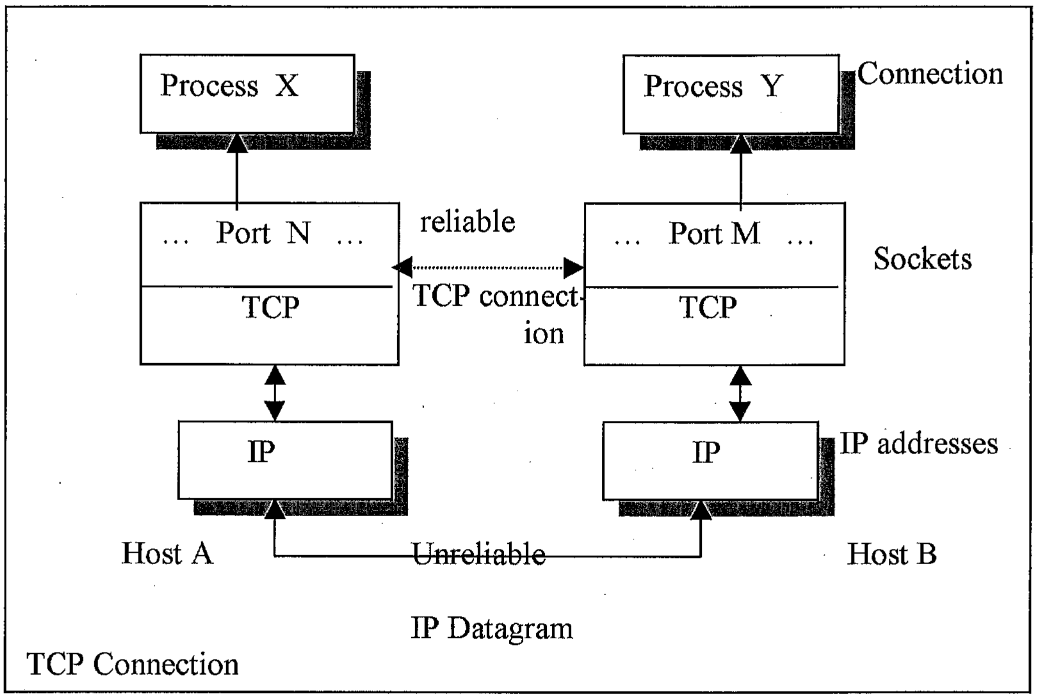


TCP can be characterised by the following facilities it provides for the applications using it.

1. ***Stream Data Transfer*** **:** TCP transfers a contiguous stream of bytes through the Internet. The application does not have to bother with chopping the data into basic blocks or datagrams. TCP does this by grouping the bytes in TCP segments, which are passed to IP for transmission to the destination. Also, TCP itself decides how to segment the data it may forward the data at its own convenience.
2. ***Reliability*** **:** TCP assigns a sequence number to each byte transmitted. TCP provides reliability with a mechanism called Positive Achnowledgement and Retransmission (PAR). The receiving TCP uses the sequence numbers to rearrange the segments when they arrive out of order, and to eliminate duplicate segments.
3. ***Flow Control*** **:** The receiving TCP, when sending an acknowledgement back to the sender, also indicates to the sender the number of bytes it can receive beyond the last received TCP segment, without causing overrun and overflow in its internal buffers. This is sent in the acknowledgement in the form of highest sequence number it can receive without problems. This mechanism is also referred to as a Window mechanism.

4. ***Multiplexing*** **:** This is achieved through the use of ports.
5. ***Logical Connections*** **:** The reliability and flow control mechanism require that TCPs initialize and maintain certain status information for each 'data stream'. The combination of this status, including sockets, sequence numbers and window sizes, is called a logical connection(or virtual circuit).

Each connection is uniquely identified by the pair of sockets used by the sending and receiving processes.



6. ***Full Duplex*** **:** TCP provides for concurrent data streams in both directions.

TCP is responsible for delivering data received from IP to the correct application. The application that the data is bound for is identified by a 16-bit number called the port number.

Source Port : The 16-bit source port number used by the receiver to reply.

Destination Port : The 16-bit destination port number.

Sequence Number : The sequence number of the first data byte in this segment. If the SYN control bit is set, the sequence number is the initial sequence number(n) and the first data byte is n+1.

Acknowledgement Number : If the ACK control bit is set, this field contains the value of the next sequence number that the receiver is expecting to receive.

Data Offset : The number of 32 bit words in the TCP header. It indicates were the data begins.

Reserved : 6 bit reserved for future bits must be zero.

URG : Indicates that the acknowledgement field is significant in this segment.

PSH : Push funtion.

RST : Resets the connection.

SYM : Synchronizes the sequence numbers.

FIN : No more data from sender.

Window : Used in acknowledgement segments. It specifies the number of data bytes beginning with the one indicated in the acknowledgment number field which the receiver is willing to accept.

Checksum : The 16-bit one's complement of the one's complement sum of all 16 bit words in a pseudo header, the TCP header and the TCP data. While computing the checksum, the checksum field itself is considered zero.

Urgent Pointer : Points to the first data octet following the 'urgent data'. Only significant when the URG control bit is set.

Options : They can be either

1. a single byte containing the option number
2. a variable length option in the following format

There are only three option fields defined :-

Kind	Length	Meaning
0	-	End of Option List
1	-	No-Operation
2	-	Maximum Segment Size.

This option is used during establishment of the connection(SYN control bit set) and is sent from the side that is to receive data to indicate the maximum segment length it can handle. If this option is not used, any segment size is allowed.

Padding : All zero bytes used to fil up the TCP header to a total length that is a multiple of 32 bits.

Application Layer : At the top of the TCP/IP protocol architecture is the Application layer. This layer includes all processes that use the transport layer protocols to deliver data. There are many application protocols, they communicate with applications on other Internet hosts and are user visible interface to the TCP/IP protocol suite.

Characteristics of Applications

All of the higher level protocols have some principle in common

1. They can be user made applications or applications standardized and shipped with the TCP/IP product. The TCP/IP protocol suite includes application protocols such as
    - TELNET(Teletypewriter Network) for interactive access to remote Internet Hosts.
    - FTP(File Transfer Protocol) for high speed disk to disk file transfers.
    - SMTP(Simple Mail Transfer Protocol) as an Internet mailing system.

These are the most widely implemented application protocols but a lot others exist.

2. They use either TCP or UDP as a transport mechanism. UDP is unreliable and offers no flow control, so in this case, the application to provide its own error recovery and flow control routines. It is often easier to build applications on top of TCP, a reliable connection oriented protocol.
3. Most of the Application protocols use client server model of interaction.

Client Server Model :

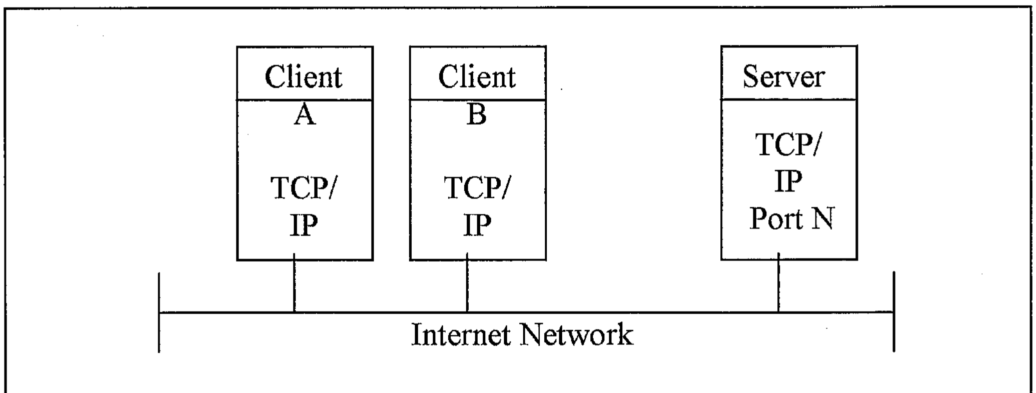
TCP is a peer to peer, connection oriented protocol. There are no master slave relations. The applications however use a client server model for communications.

A Server is an application that offers a service to Internet users ; A Client is a 'requester' of a service. An application consists of both a server and client part, which can run on the same or on different systems.

Users usually invoke the client part of the application, which builds a request for a particular service and sends it to the server part of the application using the TCP/IP as the transport vehicle.

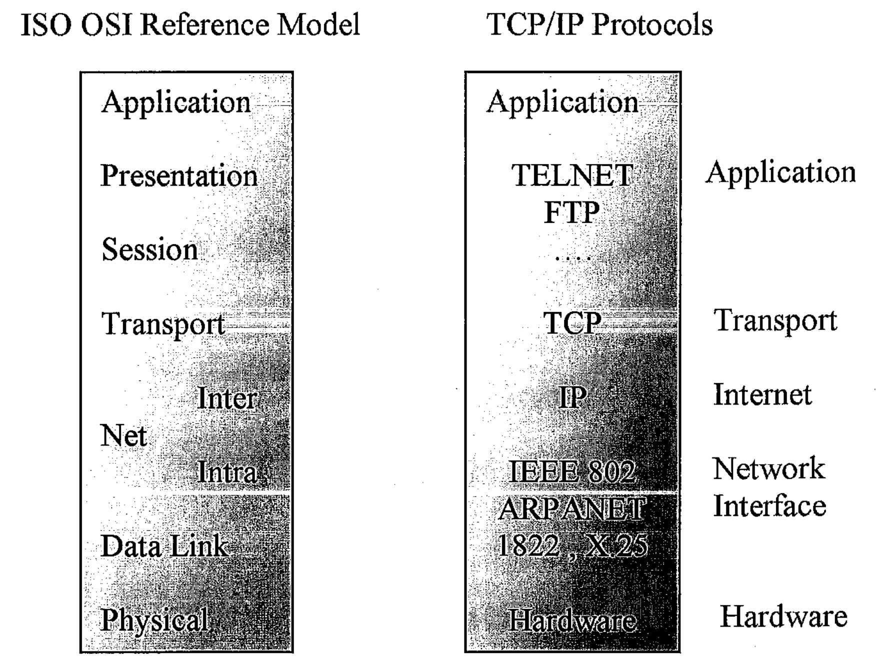
The server is a program that receives a request, performs the required service and sends back the results in a reply. A server can usually deal with multiple requests(multiple clients) at the same time.

Some servers wait for requests at a well known port so that theri clients know to which IP socket they must direct their requests. The client uses an arbitrary port for its communications. Clients that wish to communicate with a server that does not use a well known port must have anothr mechanism for learning to which port they must address their requests. This mechanism might employ a registration service such as a Portmap which uses a well known port.



The Client-Server Model of Applications

TCP/IP Model versus OSI Model :



TCP/IP and OSI. Functional Positioning of the Layers.

One can only functionally position the Internet model to the ISO OSI model because basic differences exist such as

- In the Internet Protocol suite, a layer represents a reasonable 'packaging' of functionality. For instance the transport layer has protocols like TCP(reliable interprocess communication, data are byte streams), but also UDP(simple demultiplexer on top of IP and therefore unreliable, no flow control, no error recovery). The TCP and UDP have the same concept of 'sockets' over which applications communicate(establish a logical connection).

  The ISO view, on the other hand treats layers as rather narrow functional groups attempting to force modularity by requiring additional layers for additional functions.

  In the Internet Protocol, a given protocol or a part of it can be used by other protocols within the same layer, whose OSI concept tends to define two layers in such occurences. Examples of such 'horizontal dependencies' are FTP, which uses the same common representation as TELNET on the ' application layer ' , and ICMP which uses IP for sending its datagrams on the 'Internetwork' Layer.

- End-to-End versus Link level reliability : Another major difference is reliability. With X.25, for instance, complex protocols guarantee the transfer between a host and the packet switch at the link level; while in TCP/IP intermediate IP gateways do not guarantee reliability. So, reliability is the responsibility of the ends of a connection, not of the intermediate machines.

- While the OSI has the notion that all 'layers N' must communicate with each other by means of the 'n-1 layers' below them, regardless of whether they are on the same or different hosts, the Internet model is not so strict. If the entities are on the same host, only one lower layer is necessarily traversed. This brings us to an even more fundamental difference.
- Efficiency and Feasibility : The OSI norms tend to be perspective(for instance the 'layer n' must go through all layers below it) whereas the TCP/IP protocols are descriptive, and leave a maximum of freedom for the implementers. One of

the advantages is that each particular implementation can use operating system dependent features, generally resulting
in a greater efficiency(fewer CPU cycles, more throughput for similar functions), while still ensuring 'interoperability' with other implementations. Most of the Internet Protocols has first been developed (coded and tested), before being described in a RFC which clearly shows the feasibility of the protocols.

Ports and Sockets :

Each process that wants to communicate with another process identifies itself to the TCP/IP protocol suite by one or more ports. A port is a 16 bit number, used by the host-to-host protocol to identify to which higher level protocol or application program(process) it must deliver incoming messages.

As some higher level programs are themselves protocols, standardized in the TCP/IP protocol suite, such as TELNET and FTP, they use the same port number in all TCP/IP implementations. Those 'assigned' port numbers are called **Well Known Ports** and the standard applications are called **Well Known Services**.

A Socket is a special type of file handle which is used by a process to request network services from the operating system.

A Socket address is a triple consisting of :

[protocol, local address, local process]

The socket interface is one of the several APIs to the communication protocols.

INTRODUCTION TO TCL/TK

Tcl, Tool Command Language, is an interpreted language with programming features, available across platforms running Unix, Windows and the Apple Macintosh operating system. Tk, the associated toolkit is an easy and efficient way of developing window based applications.

Application tasks are split into modules and any new application specific task is written and compiled as C or C++ program and exported as a new Tcl command. Then a Tcl script, a series of existing and new Tcl commands, is composed to make the overall application. The scripting language, much like any shell language, has the ability to access and execute any other programs. Therefore several Tcl based applications could be made to work together to create or extend into a new application.

Tcl consists of few syntax rules and a (still growing) set of core commands. Tk Provides a higher level application programming interface for developing interactive widgets based applications, particularly for those who wish to concentrate on the functionality of their application and have no need to gain indepth programming expertise in the underlying window system and/or verbose toolkits such as OSF/Motif. Tcl/Tk is free, available now on Apple Macintosh and Windows and has a wide user base with a rich and growing mass of useful contributed software.

The wider availabilty, usage and ease of teaching and learning of Tcl/Tk makes it the most appropriate tool for teaching the principles of Graphical User Interface design and development.

KEY FEATURES OF TCL/TK

- **Tcl is a high-level scripting language:**
  One need to write a lot less code to get the job done, especially when compared to Motif or Win32 applications.

- **Tcl is interpreted:**
  The code can be executed directly, without compiling and linking (through Tcl compilers are available).

- **Tcl is extensible:**
  It's very easy to add your own commands to extend the Tcl language. The commands can be written in C or Tcl.

- **Tcl is embeddable in your applications:**
  The Tcl interpreter is merely a set of C functions that can be called from the code. This means you can use Tcl as an application language, much like a macro language for a spreadsheet application.

- **Tcl runs on many platforms:**
  Versions exist for UNIX, Windows and Macintosh platforms. Except for a few platform differences, your Tcl scripts run the same on all systems.

- **Tcl's auto-loading facility makes for smaller applications:**
  Tcl will automatically load in the libraries of Tcl procedures (with one line of code to let up the path to your library). Tcl will also, on systems that support it, automatically load dynamic (shared, DLL) libraries when needed. This is very handy.

FTP

INTRODUCTION TO PC/TCP SYSTEM CALLS

The PC/TCP system calls provide the lowest- level application programming interface(API) to the PC/TCP kernel. The PC/TCP kernel implements the TCP/IP protocols for personal computers (PCs). The kernel is available as a DOS terminate-and-stay-resident (TSR) program. When the PC/TCP kernel loads, it installs as an Interrupt Service Routine (ISR) at Interrupt 0x61(by default).

The PC/TCP system calls, also known as native mode calls, provide an API to the TCP/IP transport and Internet Protocol Layers, and to the configuration and management of the running kernel. All of the other APIs the PC/TCP software development kit supports are built on top the system calls; that is, all routines that access the transport and internet protocol layers do so through internal system calls to the kernel.

As a C language interface for programming DOS applications, the PC/TCP system calls are in NETLIB.LIB library. This library is available for all memory models the PC/TCP SDK supports. The NETLIB.LIB library also contains name resolution and translation facilities, error reporting, and interrupt vector finding routines that are not system calls.

One can access function (NETLIB.LIB system calls) directly through interrupt instructions. The assembly language source code for NETLIB.LIB system calls is in the .ASM files in the PC/TCP SDK SOUCE \LIBRARY\NETLIB directory.

COMMUNICATING OVER A NETWORK

The TCP/IP suit of protocols provides layers of communication services to hosts on a network. For example, the Internet Protocol (IP) is the Internet layer of the TCP/IP protocols. Communication across a network occurs between the corresponding layers on the different network hosts, so that, for instance, the IP layer on one host communicate with the IP layer on another host. Communication between hosts is usually based on the client /server model, where in a client application sends a request for a service to a server application, which provides the service to the clients.

In native mode programming, communication between hosts is based on a *network descriptor.* A network descriptor is very similar to a socket in Berkley Software Distribution (BSD) UNIX operating system socket abstraction. A network descriptor is a integer value handle that identifies a specific connection or the end points for a communication and the protocol type to use for the communication. Some system calls use the network descriptor to affect or report on the attributes or status of the connection or communication.

An address structure associated with the network descriptor provides the kernel with following information:

- Protocol type
- Remote port
- Local port
- Remote IP address

The IP address of the local host is implied, since the PC/TCP kernel currently supports only one network interface on a machine.

The protocol types are either connectionless or connection oriented. The IP, ICMP & UDP protocols are connectionless. In using these protocols network hosts communicate by sending and receiving individual datagrams. The TCP Protocol is connection oriented; i.e, it creates a virtual connection between the local and remote host. The hosts communicate in a stream of data, and not by datagrams. Each host is identified by an IP address.

The end point for the UDP and TCP transport layers is a *port*, which is identified by a number. Some protocol services use reserved port numbers, called "well-known" ports. The PC/TCP SDK header file INCLUDE \PCTCP\SOCKETS.H contains well-known port numbers that native mode programmers use.
Refer to Appendices for FTP System calls.

