1449

ENHANCEMENT OF FTP FOR INCREASED RELIABILITY

Dissertation Submitted to
JAWAHARLAL NEHRU UNIVERSITY
in partial fulfilment of requirements
for the award of the degree of
Master of Technology
in
Computer Science & Technology

by

BHAWANI NANDAN PRASAD

103p+ fig

Jawaharlal Nehru University

SCHOOL OF COMPUTER & SYSTEMS SCIENCES
JAWAHARLAL NEHRU UNIVERSITY
NEW DELHI - 110 067
January 1997

CERTIFICATE

This is to certify that the dissertation entitled " ENHANCEMENT OF FTP FOR INCREASED RELIABILITY" being submitted by 'BHAWANI NANDAN PRASAD' to School of Computer and System Sciences, Jawaharlal Nehru University, New delhi, in partial fulfilment of the degree of Master of Technology in Computer Science, is a bonafide work carried by him under the guidance and supervision of Prof. R.C.Phoha. This work has not been submitted elsewhere for any other purpose.

[illegible]	[illegible]
Dean, SC&SS	Supervisor
(Prof. G.V.Singh)	(Prof. R.C.Phoha)
SC&SS, J.N.U.,	SC&SS, J.N.U.,
New Delhi-110067.	New Delhi-110067.

ACKNOWLEDGEMENT

I express my profound sense of gratitude to Prof. R.C. Phoha, SC&SS, JNU, under whose invaluable guidance and incessant encouragement, my work has taken its present shape.

I would like to express my sincere thanks to the staff members of the Computer Laboratory, SC&SS, for providing me with all the facilities required during the Project.

Finally, I would like to thank my friends for the help and company they gave me on the long nights spent in the laboratory for the successfully completion of this Project.

BNPrasad

(BHAWANI NANDAN PRASAD)

CONTENTS

FIGURES WITHIN THIS THESIS :

CHAPTER .1

INTRODUCTION

The major goals of computer networking are resource-sharing, to provide high reliability by having alternate source of supply, saving money, access to remote programs, access to remote databases and value-added communication facility to provide a powerful communication medium among widely separated people. But, the main motivation for constructing computer networks is data exchange among machines. For this purpose, a lot of file accessing protocols have been developed, like FTP, TFTP, UUCP etc. Each of these protocols are not suitable. For example, when a client is interested in only a small part of a large file then transferring the whole file will require more memory for storage and will cause network congestion. Existing file transfer protocols do not support partial file transfer facility.

For a variety of reasons, file transfer in the internet has been implemented as interactive or "Forground " service. That is, a user runs the appropriate local FTP user interactive command and requests a file should fail to complete for any reason, the user must reissue the transfer request. Forground file transfer was suited to the requirements during the early days of networking when the INTERNET / ARPANET was lightly loaded and reasonably reliable.

More recently, the internet has become increasingly subject to congestion and long delays, particularly during the times of peak usage. In addition, as moreof the world becomes interconnected, planned and unplanned outages of hosts, gateways and networks sometimes make it difficult for users to successfully transfer files in FORGROUND. Performing file transfer asynchronously(i.e. in BACKGROUND) provides a solution to some of these problems by eliminating the requirement for a human user to be directly involved at the time that a file transfer takes place. Background file transfer has a number of potential advantage for a user.

*No waiting :

The user can request a large transfer and ignore it until a notification messages arrives through some common channel (e.g. electronic mail).

* End-to-End Reliability :

The FTC daemon can try transfer repeatedly until it either succeeds or fails permanently. This provides reliable end-to-end delivery of a file. In spite ofthe source or destination host being down or poor Internet connectivity during some time period.

* Multiple file delivery

* Deferred Delivery :

The user may wish to defer a large transfer until an off-break period . This may become important when parts of the Internet adopt accounting and traffic-based cost recovery mechanisms.

Also, when large files are transferred over long distances, the reliability of the link becomes problem. Actually, it is very difficult to have a link reserved for long tme over far distances.

OBJECTIVE :

Objective of FTP are :

1. To promote sharing of files (computer programs and/or data).

2. To encourage indirect or implicit (via program) use of remote computers.
3. To shield a user from variations in file storage systems among hosts, and
4. To transfer data reliably and efficiently.

The following features of available " FTP under UNIX operating system " will be enhanced to improved its reliability and efficiency :

1. One can connect to one remote machine (remote login facility)
   ----- Enhance to connect more than one machine at a time .

Problem specification :

A user can OPEN two machines at a time by giving "open "command one by one for each machine. Also, a user is working on one machine (i.e. unix server) but temporarily he wants to work on another machine then he can change the machine without doing "logout" from that machine.

2. No Background transfer available :
   ------ Provide background transfer.

Problem specification :

Background transfer includes two features :

(a). Suppose you are working on one unix machines (say jnu1), you can transfer files/data from second unix machine (say jnu2) to third unix machine (say jnu3).