INTRODUCTION TO UNIX

UNIX is a multi-user, multi-tasking, time-sharing operating system. An operating system called MULTICS(Multiplexed Information and Computer System) was developed in 1969 by Bell Telephone Laboratories along with General Electricals and a project team from Massachusetts Institute of Technology. The whole operating system was written in an assembly language. Later Thompson and Dennis Ritchie of Bell Laboratories modified the operating system to make it work on a DEC-PDP11 computer. In 1973, the two again rewrote the whole operating system in a high level language called 'C'. The O.S started to be known as UNIX(a pun on MULTICS).

UNIX is a popular operating system for a variety of reasons. Some of them are

- UNIX is Portable, i.e. the operating system is written in a high-level language, making it easy to read, understand, change and move to other machines.
- It has a simple user interface that has the power to provide the services that users want.
- It provides primitives that permit complex programs to be built from simpler programs.
- It uses a hierarchical file system that allows easy maintenance and efficient implementation.
- It uses a consistent format for files, the byte stream, making application programs easier to write.
- It provides a simple, consistent interface to peripheral device
- It is a multi-user, multi-process system; each user can execute several processes simultaneously.
- It hides the machine architecture from the user, making it easier to write programs that run on different hardware implementations.

One of the other most important reason is that UNIX has built in support for Networking i.e. UNIX has various programs and utilities that help in Networking.

UNIX has a central layer called Kernel(or System Kernel) which forms the heart of the Operating System. The Kernel interacts directly with the hardware, providing common

services to programs and insulating them from hardware idiosyncrasies. It has programs like Shell and Editors on the outer layer. The user interacts with the shell. The shell in turn interacts with kernel by invoking a well-defined set of system calls. Unix has several tools and utilities built along with the main kernel system.

UNIX has several systems calls, out of which certain system call like pipe, kill etc.. allows processes to exchange data and synchronize execution. Since UNIX is a time-sharing, multi-user, multi-tasking operating system it supports NETWORKING of systems to a great extent.

INTRODUCTION TO C

'C' is a general purpose language which features economy of expression, modern control flow and data structures and a rich set of operators. The absence of restrictions in 'C' and its generality makes it convenient and effective for many tasks than supposedly more powerful languages. 'C' was originally designed for and implemented on the UNIX Operating System on the DEC PDP-11. However 'C' is not tied to any particular hardware or system.

'C' provides a variety of data types. The fundamental types are characters, integers and floating point numbers of several sizes. In addition, there is a hierarchy of derived data types created with pointers, arrays, structures and unions. Expressions are formed from operators and operands; any expression including an assignment or a function call, can be a statement. Pointers provide for machine-independent address arithmetic. 'C' provides the fundamental control flow constructions required for well-structured programs; statement grouping, decision making(if..else..), selecting one of a set of possible cases (switch.. case), looping with the termination test at the top(while, for) or at the bottom (do..while) and early loop exit(break).

Functions may return values of basic types, structures, unions or pointers. Any function may be called recursively. Local variables are typically "automatic", or created anew with each invocation. Function definitions may not be nested but variables may be declared in a block-structured fashion. The functions of
a C program may exist in separate source files that are compiled separately. Variables may be internal to a function, external but known only within a single source file, and conditional compilation.

A pre-processing step performs macro substitutions on program text, inclusion of other source files, and conditional compilation.

'C' is relatively "low level" language. 'C' offers only straight-forward, single-thread control flow: tests, loops, grouping and subprograms, but not multiprogramming, parallel operations, synchronization or co-routines. A library is defined along with 'C'. It specifies functions for accessing the operating system, formatted input and output , memory allocation, string manipulation, etc.. . A collection of standard headers provides uniform access to declarations of functions and data types.

Although 'C' matches the capabilities of many computers, it is independent of any particular machine architecture and hence it is easy to write portable programs using 'C'. 'C' is not a strongly typed language, but as it is evolved, its type-checking has been strengthened.

'C' like any other languages has its blemishes. Some of the operators have the wrong precedence; some parts of the syntax could be better. Nonetheless, 'C' has proven to be an extremely effective language for a wide variety of programming applications.

BASIC DATA TYPES AND THEIR SIZES

• char	:	single byte, capable of holding one character.
• int	:	holds one integer value.
• float	:	single-precision floating point.
• double	:	double-precision floating point.

COMMON CONSTRUCTS

1. IF..ELSE Statement

```
if (Expression)
        Statement1;
else
        Statement2;
```

2. SWITCH..CASE Construct

```
switch (Expression)
{
        case Constant-Expression : Statement1;
        case Constant Expression : Statement2;
        default                  : Statement3;
}
```

3. WHILE Statement

```
while (Expression)
        Statement;
```

4. FOR Construct

```
for ( Expression1; Expression2; Expression3)
        Statement;
```

5. DO..WHILE Construct

```
do
        Statement;
while (Expression);
```

FUNCTIONS

Functions break large computing tasks into smaller ones, and enable the programmers to build programs in a modular fashion. 'C' has been designed to make functions efficient and easy to use. The function definition, declaration constructs are shown below.

Declaration of a Function

```
return-type Function-Name(arguments declaration)
```

Definition of a Function

```
return-type Function-Name(arguments declaration)
{
        variable and prototype declarations.
        Statements of a function.
        return expression;
}
```

return expression is used to return a value from the function.

CLIENT SERVER MODEL

From the viewpoint of an application, TCP/IP like most other computer communication protocols merely provides basic mechanisms used to transfer data. In particular, TCP/IP allows a programmer to establish communication between two application programs and to pass data back and forth. Thus, we say that TCP/IP provides peer-to-peer communication.

Although TCP/IP specifies details of how data passes between a pair of communicating applications, it does not dictate when or why peer applications interact, nor does it specify how programmers should organize such application programs in a distributed environment. In practice, one organizational method dominates the use of TCP/IP to such an extent that all applications use it. This method is known as *CLIENT SERVER PARADIGM.*

The client server paradigm divides communicating applications into two broad categories depending on whether the application waits for communication or initiates it.

CLIENT AND SERVERS :

The Client-Server paradigm uses the direction of initiation to categorize whether a program is a Client or Server.

In general an application that initiates peer-to-peer communication is called a CLIENT. End users usually invoke client software when they use a network service most client software consists of conventional application programs. Each time a client application executes, it contacts a server, sends a request and awaits a response. When the response arrives, the client continues processing. Clients are often easier to build than servers and usually require no special system privileges to operate.

By comparison, a server is any program that waits for incoming communication requests from a client. The server receives clients request performs the necessary computation and returns the result to the client.

PRIVILEGE AND COMPLEXITY :

Because servers often need to access data, computations, or protocol ports that the operating system protects, server software usually requires special system privileges. Because a server executes with special system privileges, care must be taken to ensure that if does not inadvertently pass privileges to the clients that use it. For example a file server that operates as a privileged program must contain code to check whether a client can access a given file. The server cannot rely on the usual operating system checks because its privileged status overrides them.

Servers must contain code that handles issues of:

- **AUTHENTICATION:** verify the identity of the client.
- **AUTHORISATION:** determining whether a given client is permitted to access the service the server supplies
- **DATA SECURITY:** guaranteeing that data is not unintentionally revealed or compromised.
- **PRIVACY:** keeping information about an individual from unauthorized access.
- **PROTECTION:** Guaranteeing that network applications cannot abuse system resources.

The combination of special privileges and concurrent operation usually makes servers more difficult to design and implement than clients.

CLASSIFICATION OF CLIENT SOFTWARE :

Client application programs can be classified into two classes:

1. STANDARD APPLICATION SERVICES: consist of those services defined by TCP/IP and assigned well-known, universally recognized protocol-port identifiers.
2. NON-STANDARD APPLICATION SERVICES: consist of all those services other than the standard application services.

ITERATIVE vs CONCURRENT SERVERS :

In case of Concurrent Servers, the server doesn't know how much time is required to service the request made by the client beforehand. For eg. To print a file in a shared printer. The amount of time required to service the request depends on the type of the request made. For eg. Say size of file in case of printing of a file. Therefore in these cases the Servers are coded in a concurrent fashion i.e the Server creates a sub-process and service of request is taken over completely by that child process that is created.

In case of Iterative Servers, the server's most of the time know the amount of time required to process a request. For eg. Getting time of day. The iterative server cannot process more than one request at a time and hence any request made while the server is servicing a request has to wait until it comes back to sleep.

CONNECTIONLESS Vs CONNECTION ORIENTED SERVERS :

When programmers design client-server software, they must choose between two types of interaction:

- A Connection less style
- A Connection Oriented style

The two styles of interaction correspond directly to the two major transport protocols that the TCP/IP protocol suite supplies. If the client and server communicate using UDP, the interaction is connectionless. If they use TCP the interaction is connection oriented.

STATELESS Vs STATEFUL SERVERS :

Information that a server maintains about the status of ongoing interactions with clients is called STATE INFO. Servers that do not keep any state info are called Stateless servers; others are called Stateful servers.

CHAPTER 3

INTRODUCTION

A GUI FOR REMOTE CONTROL OF
HEALTH MONITORING MODULES
IS A PROJECT WORK
CARRIED OUT AT
RAMAN RESEARCH INSTITUTE
BANGALORE

INTRODUCTION

RRI has 10.4M Radio telescope using which Radioastronomical observations are made.

Softwares for Radioastronomical Observations with 10.4M telescope.

The Software for the radioastronomical observations with 10.4M telescope are divided as workstation software with UNIX_Solaris 2.7 as platform and PC softwares which are developed on MSDOS 6.2. The process of Observations are automised by the use of workstation and pc softwares. The sunworkstation acts as a coordinator. The subsystems like frequency switch system,beamswitch box, backends, high resolution spectrometers, inclinometers, telescope motors and many other decvices are controlled and monitored by the various computers. These computers also acquires data from backends.

The computer systems that are involved in observation and their tasks are listed below.

Sun Workstation SunSolaris 2.7 :Used as a coordinator

Control PC MSDOS 6.2 : Used mainly for tracking, totalpower data acquisition, controlling beam switch system and time keeping by interfacing with astronomical lock.

Health Monitor PC MSDOS 6.2 :Used for controllig frequency switch systems and monitors the health of various systems like

PC MSDOS 6.2 Used for data acquisition from system –data acquisition from Filter banks and AOS.

High Resolution Spectrometer PC MSDOS 6.2 Used for controlling and acquiring data from a high resolution spectrometer.

Correlator OS 9 Correlator

A typical observation :

An obsfile which indicates method in which observation should be done is prepared and fed to the main observing software in the workstation. This software then communicates with the various above mentioned systems to bring about the necessary actions. The data is stored back in the workstation and later is analyzed with unipops - an data reduction package.

Types of observation and the softwares used.

1. Spectral line observations
    a) millimeter wave observation - Software, Data acquisition software
    b) 21cm observations - Data acquisition software
    c) 6.7GHz observations- Data acquisition software.

This project entitled " **A GUI FOR REMOTE CONTROL OF HEALTH MONITORING SYSTEM** " mainly deals with Automation of Health Monitoring System for Spectral Line observation of the 10.4 meter Radio Telescope. This is done by establishing communication between the Health Monitoring System which monitors the telescope and the Workstation which checks for error data and displays the current status of communication.

The health monitoring system receives the Radio signals from the three Monitor Control Modules . Each of these Monitor Control Modules, namely CABIN, TBCC, RX-ROOM have 32 channels.

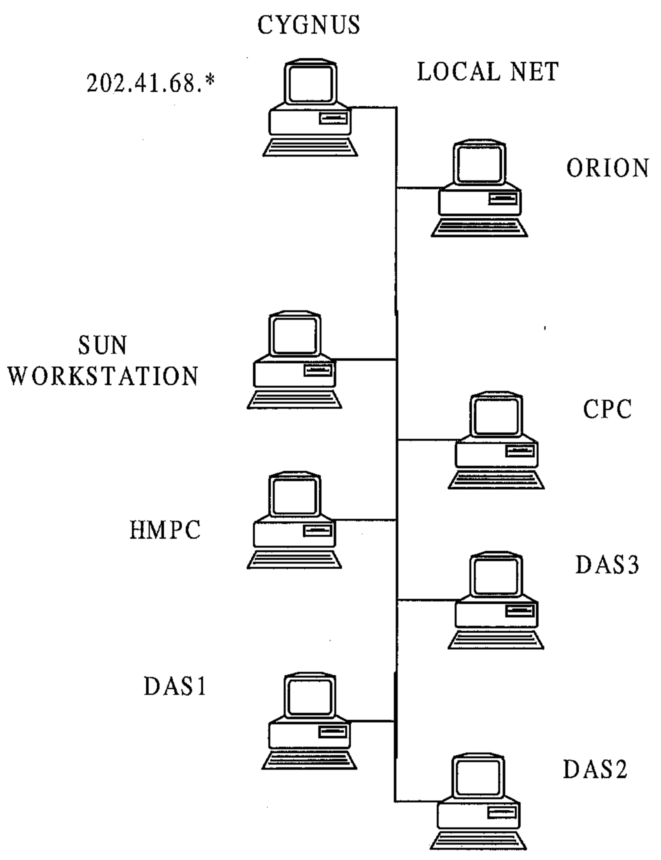
The main objective of the project is to configure the monitoring modules thorough Health Monitoring system. The values (voltage) of the received signals are compared with the configured values. In case the received values are at error then it is communicated to the Workstation. The Workstation asks for the error data which is sent by the Health Monitoring System. Depending on the requirement of the user the error data is logged into a desired file which can be used for further analysis.

Requirement of remote control of Monitor Control Module

The already existing Health Monitoring System receives the signals from control modules of the Telescope. There is no provision for reporting the error immediately after comparing these values with the configured values and also there was no interface for the user at the Workstation. As a result of which even in the wake of error, which is not detected, the receival of signals will continue until the end of the observation . Due to this the whole of the observation had to be redone.

By remote controlling the monitoring modules, comparison between the received values and the configured values is made and the error is brought to the notice of the user.
The monitoring module and the particular channels at error is also brought to the notice of user, through which the user can do further analysis. All these are done through a Graphical User Interface which is more appealing to the user.

OBSERVATORY COMPUTER SYSTEMS



CHAPTER 4

PROBLEM ANALYSIS AND DESIGN

- REQUIREMENT ANALYSIS PHASE
- DESIGN AND IMPLEMENTATION

REQUIREMENTS ANALYSIS PHASE

The requirement analysis phase is the first phase of any software project. During requirement analysis important data and information needed for the successful completion of the project is collected. The output of this phase is the Software Requirements Specification document (SRS).

The requirements were as follows:

I. **A Graphical user interface for displaying the Main window, to do the following functions:**

- An interface to 'Create', 'Edit' a configuration file
- An Interface to establish the connection between the WorkStation and the Health Monitoring System for communication.
  - ➢ Send the Configuration file to Health Monitoring System
  - ➢ Send 32 bits of data to WorkStation
  - ➢ Receive error values from Health Monitoring System.
  - ➢ Send 192 bytes data to the WorkStation in case of error.
- Display the current status of communication.

II. **Establish the communication between Health Monitoring System and the Workstation.**

- ➢ Send configured file to Health Monitoring System.
- ➢ Receive the configured file at workstation.
- ➢ Send signal data from Health Monitoring System to workstation.
- ➢ Receive error values from Health Monitoring System.

III. **Display the values, either the error values or the configured values**

IV. **Log the results into a file with current date and time**

The conclusion from the requirement analysis phase was:

To develop software with user-friendly GUI which would allow to configure a Monitoring Control Module file. Establish communication between Health Monitoring System and Workstation, send and receive radio Signal , display the configured values, check for error in the data at the Workstation. In case of error, read error data from Health Monitoring System and display the values. Log the data into file if required.

DESIGN AND IMPLEMENTATION

The pre-requisites for implementation of the software 'A GUI FOR REMOTE CONTROL OF HEALTH MONITORING SYSTEM' is designing the following

- Client-Server Model.
- Create a user-friendly Graphical Interface.

CLIENT SERVER MODEL

Client-Server Model is a standard model for Network Communication. The roles of the Client and Server process are *Asymmetric*. This means that both the halves are coded differently.

The SERVER process is started first and typically does the following steps.

- Open a communication channel and inform the local host of its willingness to accept the client request on some well-known addresses.
- Wait for a client-request to arrive at a well-known address.
- For an Iterative Server process the request and send the reply. Iterative Servers are typically used when a client request can be handled within a single response from the server
- For an Concurrent Server a new process is spawned to handle the this client request. This requires a *fork* and possibly *exec* under UNIX. This new process then handles this client request and does not have respond to other client requests. When this new process is finished it closes its communication channel with the client and terminates.
- Go back to step 2 and wait for another client request.