(b). Suppose file transfer from source machine to destination machine is in process, you can also see the file directory on both the machines.

For this task, two connections will be required :

(1). for file transfer when userlogs into the remote site. In this session, usercan executes the usual FTP commands like ls,cd,put,get,etc.

(2). User interface (to execute ls,dir, etc)

----- Message queue implementation required.

3. Low reliability in transfer of large files to a long distance on INTERNET / ETHERNET

------- Break the large size file in N parts (say 5 parts), transfer the parted sub-files parrallel (i.e. concurrently) in different virtual paths (i.e. chunks)through different ports to destination machine and then assemble the transferred parts inqueue at different recognisable address.

Problem specification :

when large files are transferred over long distances the reliability of the link becomes a problem . Actually, it is very difficult to have a link reserved for long time over large distance. This project provides a simple solution to this problem for only class of users. This project mainly aims at users who have proper shell account at machine (which I assume to be situated at a far away site).

The solution which has been tried is to transfer smaller files over long distances instead of large ones. So, large files are split into smaller chunks and then transferred. Also, all these files-chunks are transferred concurrentlythen the total time taken to transfer the main large file will be reduced and the link is required to be reserved for smaller time .Hence, reliability of FTP for large file transfer will be increased.

In this case,the large (i.e. big) file is breaked in 5 parts of almost equal size. The 5 virtual paths (i.e. channels) will be made with N ports and bandwidth distribution utilisation .

The virtual channels have advantage of capturing more bandwidth (i.e. bandwidth hogging) on internet to facilitate the efficient transfer of files from this terminal among all other users.

The parted files of the original big file is to be transferred from different ports by virtual paths at a time. Then, finally the data packets transferred from each path is to be received separately and joined in a quue to original transeferred file. Hence, the transfer time will be reduced almost 1/n times for large size file. But, for small size file at small distance (say within department) its reliability may not increase, it may take more time.

This feature of FTP will be applicable to all the files on ETHERNET/INTERNET, but its reliability of transfer time reduction will be significant only in the case of transfer of large file from far distance in the INTERNET.

CHAPTER . 2

OVERVIEW OF FTP AND RELATED PROTOCOL

The network systems provide computers with the ability to access files on remote machines. Designers have explored a variety of approaches to remote access . Broad parameters for appraisal in this area are cost,speed, reliability in transfer and possible modes of transfer etc. We take a cursory look at various file transfer facility existing today and brief discussion of their main distinguishing features . The terms defined in this section are only those that have special significance in FTP. Some of the terminology is very specific to the FTP model.

2.1. OVERVIEW OF FTP :

FTP is the internet standard for the file transfer and is the most widely used protocol . It facilitates file transfer between two machines(which may be hetrogenous) on a network . FTP permits authorized users to log into a remote system, identify themselves, list directories ,send and receive files. Although Ftp is often used to transfer files interactively, it is actually designed to be used by programs . It can transfer simple text files or executable binaries.

FTP has the following main features :

(1). INTERACTIVE ACCESS :

Although FTP is designed to be used by programs, most implementations provide an interactive interface that allows humans to easily interact with remote servers. For examples, a user can ask for a listing of all files in a directory on a remote machines. Also, the client usually responds to input like "help" by showing the user information about commands that can be invoked.

(2). FORMAT (representation) specification :

FTP allows the client to specify the type and format of stored data. For examplethe user can specify whether a file contains text or binary integer and whether text files use the ASCII or EBCDIC character set.

(3). AUTHENTICATION CONTROL :

FTP requires clients to authorize themselves by sending a login name and password to the server before requesting file transfers . The server refuses access to clients that can not supply a valid login and password.

Like other servers,most FTP implementations allow concurrent access by multiple clients. clients use TCP to connect to the server which is reliable Transport Layer Protocol. Hence, a single master server process awaits connections and creates a slave process to handle each connections. Unlike most servers, however, the slave accepts and handles the control connection from the client, but uses an additional process or processes to handle a separate data transfer operations.

FTP also supports some housekeeping operation as making directories and changing the directory at the remote host as well as on the local host. There are some optional fuctions that are not so generally available . One of these is third party in which a user at one host causes a file to be copied between two other remote hosts. Another service is restart recovery, the ability to restart a failed file transfer where it is left off.

2.1.1. HISTORY :

FTP has had a long computation over the years. The first proposed file transfer mechanism in 1971 that were developed for implementation on hosts at M.I.T. (RFC114) plus comments and discussions in RFC - 141 . RFC- 172 provided a user-leveloriented protocol for file transfer between host computers. A revision of this as RFC- 265, restated FTP for additional review, while RFC- 281 suggested furtherchanges. The use of a " set data type " transaction was proposed in RFC- 294 in jan 1982.