The CLIENT process performs a different set of actions. They are

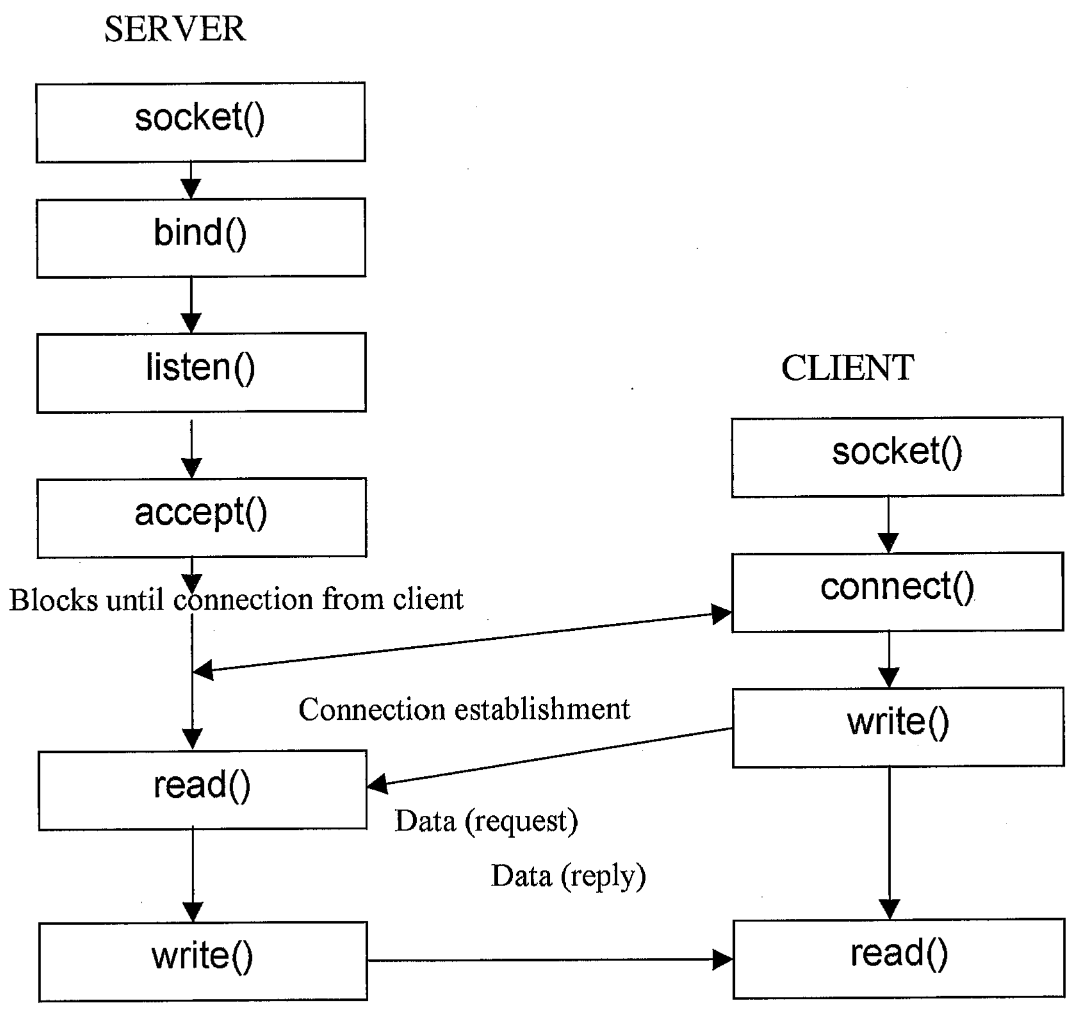
- Open a communication channel and connect to a well-known address on a specific host (i.e. Server)

- Send service request messages to the server and receive the response, continue doing this as long as necessary.
- Close the communication channel and terminate.

When a server opens a communication channel and waits for a client request we call this a PASSIVE OPEN.

The client on the other hand executes what is called an ACTIVE OPEN, since it expects the server to be waiting.

We have employed the Client-Server model in our project and the general design structure for the client-server model is as below



IMPLEMENTATION OF THE ABOVE MODEL

For the networking TCP/IP socket programming is used. The Operating platforms present in the Health Monitoring System (Dos) and Workstation (Solaris) are different .TCP/IP socket programming was used for Communication from the Workstation side and PC/TCP 3.2 for DOS Software Development Kit on the Health Monitoring System.

Since communication was to be established between two processes in the Workstation, Unix Domain Socket programming was used.
GUI was developed using TCL/TK for the following reasons

It is a platform independent GUI developing tool.
It is developed using 'C' libraries.
It can spawn 'C' functions.

APPLICATION GUI(TCL/TK)
NETWORK (TCP/IP SOCKETS) **SOLARIS 2.7**
FTP **DOS 6.2**

For the communication between Health Monitoring System and Workstation Internet Domain Socket Communication is used in which Health Monitoring System is the Server and Workstation is the Client. Since there exists a delay in establishing the connection between Health Monitoring System & Workstation , this connection has to be permanently present until the user closes the connection. The Internet domain socket descriptor has to be used to send and receive data between WorkStation and Health Monitoring System. The process which wants to communicate has to receive the Internet

Domain Socket descriptor from Client. The passing of the Internet Domain Socket descriptor between the Client of the Internet Domain Socket communication & the process which wants to communicate with the Health Monitoring System is done using UNIX Domain Socket Communication . In this case the client for the Internet Domain Socket communication acts as a server for Unix Domain Socket communication & the process communicating with the Health Monitoring System acts as a client for Unix Domain Socket communication.

When the communication takes place for the first time the selected configuration file (.cfg) is sent to the Health Monitoring System. The values in the configuration file received by the Health Monitoring System is compared with the values received from the three monitoring modules (CABIN, TBCC, RX-ROOM) . The result of the comparison may be an error or no error which is indicated by a '1' or '0' respectively. For each of the 32 channels of the monitoring modules , the result of the above comparison is sent to the Workstation.

The process of comparison and sending the compared result to the Workstation takes place every 'One second' .The Workstation on receival of the compared result checks for a '1' in the result. On encountering a '1', sends an error message to the Health Monitoring System. In turn the Health Monitoring System sends the 192 bytes of data (2 bytes * 32 channels *3 modules), which is received from the monitoring modules, to the Workstation. The software displays the received values highlighting the channels and the modules at error at the workstation.

GRAPHICAL USER INTERFACE

Graphical User Interface is an important part of any software. GUI was developed using TCL/ TK (Tool Command Language/Tool Kit). The routines provided by TCL/TK for the creation of buttons, labels , entry textboxes etc and the event handling routines were used. The basic building blocks used in TCL/TK are **widgets.** One of the most useful extensions to Tcl is Tk , which is a toolkit. Tcl is an interpreted language where new scripts can be generated and executed on the fly without recompiling or restarting the application.

I. A Graphical user interface for displaying the Main window, consisting of various control buttons namely

For implementing the requirement of creating & editing the configuration file the following widgets are designed.

- MCM FILE : When this button is clicked it pops up a window with the following widgets.
  - ➢ CREATE : Pops up a window and helps in creating a new configuration file. This window has the options to
    1 'Save', save the newly created configuration file.
    2 'Save As', save the already created file under a new name.
    3 'Reset', reset all the entries to reset values as specified .
    4 'Default', set all entries to default values so as to make any necessary changes in it to create a new file.
    5 '>>', next button, to browse through the next eight channels.
    6 '<<', back button, to browse through the previous eight channels.
    7 '@', help to go the next Monitoring Module.
    8 'Exit', close the window.

- EDIT : Pops up a window and helps in editing an existing file. This Window is same as window which pops up when 'Create' button is clicked. In addition there is provision to open an existing file and edit (change) that file. For configuring the data , Entry widgets are used. Vertical Scrollbars are used for incrementing or decrementing the data values.
- CLOSE : Closes the Create/Edit window.

To implement the requirement of connecting the Health Monitoring System and Workstation the following widget Button was used.

- CONNECT : To establish the connection between the WorkStation and the Health Monitoring System. The connection takes place as explained in the in the implementation details of client server model .

To implement the requirement to send and receive the data between the Health Monitoring System and Workstation the following widget Button was used.

- SEND/RECEIVE MCM DATA When this button is clicked, forks a process to communicate with Health Monitoring System. Pops up a window to display either configured values in case of no error or the values(error data) received from Health Monitoring System in case of error.

To display the current of status of communication the following widget was used

- STATUS- Displays the current status of communication.

  Depending on the present status one of the following message is displayed.

  1 Connection is not established: There exists no connection between Health Monitoring System and workstation.

2 Connection is established: There exists a connection between Health Monitoring System and workstation but data communication is not taking place.

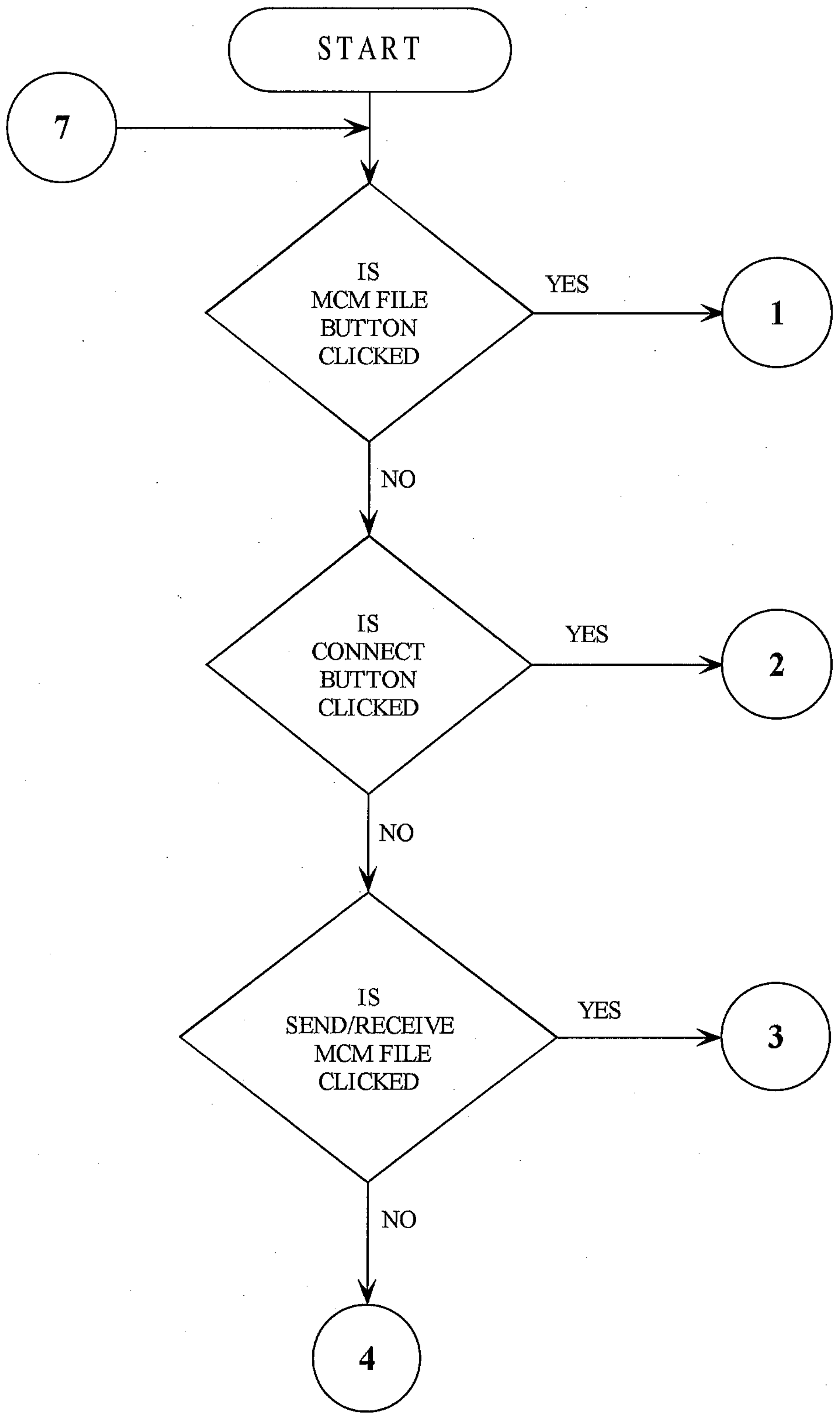
3 Communication is in progress: Data communication is taking place.

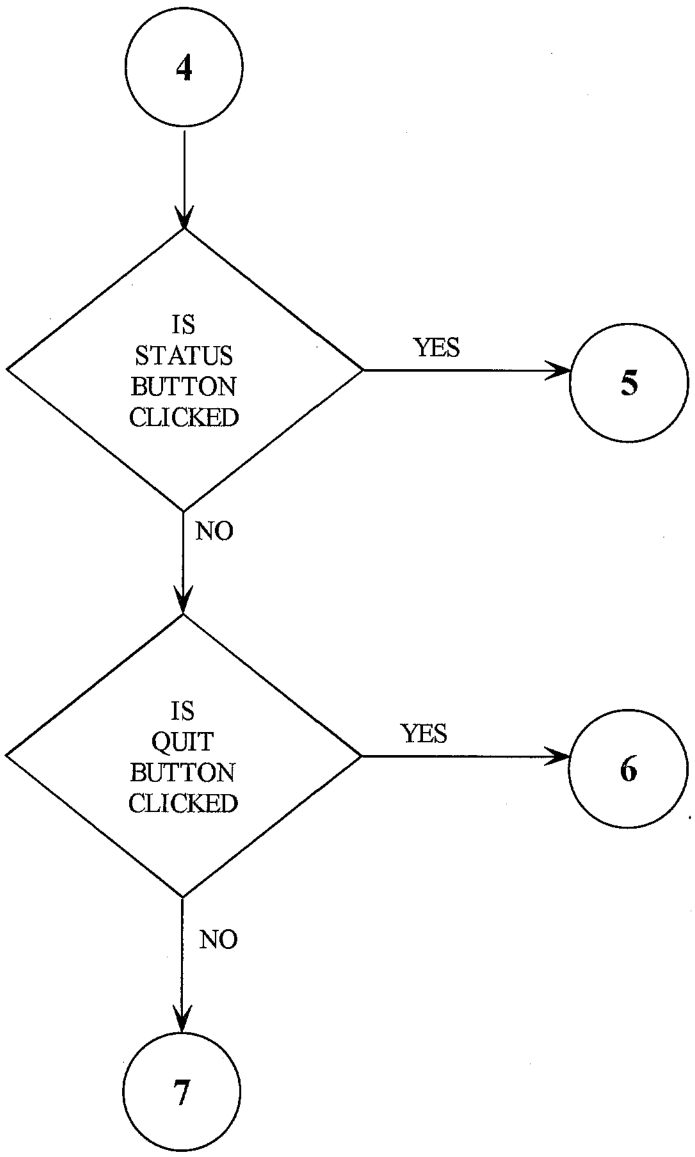
- QUIT -Closes the connection between the Health Monitoring System and the WorkStation and closes all the windows. Tk_messageBox which is a toplevel widget is used to display the message that the system is quitting.

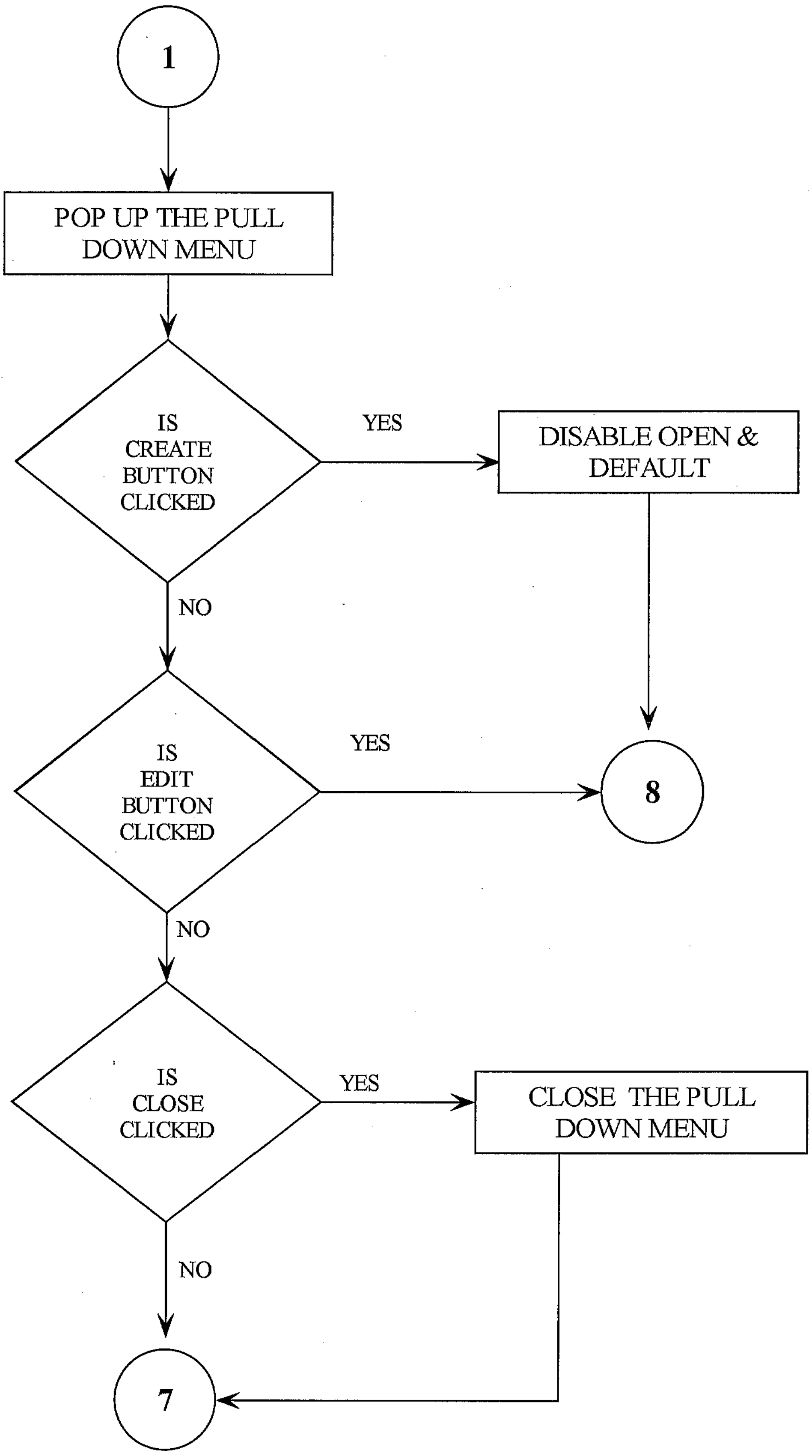
IMPLEMENTATION DETAILS

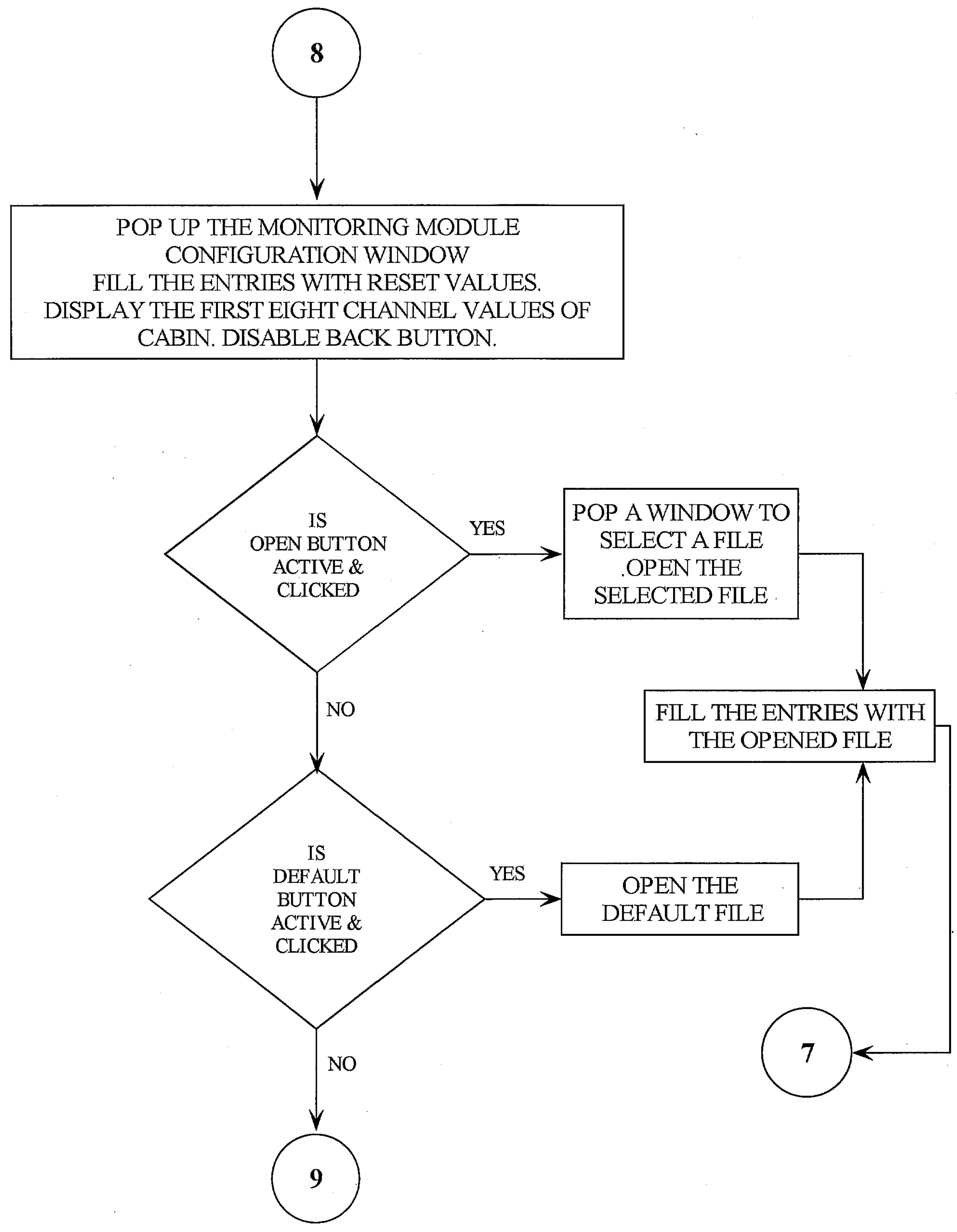
The first step in the implementation of the software was to design a flow chart. The following flow charts were the first step towards the implementation

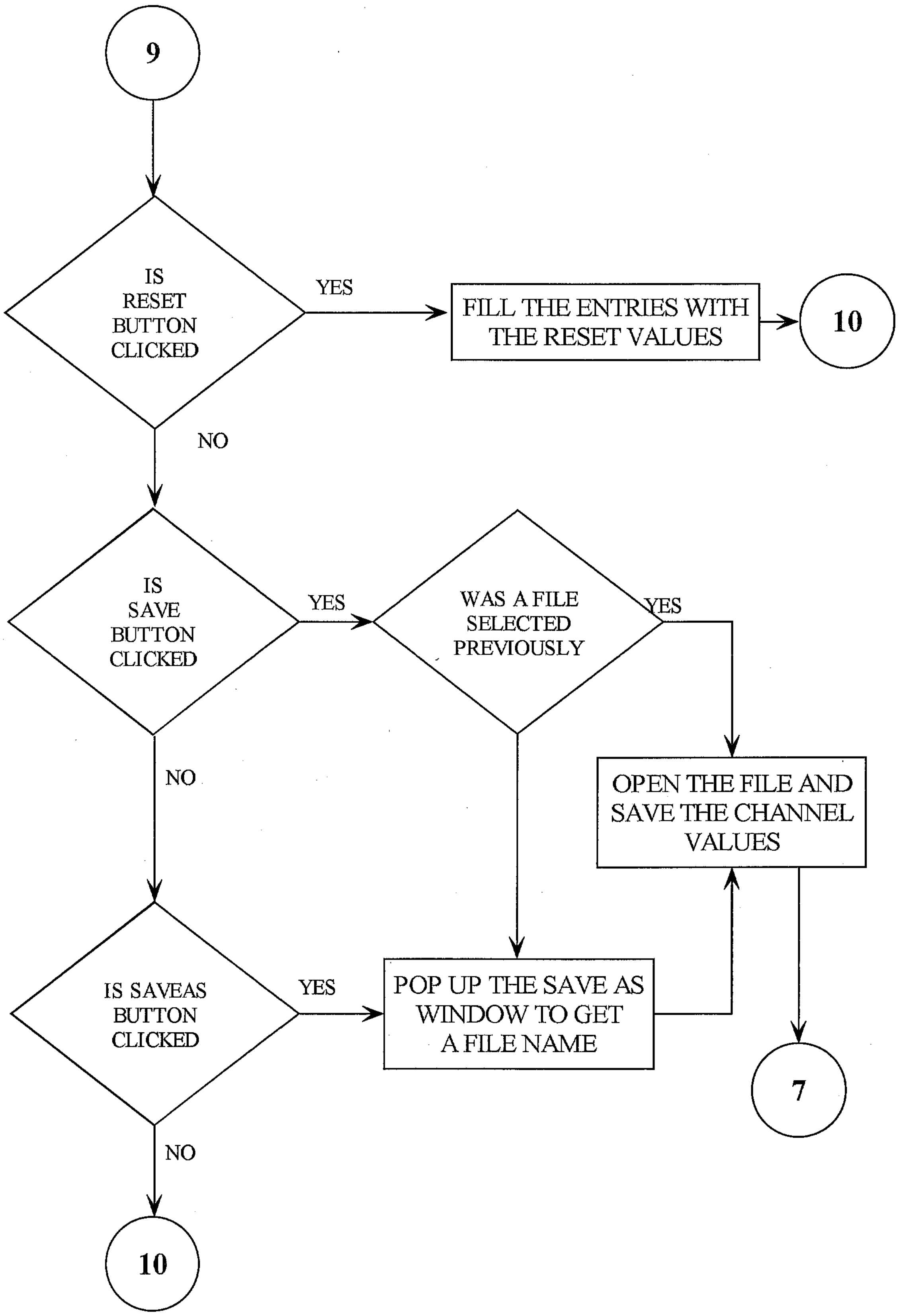
FLOW CHART

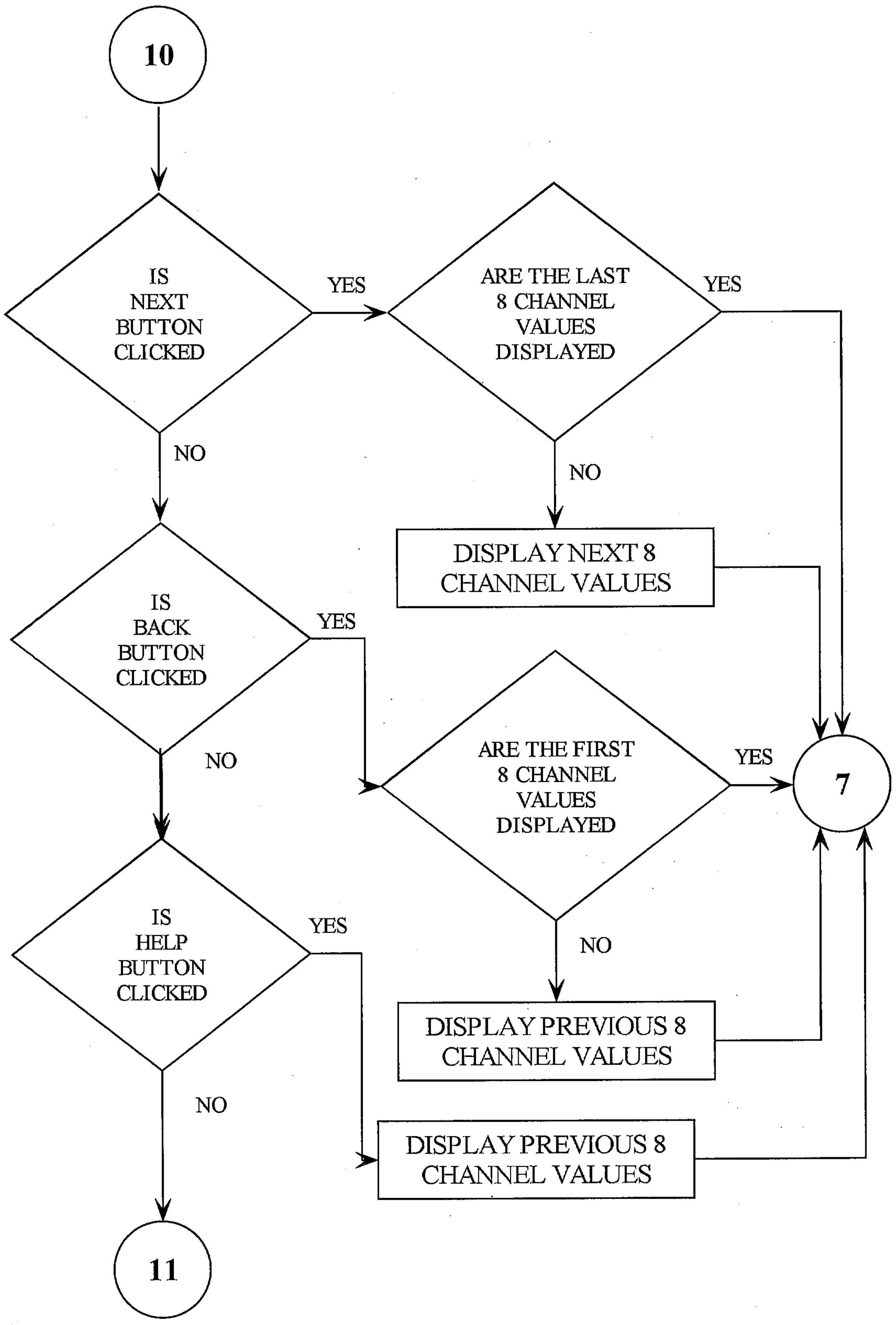

START
7
IS MCM FILE BUTTON CLICKED
YES
1
NO
IS CONNECT BUTTON CLICKED
YES
2
NO
IS SEND/RECEIVE MCM FILE CLICKED
YES
3
NO
4

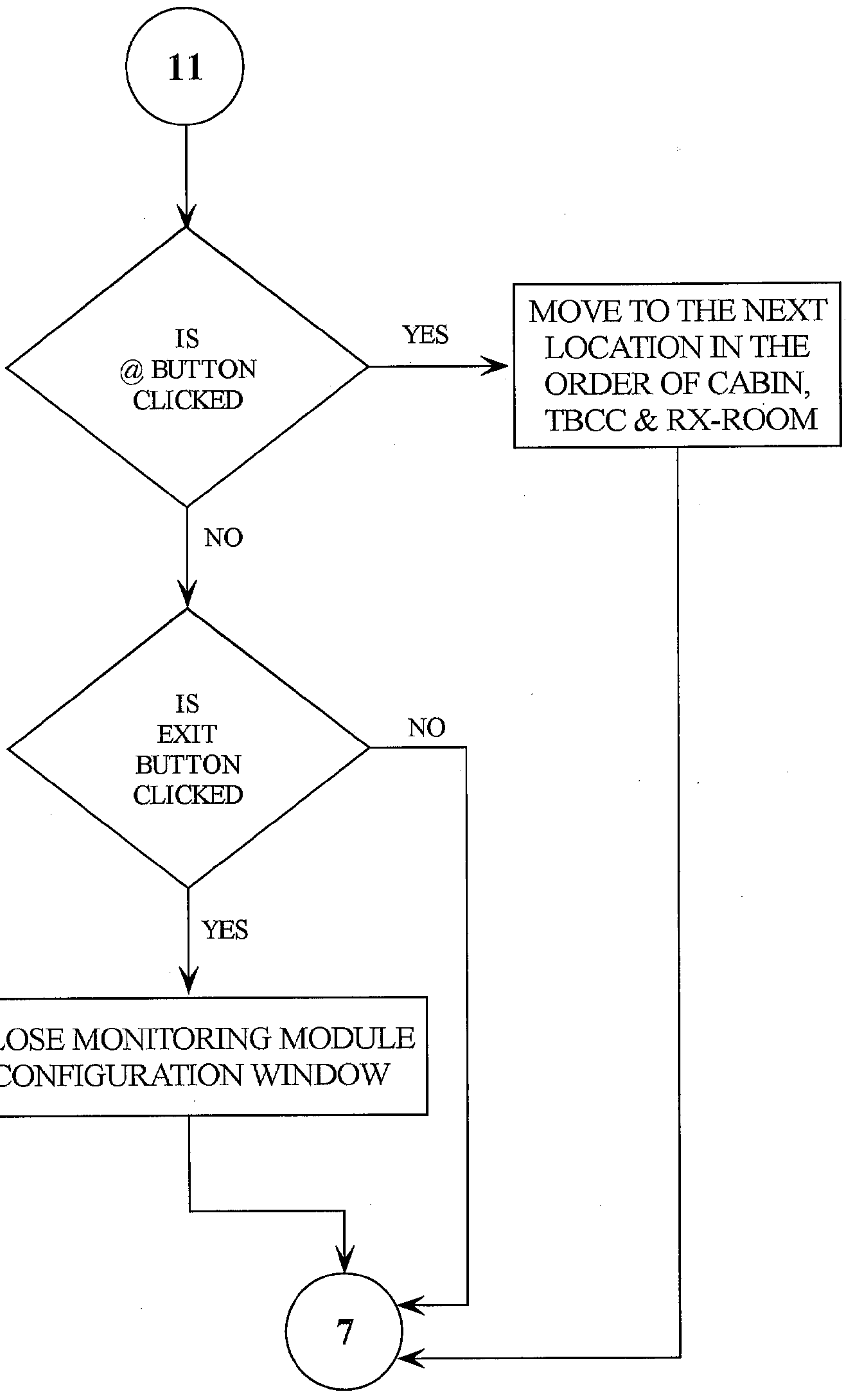

4
IS STATUS BUTTON CLICKED
YES
5
NO
IS QUIT BUTTON CLICKED
YES
6
NO
7

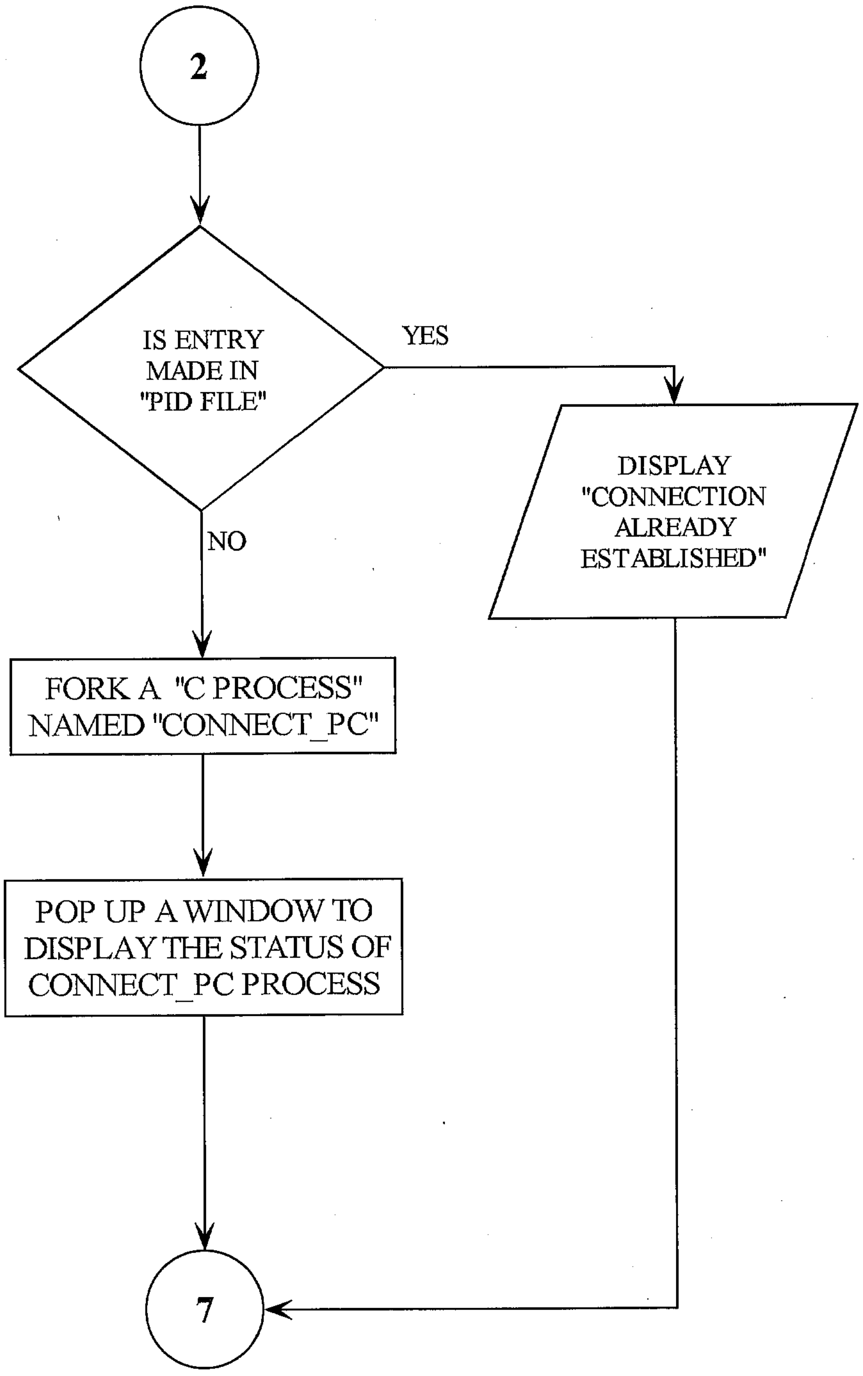

1
POP UP THE PULL DOWN MENU
IS CREATE BUTTON CLICKED
YES
DISABLE OPEN & DEFAULT
NO
IS EDIT BUTTON CLICKED
YES
8
NO
IS CLOSE CLICKED
YES
CLOSE THE PULL DOWN MENU
NO
7


8
POP UP THE MONITORING MODULE CONFIGURATION WINDOW
FILL THE ENTRIES WITH RESET VALUES.
DISPLAY THE FIRST EIGHT CHANNEL VALUES OF CABIN. DISABLE BACK BUTTON.
IS OPEN BUTTON ACTIVE & CLICKED
YES
POP A WINDOW TO SELECT A FILE .OPEN THE SELECTED FILE
NO
FILL THE ENTRIES WITH THE OPENED FILE
IS DEFAULT BUTTON ACTIVE & CLICKED
YES
OPEN THE DEFAULT FILE
NO
7
9

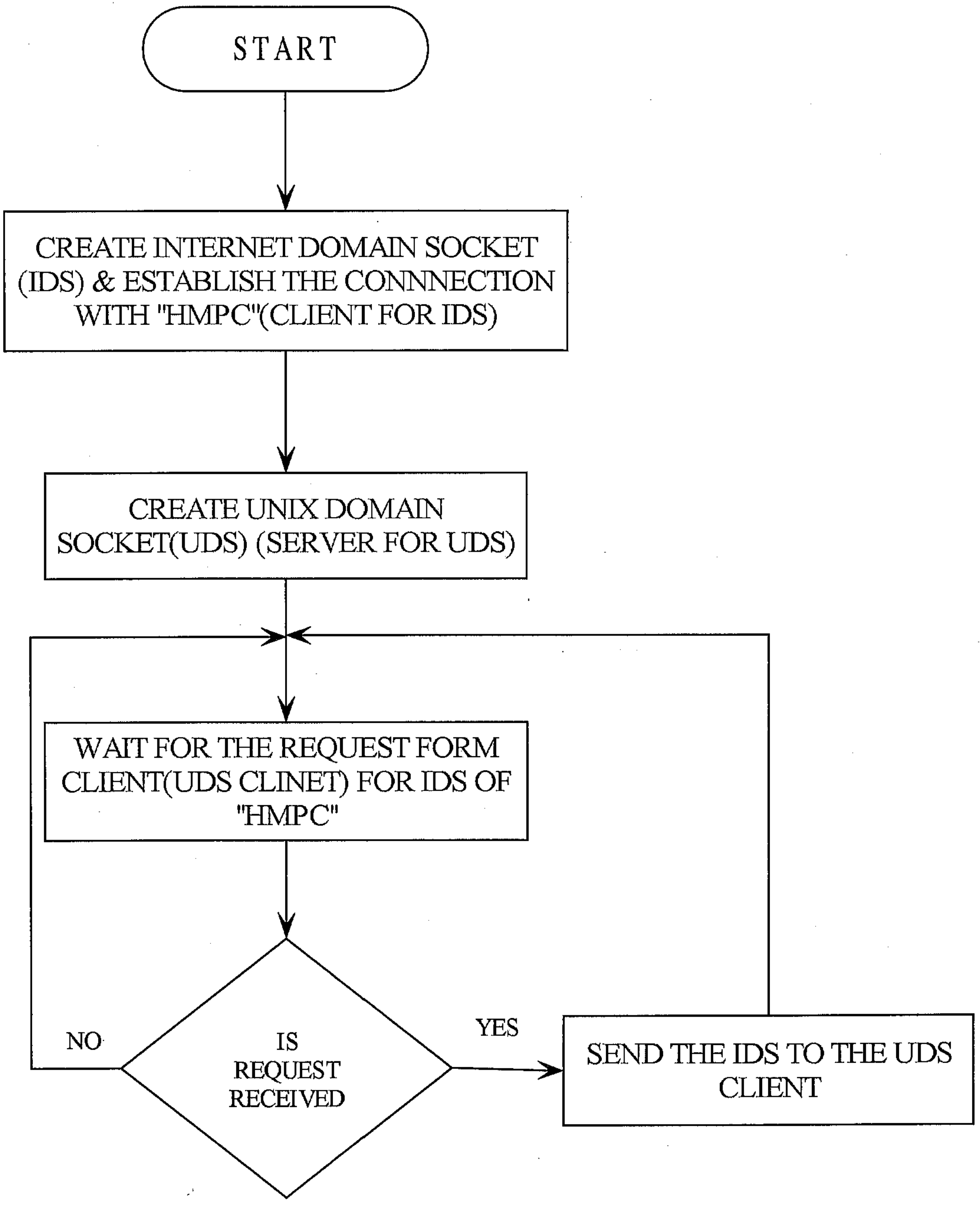

9
IS RESET BUTTON CLICKED
YES
FILL THE ENTRIES WITH THE RESET VALUES
10
NO
IS SAVE BUTTON CLICKED
YES
WAS A FILE SELECTED PREVIOUSLY
YES
OPEN THE FILE AND SAVE THE CHANNEL VALUES
NO
IS SAVEAS BUTTON CLICKED
YES
POP UP THE SAVE AS WINDOW TO GET A FILE NAME
7
NO
10

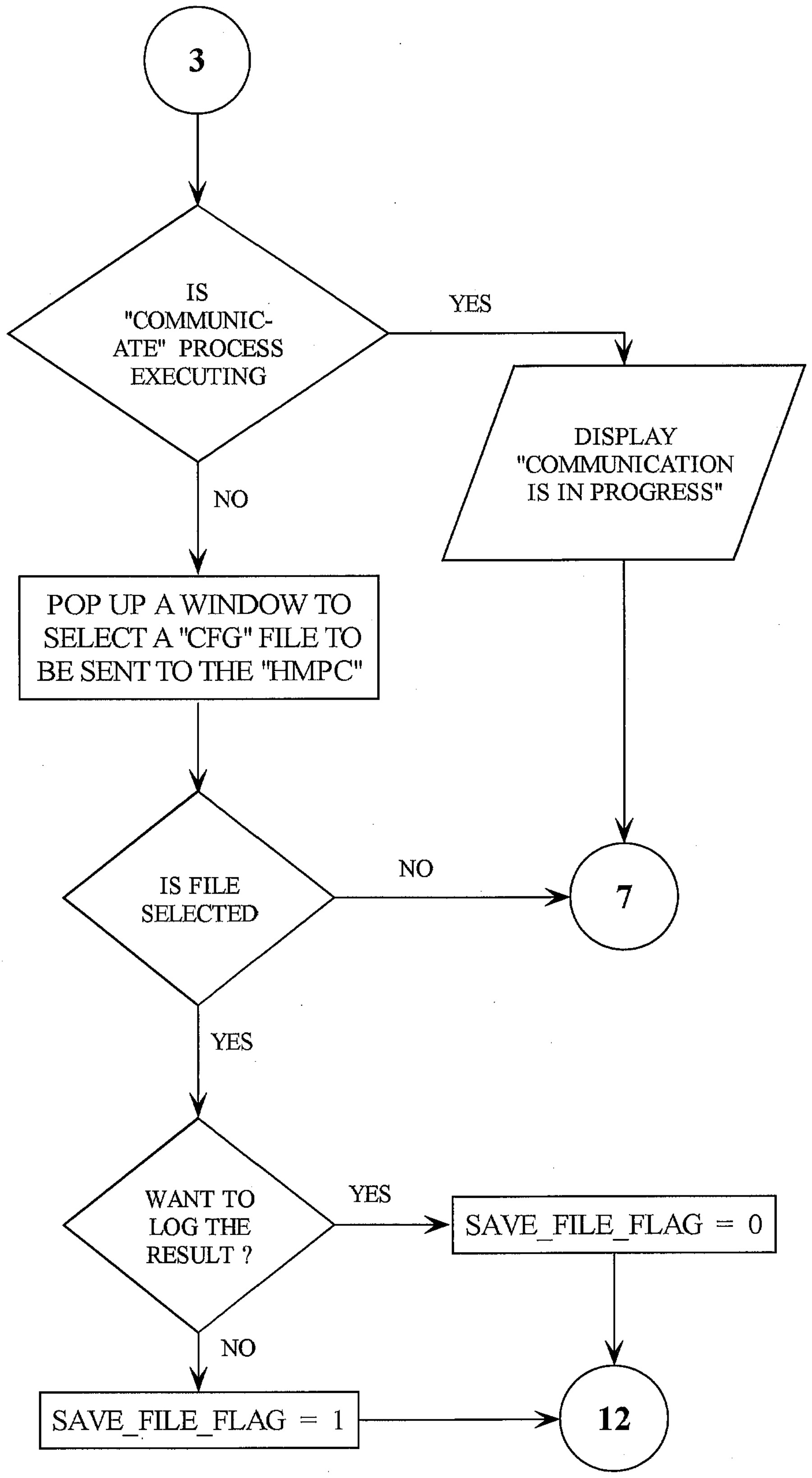

10
IS NEXT BUTTON CLICKED
YES
ARE THE LAST 8 CHANNEL VALUES DISPLAYED
YES
NO
NO
DISPLAY NEXT 8 CHANNEL VALUES
IS BACK BUTTON CLICKED
YES
NO
ARE THE FIRST 8 CHANNEL VALUES DISPLAYED
YES
7
NO
IS HELP BUTTON CLICKED
YES
DISPLAY PREVIOUS 8 CHANNEL VALUES
NO
DISPLAY PREVIOUS 8 CHANNEL VALUES
11

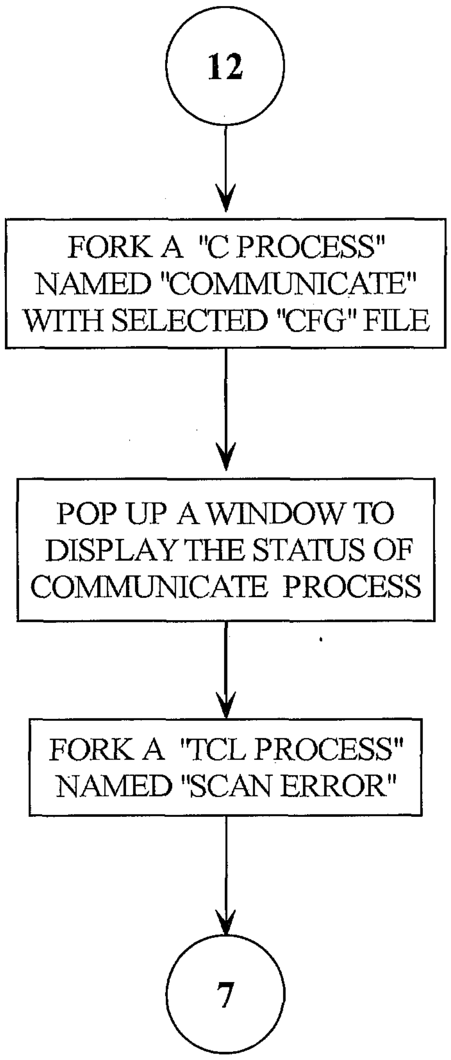

11
IS @ BUTTON CLICKED
YES
MOVE TO THE NEXT LOCATION IN THE ORDER OF CABIN, TBCC & RX-ROOM
NO
IS EXIT BUTTON CLICKED
NO
YES
CLOSE MONITORING MODULE CONFIGURATION WINDOW
7


2
IS ENTRY MADE IN "PID FILE"
YES
NO
DISPLAY "CONNECTION ALREADY ESTABLISHED"
FORK A "C PROCESS" NAMED "CONNECT_PC"
POP UP A WINDOW TO DISPLAY THE STATUS OF CONNECT_PC PROCESS
7

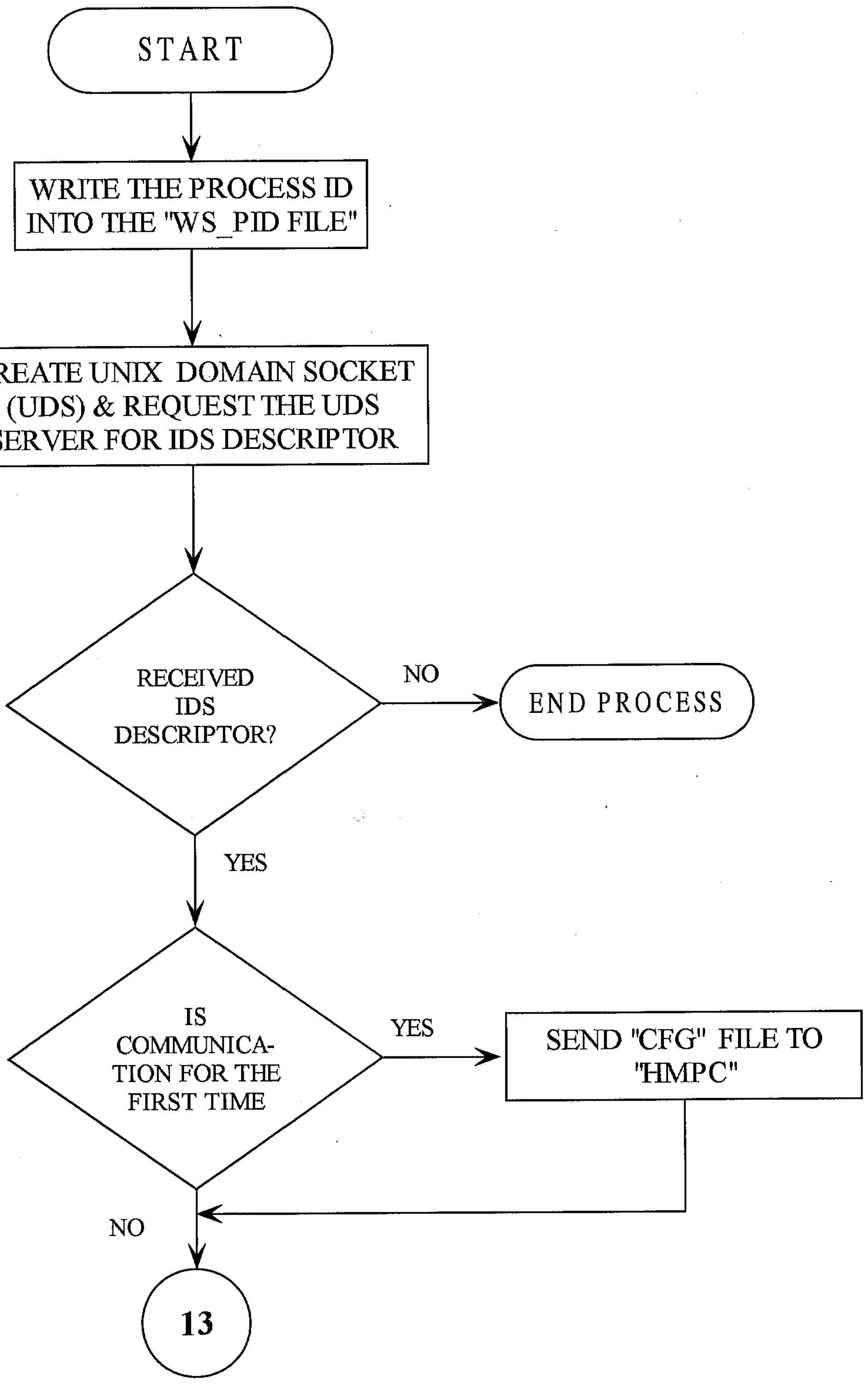
CONNECT_PC PROCESS

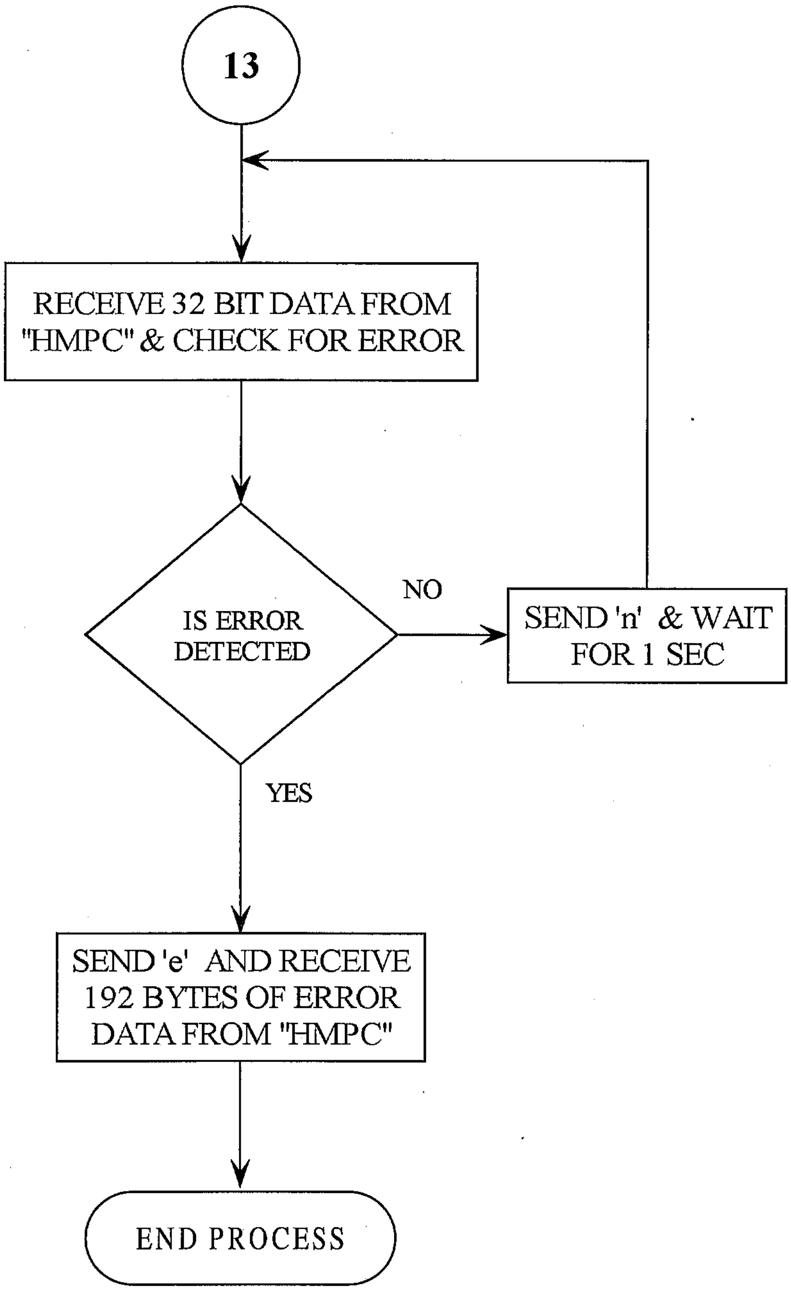

START
CREATE INTERNET DOMAIN SOCKET (IDS) & ESTABLISH THE CONNNECTION WITH "HMPC"(CLIENT FOR IDS)
CREATE UNIX DOMAIN SOCKET(UDS) (SERVER FOR UDS)
WAIT FOR THE REQUEST FORM CLIENT(UDS CLINET) FOR IDS OF "HMPC"
IS REQUEST RECEIVED
NO
YES
SEND THE IDS TO THE UDS CLIENT


3
IS "COMMUNIC-ATE" PROCESS EXECUTING
YES
DISPLAY "COMMUNICATION IS IN PROGRESS"
NO
POP UP A WINDOW TO SELECT A "CFG" FILE TO BE SENT TO THE "HMPC"
IS FILE SELECTED
NO
7
YES
WANT TO LOG THE RESULT ?
YES
SAVE_FILE_FLAG = 0
NO
SAVE_FILE_FLAG = 1
12

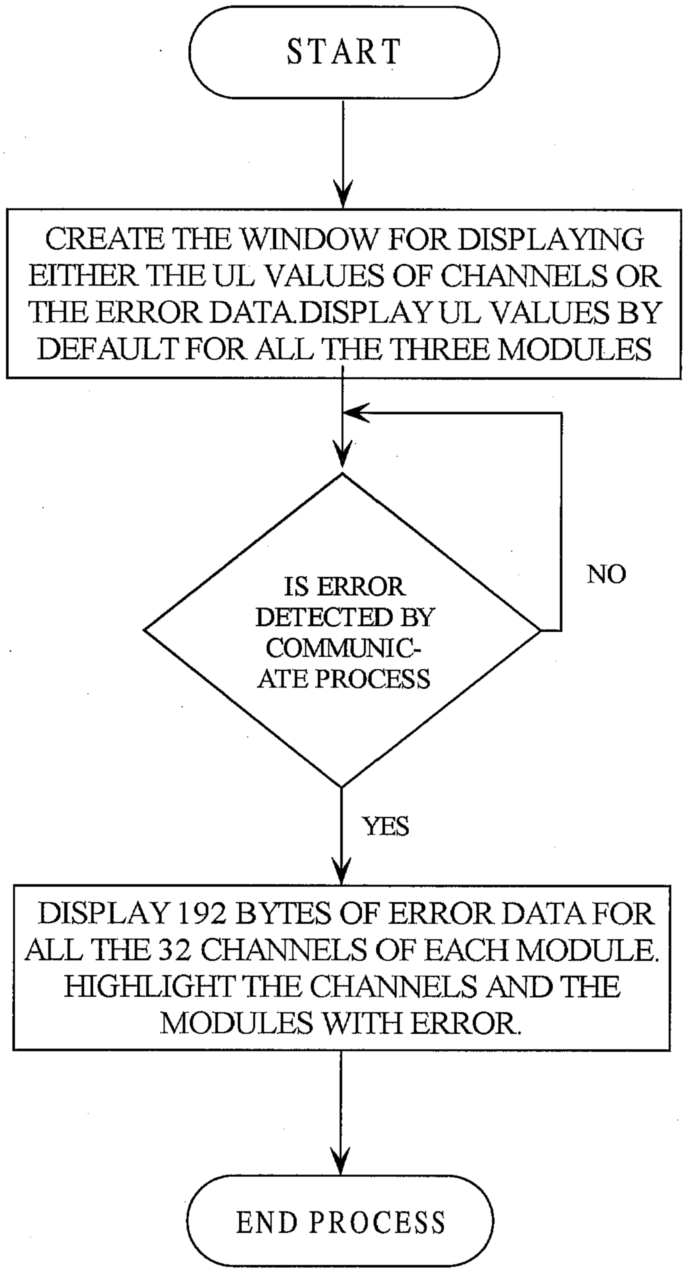

12
FORK A "C PROCESS" NAMED "COMMUNICATE" WITH SELECTED "CFG" FILE
POP UP A WINDOW TO DISPLAY THE STATUS OF COMMUNICATE PROCESS
FORK A "TCL PROCESS" NAMED "SCAN ERROR"
7

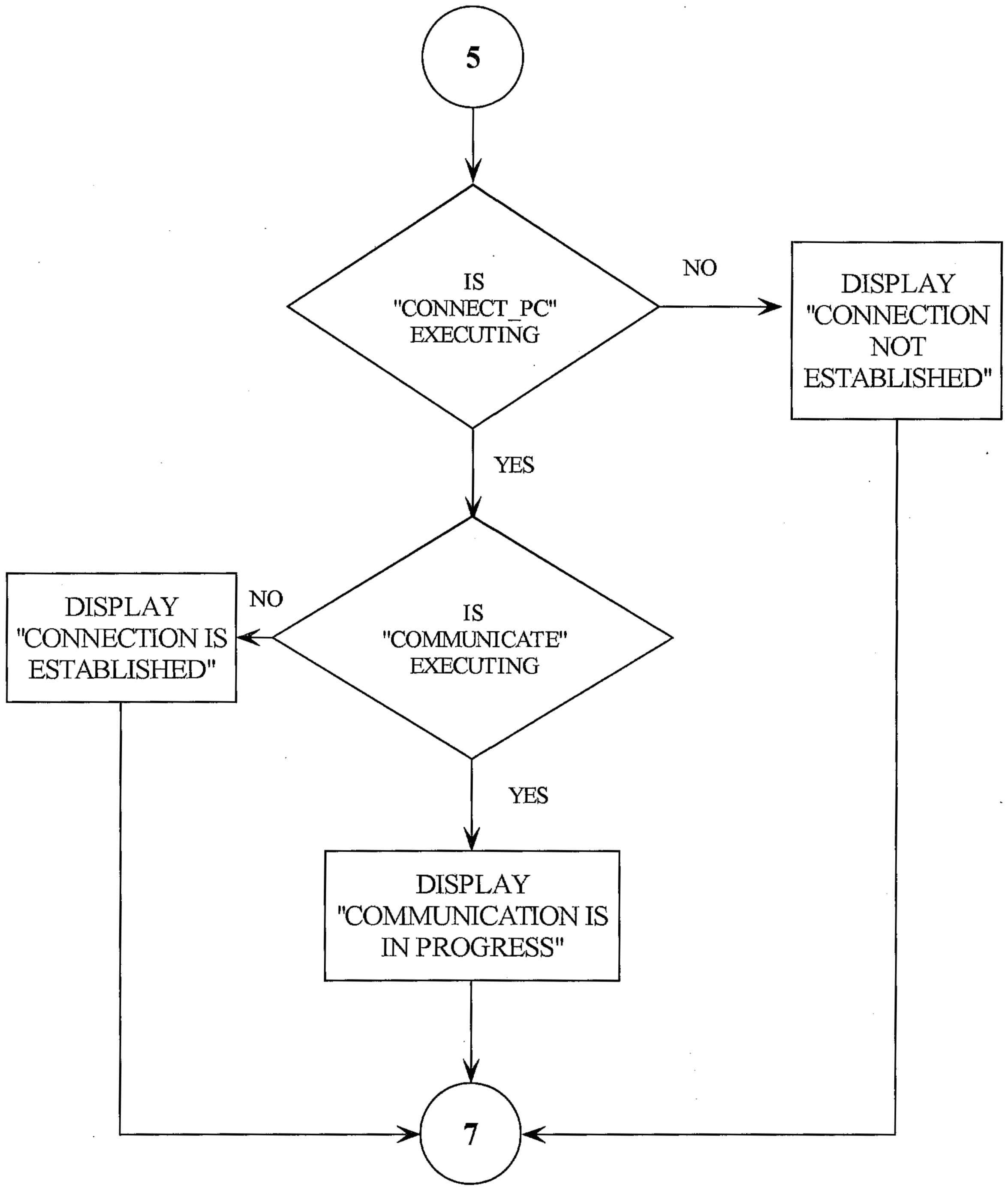
COMMUNICATE PROCESS

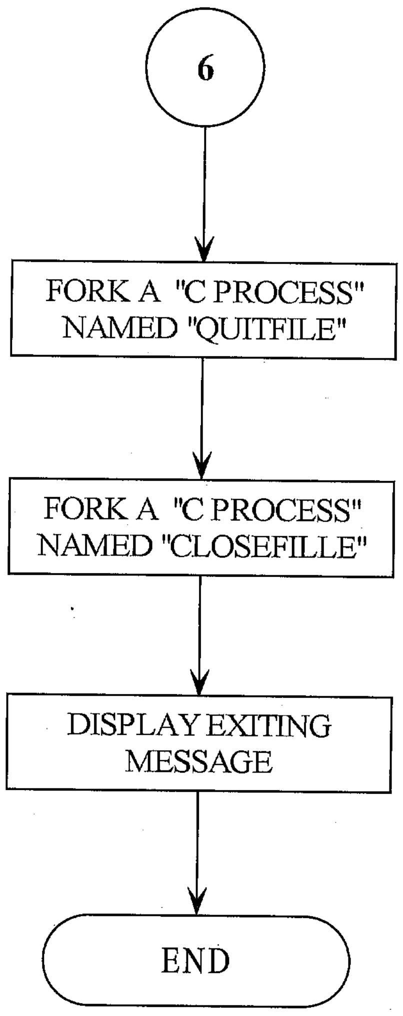

START
WRITE THE PROCESS ID INTO THE "WS_PID FILE"
CREATE UNIX DOMAIN SOCKET (UDS) & REQUEST THE UDS SERVER FOR IDS DESCRIPTOR
RECEIVED IDS DESCRIPTOR?
NO
END PROCESS
YES
IS COMMUNICA-TION FOR THE FIRST TIME
YES
SEND "CFG" FILE TO "HMPC"
NO
13


13
RECEIVE 32 BIT DATA FROM "HMPC" & CHECK FOR ERROR
IS ERROR DETECTED
NO
SEND 'n' & WAIT FOR 1 SEC
YES
SEND 'e' AND RECEIVE 192 BYTES OF ERROR DATA FROM "HMPC"
END PROCESS

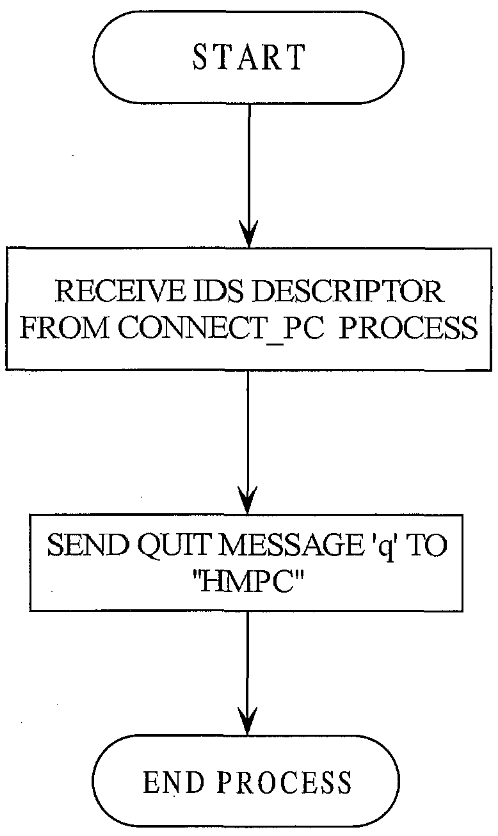
SCAN_ERROR PROCESS


START
CREATE THE WINDOW FOR DISPLAYING EITHER THE UL VALUES OF CHANNELS OR THE ERROR DATA.DISPLAY UL VALUES BY DEFAULT FOR ALL THE THREE MODULES
IS ERROR DETECTED BY COMMUNIC-ATE PROCESS
NO
YES
DISPLAY 192 BYTES OF ERROR DATA FOR ALL THE 32 CHANNELS OF EACH MODULE. HIGHLIGHT THE CHANNELS AND THE MODULES WITH ERROR.
END PROCESS

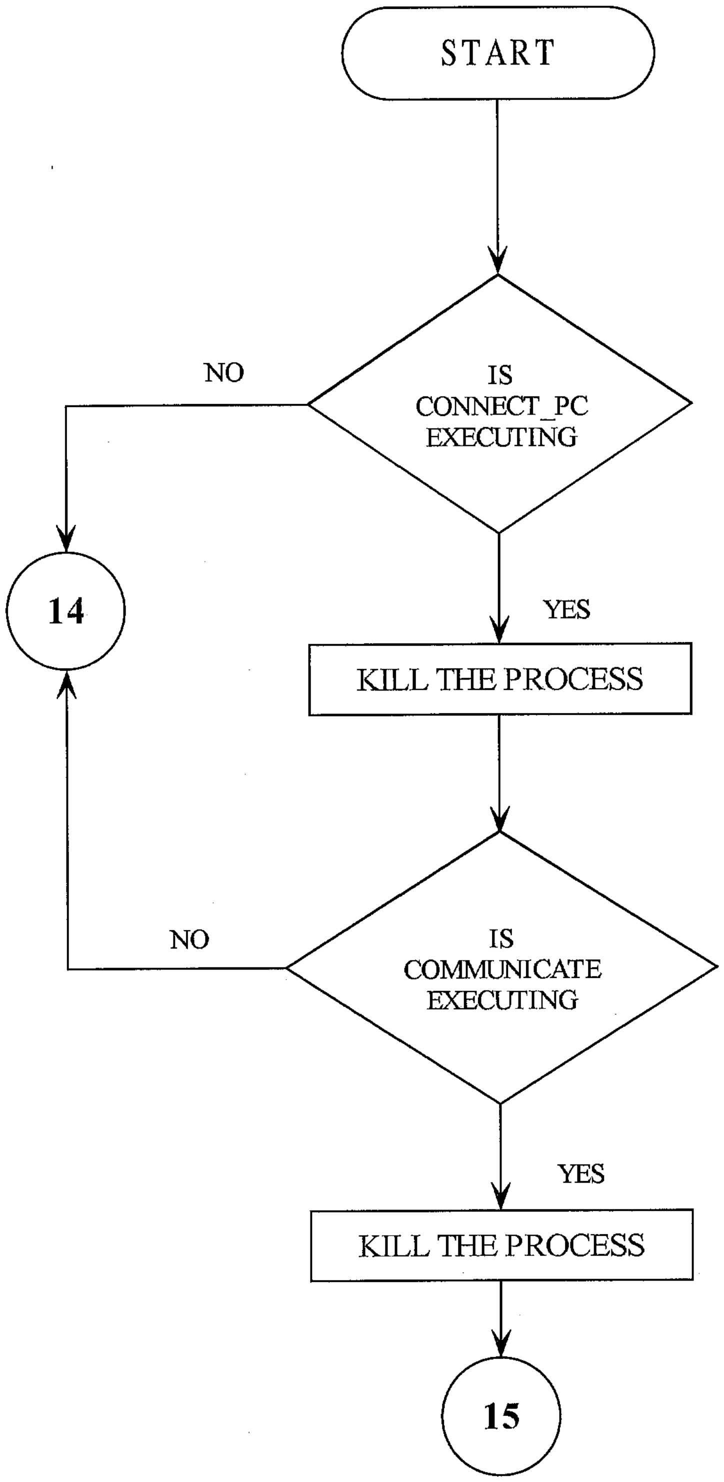

5
IS "CONNECT_PC" EXECUTING
NO
DISPLAY "CONNECTION NOT ESTABLISHED"
YES
IS "COMMUNICATE" EXECUTING
NO
DISPLAY "CONNECTION IS ESTABLISHED"
YES
DISPLAY "COMMUNICATION IS IN PROGRESS"
7

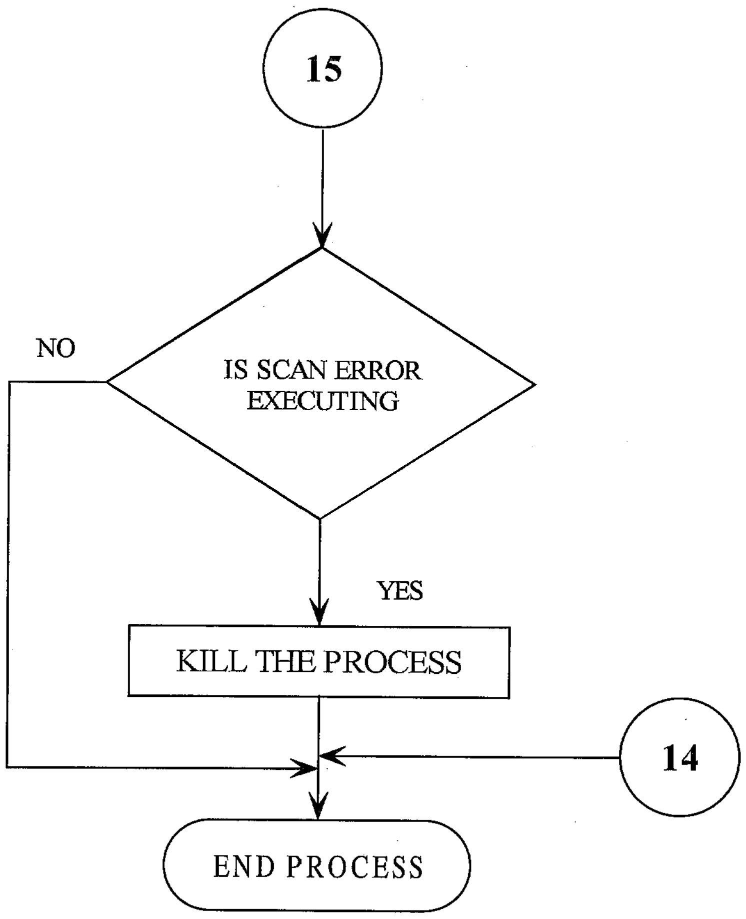

6
FORK A "C PROCESS" NAMED "QUITFILE"
FORK A "C PROCESS" NAMED "CLOSEFILLE"
DISPLAY EXITING MESSAGE
END

QUITFILE PROCESS

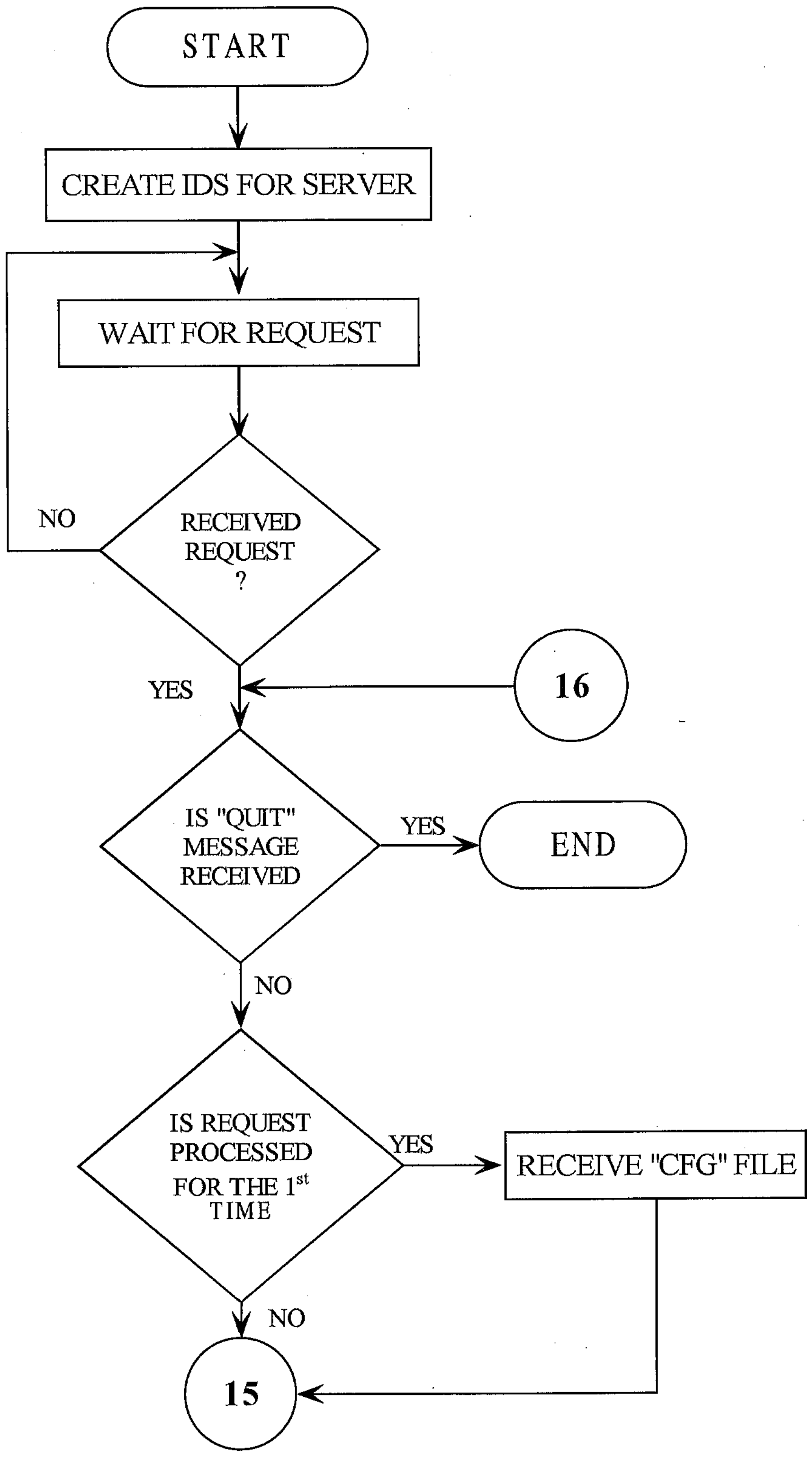

START
RECEIVE IDS DESCRIPTOR FROM CONNECT_PC PROCESS
SEND QUIT MESSAGE 'q' TO "HMPC"
END PROCESS

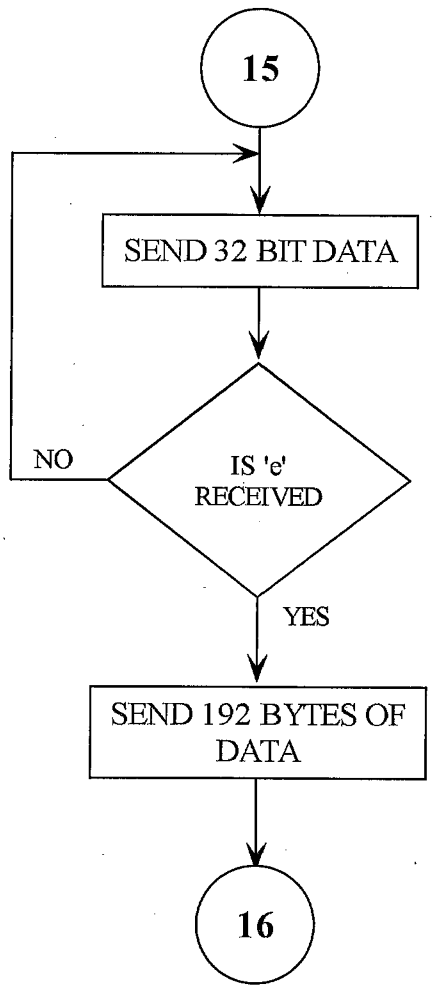
CLOSEFILE PROCESS




15
IS SCAN ERROR EXECUTING
NO
YES
KILL THE PROCESS
14
END PROCESS

MONITORING CONTROL MODULE




15
SEND 32 BIT DATA
IS 'e' RECEIVED
NO
YES
SEND 192 BYTES OF DATA
16

CHAPTER 5

TESTING

TESTING

Testing is a most important phase of the software development. Any software would be a failure if it does not handle all exceptions and carries out the actions in the manner specified in the requirement specification document.

The tests were done at the following stages
Unit Testing
Module level Testing
Integrated Testing
System level Testing

At each phase of the software development testing was carried out independently on different modules and all these modules were integrated to from a full fledged software. The module was tested with sample data . The results obtained satisfy the requirements. The software is found to be working satisfactorily.

CHAPTER 6
CONCLUSION

CONCLUSION

The Software 'A GUI FOR REMOTE CONTROL OF HEALTH MONITORING SYSTEM' carried out at "RAMAN RESEARCH INSTITUTE", mainly deals with remotely controlling the Monitoring modules through HEALTH MONITORING SYSTEM from Workstation. This is done by establishing communication between the Health Monitoring System which monitors the telescope and the Workstation which checks for error data and displays the current status of communication.

The Software has been coded using 'C' language. GUI has been designed using TCL/TK libraries that are part of 'C' subroutine libraries. The software has been implemented in a Client-Server model fashion and TCP/IP Protocol Suite has been used for communication between the Client and the Server. The Platform on which the program has been built is SUN SOLARIS which is a flavor of UNIX Operating System. The Entire Software is built on a PC and three 80186 based single board computers(SBC). A network communication module to connect to workstation. FTP Software Inc. Software Development Kit on DOS platform will be used for this module.

The Software designed has the following advantages over the existing software. They are

1. The existing Health Monitoring System receives the signals from control modules of the Telescope. There is no provision for reporting the error immediately after comparing these values with the configured values and also there was no interface for the user at the Workstation. As a result of which even in the wake of error, which is not detected, the data acquisition continues until the end of the observation . Due to this the whole of the observation had to be redone.

2. By remote controlling the monitoring modules, comparison between the received values and the configured values is made and the error is brought to the notice of the user.
3. The monitoring module and the particular channels with erroneous data is also brought to the notice of user, through which the user can do further analysis. All these are done through a Graphical User Interface which is more appealing to the user.

The above said project work has been carried out at "Telescope Building" of RAMAN RESEARCH INSTITUTE as part of an academic purpose to be submitted to Bangalore University in partial fulfillment of the requirements for the award of degree of the Bachelor of Engineering in Computer Science and Engineering.

CHAPTER 7

BIBLIOGRAPHY

BIBLIOGRAPHY

- UNIX Network Programming by W. Richard Stevens
- Internetworking with TCP/IP Volume I by Douglas E. Comer and David L. Stevens
- Internet Working with TCP/IP Volume III by Douglas E. Comer and David L. Stevens
- Computer Networks by Andrew S Tannebaum
- TCL and the TK toolkit by John K Ousterhout
- TCL/TK for programmers by Adrian J Zimmer
- Let Us C by Yashwant P kanetkar.
- PC/TCP 3.2 for DOS Software Development Kit.

APPENDICES

SYSTEM CALLS

The Unix System Calls are the direct entry points into the KERNEL. Several of these System Calls where used in our project. Some of these with a little Description are:

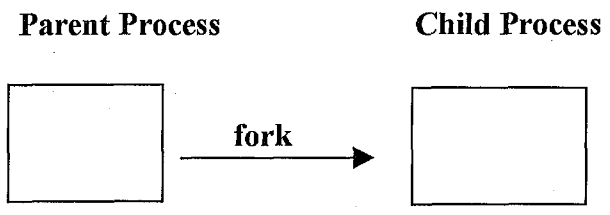
Fork

The only way in which a new process is created by unix is for an existing process to execute the fork system call.

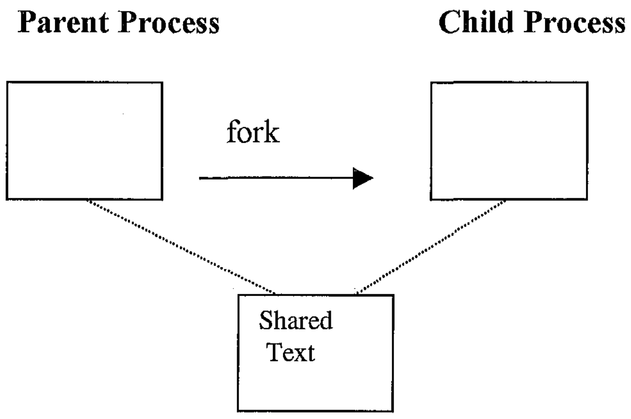
int fork();

The fork system call create a copy of the process that was executing. The process that Executed the fork is called the *parent process* and the new process is called the *child process.* The fork system call is called once (by parent process) but it returns twice (once in parent and once in the child). The only difference in the returns from the fork system call is that the return value in the parent process is the process ID of the newly created child process, while the return value in the child process is zero. If the fork system call is not successful,-1 is returned. Should the child wish to obtain the process ID of its parent process, it can call the getpid system call.

We show the fork operation as



Realize that if the text segment can be shared ,then both the parent and child can share the text segment after the fork.



Also, realise that the child's copy of the data segment is a copy of the parent's data Segment when the fork operation takes place. It is not a copy from the corresponding Program's disk file.

One important feature of the fork operation is that the files that were open in the Parent process before the fork are shared by the child process after the fork. This pass these open files to the child process. After the fork the parent closes the files that it opened for the child, so that both processes are not sharing the same file.

The values of the following in the child process are copied from the parent process.

- Real user ID
- Real group ID
- Effective user ID
- Effective group ID
- Process group ID
- Terminal group ID
- Root directory
- Current working directory

- Signal handling settings
- File mode creation mask

The child process differs from the parent process in the following ways:

- The child process has a new, unique process ID,
- The child process has a different parent process ID,
- The child process has its own copies of the parent's file descriptors,
- The time left until an alarm clock signal is set to zero in the child.

There are two uses for the fork operation

- A process wants to make a copy of itself so that one copy can handle an operation while the other copy does another task. This is typical for network servers.
- A process wants to execute another program. Since the only way to create a new process is with the fork operation, the process must first fork to make a copy of itself, then one of the copies(typically the child process) issues and exec to execute the new program. This is typical fork programs such as shells.

Signal

A signal is notification to a process that an event has occurred. Signals are sometimes called "Software Interrupts." Signals usually occur asynchronously.
By this we mean that the process doesn't know ahead of time exactly when a signal will occur. Signals can be sent

- By one process to another process (or to itself)
- By the kernel to a process

Every signal has a name, specified in the header file <signal.h>
And also every signal has its default action which can terminate the intimated process.

So the system call SIGNAL helps to handle these signal by writing our own action to the occurred signal by ***catching*** the signal.

Some of the signals that we have catched are:

Name	Description	Default action
SIGINT	Interrupt Character	Terminate
SIGTERM	Software termination signal	Terminate

A process specifies how it wants a signal handled by calling the signal system call.

#include <signal.h>

int (*signal (int sig, void (*func) (int))) (int);

what this declaration says is that signal is a function returns a pointer to a function that returns an integer. The func argument specifies the address of a function that doesn't return anything (void).

There are two special values for the func argument:

- SIG_DFL
- SIG_IGN

The signal system call always returns the previous value of func for the specified signal.

Eg: signal(SIGTERM,SIG_IGN);

<u>read</u>

Data is read form an open file using

int read(int fildes, char * buff, unsigned int nbytes);

If the read is successful, the number of bytes is returned—this can be less than the nbytes that was requested. If the end of file is encountered, zero is returned. If and error is encountered, -1 is returned.

write

Data is written to an open file using

int write (int fildes, char * buff, unsinged int nbytes);

The actual number of bytes written is returned by the system call. This is usually equal to the nbytes argument. If and error occurs , -1 is returned. Both read and write specify their nbytes argument as an unsigned integer, so that more than 32767 bytes can be read or written in one operation on a system that has 16-bit integers.

APPLICATION PROGRAM INTERFACE (API)

The API is the interface available to a programmer. The availability of an API depends on both the operating system being used, and the programming language.
The two most prevalent communication APIs for Unix systems are ***Berkeley Sockets*** and the ***System V Transport Layer Interface (TLI)***. Both of these interfaces were developed for the C language.

BERKELEY SOCKETS

The socket interface was first provided with the 4.1BSD system for the VAX, circa 1982. The interface that we describe here corresponds to the original 4.3BSD VAX release from 1986. This release supported the following communication protocols:

- Unix domain
- Internet domain (TCP/IP)
- Xerox NS domain

Socket Addresses

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 values*/
 char sa_data[14];
};
```

The contents of the 14 bytes of protocol-specific address are interpreted according to the type of address. For the Internet family, the following structures are defined in **<netinet/in.h>**

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

ELEMENTARY SOCKET SYSTEM CALLS

We now describe the elementary system calls required to perform network programming .

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).

```
#include < sys/types.h>
#include < sys/socket.h>

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

The family is one of

• **AF_UNIX**	Unix internal protocols
• **AF_INET**	Internet protocols
• **AF_NS**	Xerox NS protocols
• **AF_IMMPLINK**	IMP link layer

The AF_ prefix stands for "address family ."

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

Not all combinations of socket family and type are valid. Figure shown below shows the valid combinations, along with the actual protocol that is selected by pair.

	AF_UNIX	AF_INET	AF_NS
SOCK_STREAM	Yes	TCP	SPP
SOCK_DGRAM	Yes	UDP	IDP
SOCK_RAW	IP	Yes	SPP
SCOK_SEQPACKET			

The boxes marked "Yes" are valid, but don't have handy acronyms, The empty boxes are not implemented.

The protocol argument to the socket system call is typically set to 0 for most user application .There are specialized applications however, that specify a protocol value, to use a specific protocol.

The socket system call returns a small integer value, similar to a file descriptor.

For an association

{protocol, local-addr, local-process, foreign-addr, foreign-process }

all the socket system call specifies is one element of this 5-tuple, the protocol.

<u>Bind System Call</u>

The bind system call assigns a name to an unnamed socket.

#include < sys/types.h>

#include < sys/socket.h>

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. There are three uses of bind.

- Servers register their well-known address with the system. It tells the system "this is my address and any messages received for this address are to be given to me.". Both connection-oriented and connectionless servers need to do this before accepting client request.
- A client can register a specific address for itself
- A connectionless client needs to assure that the system assigns it some unique address,so that the other end (the server) has a valid return address to send its

responses to. This corresponds to making certain an envelope has a valid return address, if we expect to get a reply from the person we send the letter to.

The bind system call fills in the local-addr and local-process elements of the association 5-tuple.

Connect System call

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

```
#include < sys/types.h >
#include < sys/socket.h >

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

For most connection-oriented protocols (TCP and SPP, for example), the connect system call results in the actual establishment of a connection between the local system and the foreign system. The connect system call does not return until the connection is established, or an error is returned to the process.

Listen System Call

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

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

It is usually executed after both the socket and bind system calls, and immediately before the accept system call. 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. In the time that it takes a server to handle the request of and accept(the time required for the server to fork a child process and then have the parent process execute another accept) ,it is possible for additional connection requests to arrive from other clients. What the backlog argument refers to is this queue of pending request for connections.

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.

```
#include < sys/types .h>
#include < sys/socket.h>

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. If there are no connection 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 . addrlen is called a value-result argument: the caller sets its value before the system call, and the system call stores a result in the variable. Often these value-result arguments are integers that the caller sets to the size of a buffer, with the system call changing this value on the return to the actual amount of data stored in the buffer. For this system call the caller sets addrlen to the size of the sockaddr structure whose address is passed as the peer argument. On return, addrlen contains the actual number of bytes that the system call stores in the peer argument.

This system call returns up to 3 values:

- An integer return code that is either and error indication or a new socket descriptor
- The address of the client process(peer)
- The size of this address (addrlen)

Accept system call automatically creates a new socket descriptor, assuming the server is a concurrent server.

All five elements of the 5-tuple associated with newsocketid have been filled in on return from accept.

Send msg ,Recvmsg System Calls

These system calls are the most general of all the read and write system calls.

```
#include <sys/types.h>
#include  <sys/socket.h>

int sendmsg(int sockfd, struct msghdr msg[], int flags);

int recvmsg(int sockfd, struct msghdr msg[], int flags);
```

These use the msghdr structure that is defined <sys/socket.h>

```
struct msghdr {
        caddr_t         msg_name; /*optional address*/
        int             msg_namelen;  /*size of address*/
        struct iovec    msg_iov;        /*scatter/gather array*/
        int             msg_iovlen;   /* # elements in msg_iov*/
        caddr_t                 msg_accrights; /*access rights sent/recvd*/
        int             msg_accrightsleln;
};
```

The msg_name and msg_namelen fields are used when the socket is not connected. The msg_name field can be specified as a NULL pointer if a name is either not required or not desired. The msg_iov and msg_iovlen fields are used for scatter read and gather write operations. The final two elements of the structure, msg_accrights and msg_accrightslen deal with the passing and receiving of access rights between processes.

Byte Ordering Routines

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

```
#include < sys/types.h>
#include < netinet/in.h>

u_long htonl(u_long hostlong);
u_short htons(u_short hostshort);
u_long ntohl(u_long netlong);
u_short ntohs(u_short netshort);
```

These functions were designed for the Internet protocols. Fortunately, the XNS protocols use the same byte ordering as the Internet. On those systems that have the same byte ordering as the Internet protocols (Motorola 68000-based systems, for example), these four functions are null macros. The conversions done by these functions are shown.

Byte ordering functions

Htonl	convert host-to-network,long integer
Htons integer	convert host-to-network,short
Ntohl	convert network-to-host,long integer
Ntohs integer	convert network-to-host,short

Byte Operations

There are multibyte fields in the various socket address structures that need to be manipulated. Some of these fields, however, are not C integer fields, so some other technique must be used to operate on them portably

4.3BSD defines the following that operate on user-defined byte strings. This can have null bytes within them and these do not signify the end of the string. Instead, we must specify the length of each string as an argument to the function

bzero(char *dst,int nbytes);

Bzero function writes the specified number of null bytes to the specified destination.

FTP SYSTEM CALLS

Net-connect() System Call:

Initializes a network descriptor with protocol type and endpoint information. For a STREAM type network descriptor, it also performs an active open request to establish a connection.

Include files:

<pctcp/types.h>
<pctcp/error.h>
<pctcp/pctcp.h>
<pctcp/interfac.h>

Call

int net_connect(int nd, int type, struct addr *addr);

Inputs

addr	structure specifying foreign and local endpoints or network interface information(DS:SI)
nd	network descriptor net_connect() initializes for connectionless mode transport or uses for an active open of a STREAM connection(BX).
type	Type of protocol service to associate with nd(DX).

Return value

On success, net_connect() returns a network descriptor(AX).

On failure, net_connect() returns –1 and puts an error condition in neterrno.

Net_listen()System Call:

Initializes a network descriptor with protocol type and endpoint information. For a STREAM type network descriptor, it also performs a passive open.

Include files:

<pctcp/types.h>

<pctcp/error.h>

<pctcp/pctcp.h>

Call

int net_listen(int nd,int type,struct addr *addr);

Inputs

addr	structure specifying the internet,address foreign and local sockets.
nd	network descriptor with which net_listen() passively accepts connectionless services. If nd is –1,a network descriptor will be allocated and returned by net_listen()(BX).
type	Type of service(DGRAM, RAW_IP,RAW_NET,or STREAM) to associate with nd(DX).

Return value

On success, net_listen() returns a network descriptor(AX).

On failure, net_listen() returns –1 and puts an error condition in neterrno.

Net-getdesc() System Call:

Allocates a local network descriptor.

Include files:

<pctcp/error.h>

<pctcp/pctcp.h>

Call

int net_getdesc();

Inputs

None

Return Value

On success, net_getdesc() returns a local network descriptor(AX).

On failure, net_getdesc() returns –1 and puts an error condition in neterrno.

Net_read()System Call:

Reads into a buffer data received on a network descriptor (AH=0x1b)

Include files:

<pctcp/types.h>

<pctcp/error.h>

<pctcp/pctcp.h>

<pctcp/options.h>

Call

int net_read(int nd, char *buf, unsigned len, struct addr *from, unsigned flags);

Inputs

buf	pointer to the buffer containing incoming data(DS:SI).
nd	network descriptor (BX).
len	no of bytes to read into buf(CX)
from	Pointer to an addr structure containing address info extracted from datagram header(ES:DI).
flags	Boolean options for the receive operation.

Return value

On success, net_read() returns no of bytes read(AX).

On failure, net_read() returns –1 and puts an error condition in neterrno.

Net_write()System Call:

Transmits data over a network descriptor(AH=0x1a)

Include files:

<pctcp/error.h>

<pctcp/pctcp.h>

Call

int net_write(int nd, char *buf, unsigned len, unsigned flags);

Inputs

buf	pointer to the buffer containing data to transmit(DS:SI).
nd	network descriptor (BX).
len	no of bytes to transmit, 0 is also valid(CX)
flags	Boolean options to apply to the transmission

Return value

On success, net_write() returns no of bytes written(AX).

On failure, net_write() returns –1 and puts an error condition in neterrno.

TCL/TK

Widget: A control is called a widget.

Top level widget:

TK(Tool kit) has a special widget, called a Top level widget to automatically create widgets which are independent of main window.

Window Manager:

The window manager controls the look and feel of the desk top and therefore controls the size, placement and visibility of the top level windows. The window manager also adds the title bar and the decorative border that surrounds the top level window. The main application window "." is the leader.

TK has *wm* command for sending request to window manager.

Eg:
```
wm title .notice "application: Notice"
wm resizable .notice 0 0
```

Button:

To create a button :

button .b –text "Our Project" ;creates a button called .b with the caption Our Project, it accepts various events like "command", "active background" etc.

Entry:

A textbox which allows to enter text holding it in a text variable declared during entry definition.

entry .e –textvariable x; x is the variable which holds the text.

Label:

To create a label, same as Entry except that it is disabled.

Label .l –textvariable x; x is the variable which holds the text.

ScrollBar:

Scrollbars control the views in other widgets. Each Scrollbar widget is associated with some other widget such as a list box or an entry. The scrollbar displays an arrow at each end and a rectangular slider in the middle. The size and position of the slider indicate which of the list boxes entries are currently visible in its window.

A scrollbar interacts with its associated widgets using TCL scripts. Yscroll is used to scroll in vertical direction and Xscroll is used to scroll in horizontal direction

Messagebox

```
tk_messageBox –icon info –message "Project"
```

The messagebox is used to display a message with the specific buttons like "ok"; while it appears on the screen the other windows in the application are inactive. When the user dismisses the dialog, the applications responsive again, and the call to tk_messageBox returns.

Pack:

The widgets used will be hidden off-screen until they are positioned with the help of a geometry manager. This is done using the pack command is used. Each pack command positions a widget within its parent window. The entire parent window is treated as an empty cavity. Widgets are placed around the edges of the cavity – the top , bottom, left and right sides – according to the order of the pack statements.

```
Eg: button .b –text "hello"
    label  .l –text "how are you"
    pack .b .l –side top
```

Bind:

The bind command is used to create, modify, query, and remove bindings. A binding causes a particular TCL Script to be evaluated whenever a particular event occurs in a particular window. Bindings are created with commands like this one:

bind .entry <control-d> { .entry delete insert}

The first argument to the command specifies the path name of the window to which the binding applies. The second argument specifies a sequence of one or more X events. The third argument may be any TCL script.

The bind command is also used to retrieve information about bindings.

Events:

An event is a record generated by X to notify an application of a potentially interesting occurrence. Each event has a *type* that indicates the general sort of thing that occur. The event types that are most commonly used in bindings are those for user actions such as

Key or KeyPress :	A key was pressed.
KeyRelease:	A key was released.
Button or ButtonPress:	A Mouse button was pressed.
ButtonRelease:	A mouse button was released.
Enter:	The pointer moved into a window
Leave:	The pointer moved out of window.
Motion:	The pointer moved from one point to another with in a single window.

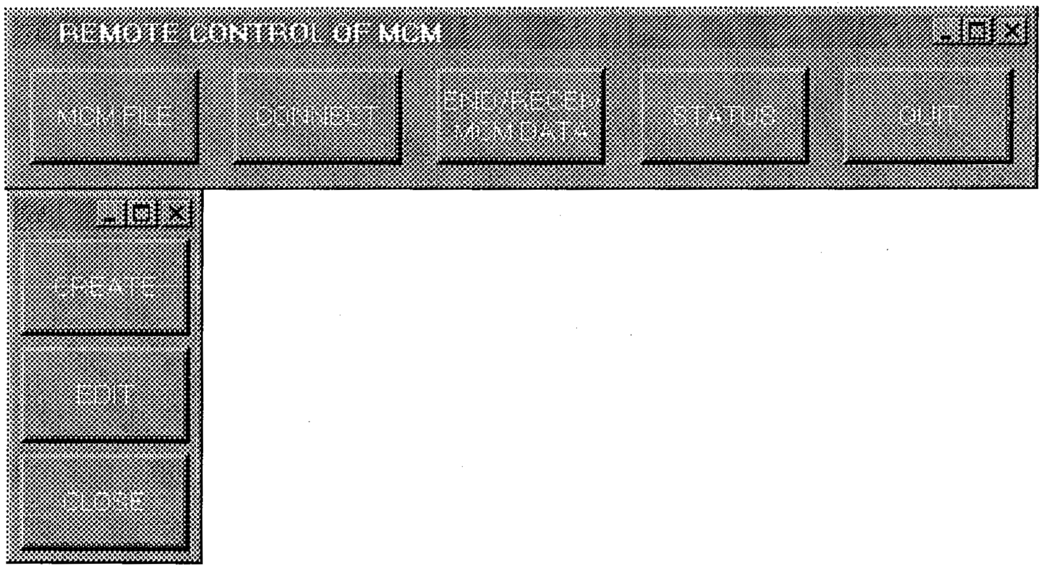
Source:

The source command reads a file and executes the contents of the file as a Tcl Script. Source takes a single argument that specifies the name of the file.

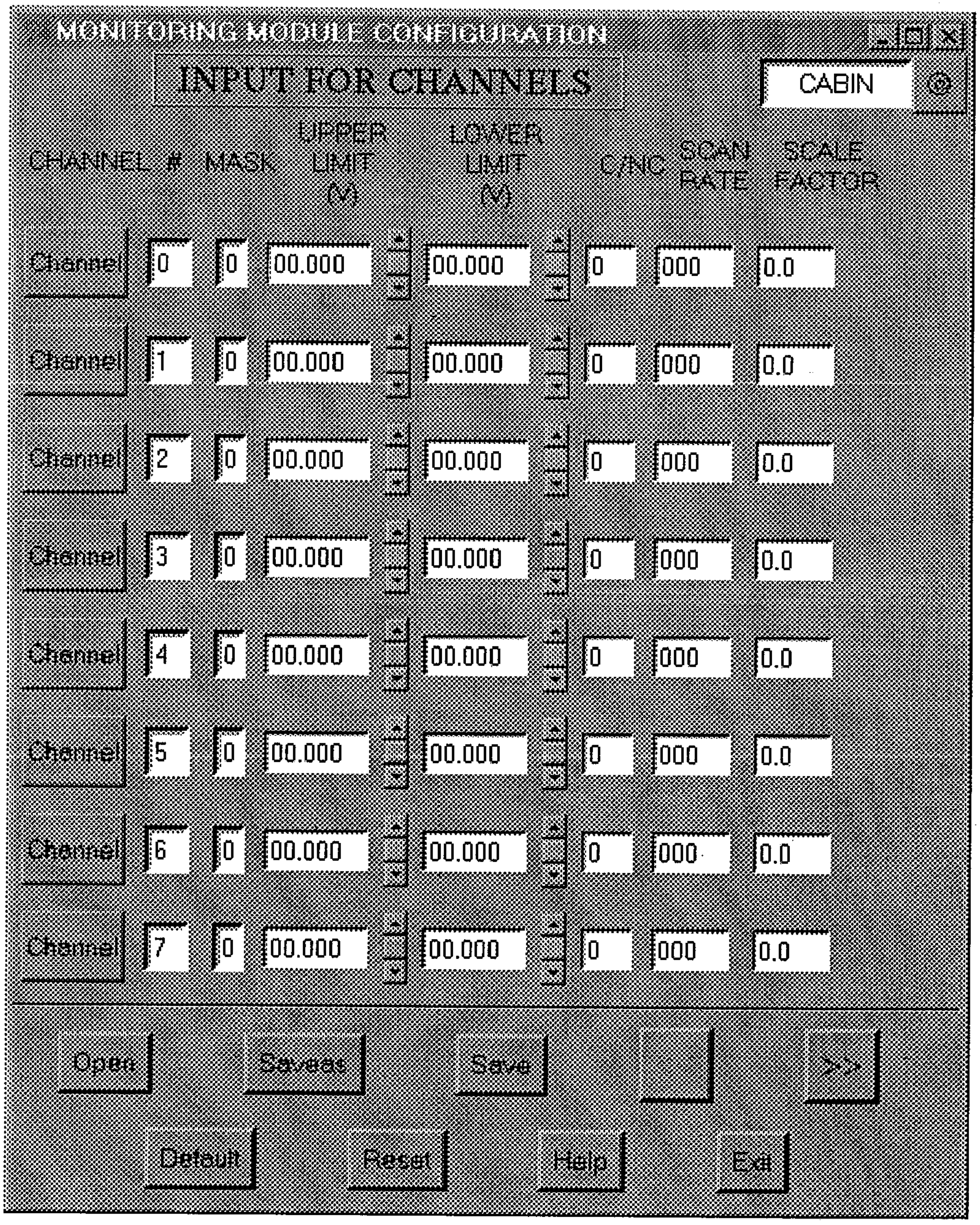
Eg: source init.tcl

will execute the contents of the file init.tcl. The return value from source will be the value returned when the file contents are executed, which is the return value from the last command in the file. In addition source allows the return command to be used in the file script to terminate the processing of the file.

MAIN MENU

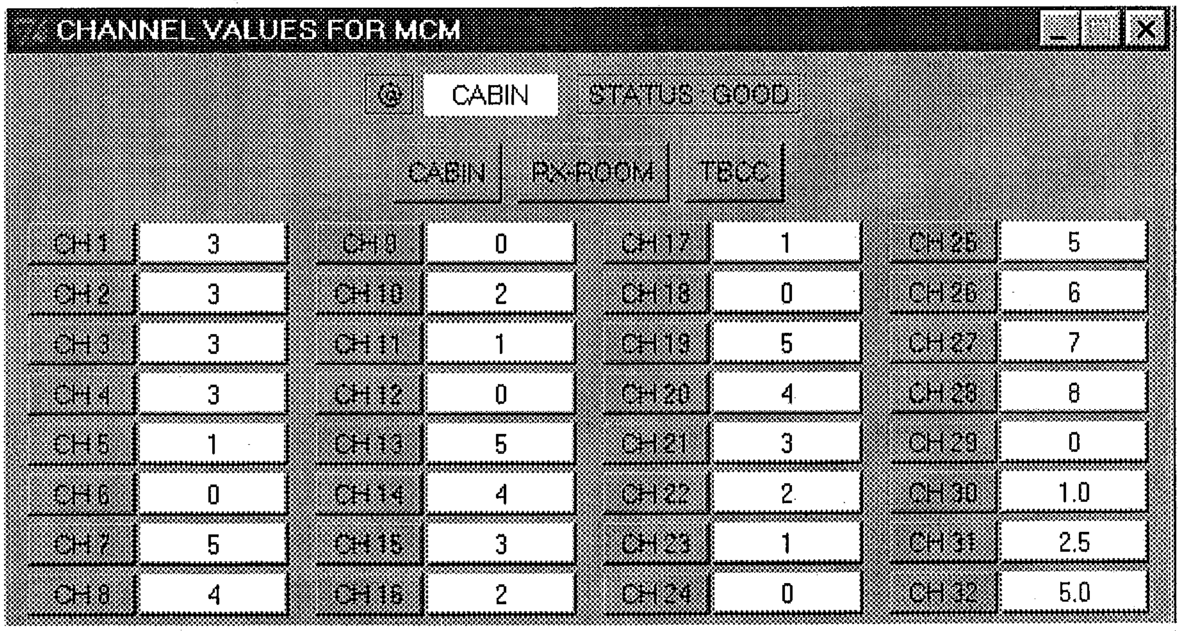

REMOTE CONTROL OF MCM
MCM FILE
CONNECT
SEND/RECEIVE MCM DATA
STATUS
QUIT
CREATE
EDIT
CLOSE

FILE CONFIGURATION SETUP MENU



CHANNEL	#	MASK	UPPER LIMIT (V)	LOWER LIMIT (V)	C/NC	SCAN RATE	SCALE FACTOR
Channel	0	0	00.000	00.000	0	000	0.0
Channel	1	0	00.000	00.000	0	000	0.0
Channel	2	0	00.000	00.000	0	000	0.0
Channel	3	0	00.000	00.000	0	000	0.0
Channel	4	0	00.000	00.000	0	000	0.0
Channel	5	0	00.000	00.000	0	000	0.0
Channel	6	0	00.000	00.000	0	000	0.0
Channel	7	0	00.000	00.000	0	000	0.0

CHANNEL VALUES DISPLAY MENU



CH 1	3	CH 9	0	CH 17	1	CH 25	5
CH 2	3	CH 10	2	CH 18	0	CH 26	6
CH 3	3	CH 11	1	CH 19	5	CH 27	7
CH 4	3	CH 12	0	CH 20	4	CH 28	8
CH 5	1	CH 13	5	CH 21	3	CH 29	0
CH 6	0	CH 14	4	CH 22	2	CH 30	1.0
CH 7	5	CH 15	3	CH 23	1	CH 31	2.5
CH 8	4	CH 16	2	CH 24	0	CH 32	5.0