RFC- 354 obsoleted RFCs 264 and 265. The file transfer protocol was now defined as a protocol for file transfer between hosts on the ARPANET, with the primary fuction of FTP defined as transferring files efficientely and reliably among hosts and allowing the convenient use of remote file storage capabilities.

RFC- 385 further commented on errors, emphasis points additions to the protocol,while RFC- 414 provided a status report on the working server and user FTPs. RFC430, issued in 1973, presented further comments on FTP. Finally, an "official" FTP document was published as RFC- 454. By july 1973, considerable changes from the last versions of FTP were made but the general structure remained the same.

RFC- 542 was published as a new "official" specification to reflect these changes. In 1974, RFCs 607 and 614 continued comments on FTP. RFC- 624 proposed furtherdesign changes and minor modifications. In 1975,RFC- 686 entitled, " Leaving well enough alone " discussed the differences between all of the early and later versions of FTP. RFC- 691 presented a minor revision of RFC- 686, regarding the subject of print files.

Motivated by the transition from NCP (network control Protocol) to TCP(Transmissiom Control Protocol) at the underlying protocol, a phoenix was born out of all of the above efforts in RFC- 765 as the specification of FTP for use on TCP. The current edition of FTP specification (RFC- 959) is intended to correct some minor documentation errors, to improve the explanation of some protocol features and to add some new optional commands.

In particulars, the following new optional commands are included in this edition of the specification :

CDUP - change to parent directory.

SMNT - structure mount.

STOU - store unique.

RMD - remove directory.

MKD - make directory.

PWD - print directory.

SYST - system.

This specification is compatible with the previous edition . A Program implemented to the previous specification should automatically be in conformance to this specification.

2.1.2. TERMINOLOGY :

ACCESS CONTROL : Access controls define users access privileges to the use of a system, and to the files in the system . Access controls are necessary to prevent unauthorized or accidental use of files. It is the prerogative of a server-FTP process to invoke access controls.

BYTE SIZE : There are two byte sizes of interest in FTP : the LOGICAL byte sizeof the file and the TRANSFER byte size used for the transmission of the data.

CONTROL CONNECTION : The communicaion path between the USER-PI and SERVER-PI for the exchange of commands and replies. This connection follows TELNET protocol.

DATA CONNECTION : A full duplex connection over which data is transferred in a specified mode and type. The data transferred may be a part of a file, an entire file or a number of files. The path may be between a server-DTP and a user-DTP, or between two server-DIPs.

DATA PORT : The passive data transfer process "listens" on the data for a connection from the active transfer process in order to open the data connection.

DTP : The data transfer process establishes and manages the data connection. TheDTP can be passive or active.

FTP COMMANDS : A set of commands that comprise the control information flowing from the user-FTP to the server-FTP process.

PI : The protocol Interpreter. The user and server sides of the protocolhave distinct roles implemented in a user-PI and a server-PI.

REPLY : A reply is an acknowledgment (positive or negative) sent from server touser via the control connection in response to FTP commands. The general form of a reply is a completion code (including error codes) followed by a next string.

The codes are for use by programs and the text is usually intended for human user.

SERVER - DTP : The data transfer process, in its normal "active" state, establishes the data connection with the "listening" data port. It sets up parameters for transfer and storage, and transfers data on command from its PI. The DTP can be placed in a "passive" state to listen for,rather than initiate a connection on the data port.

SERVER DTP PROCESS :A process or set of processes whichperform the function of file transfer in cooperation with a user-FTP process and, possibly another server. The functons consist of a protocol interpreter (PI) and a data transfer process (DTP).

SERVER-PI :The server protocol interpreter "listens" on port L for a connection from a user-PI and establishes a control communication. It receives standard FTP commandsfrom the user-PI, sends replies, and governs the server - DTP.

USER-DTP :The data transfer process "listens" on the data port for a connection from a server-FTP process. If two servers are transferring data between them, the user-DTPis inactive.

USER-FTP PROCESS :A set of functions including a protocol interpreter, a data transfer process and a user interface which together perform the function of file transfer in cooperation with one or more server-FTP processes.

USER_PI :The protocol interpreter initiates the control connection from its port U to the server-FTP process, initiates FTP commands, and governs the user-DTP if that process is part of the file transfer.

2.1.3. FTP MODEL :

The model of the FTP is shown in figure - 2.1. In the model shown in the figure,the user- protocol interpreter initiates the control connection. The control connection follows the Telnet Protocol. At the initiation of the user,standard FTP commands are generated by the user - PI and transmitted to the server process via the control connection .(The user may establish a direct control connection to the server-FTP, frrom a TAC terminal for example, and generate standard FTP commands independently, bypassing the user-FTP process.) Standard replies are sent from the server-PI to the user-PI over the control connection in response to the commands.

The FTP commands specify the parameters for the data connection (data port, transfer mode, representation type, and structure) and the nature of the file system operation (store,retrieve, append,delete,etc.). The user-DTP or its designate should "listen" on the specified data port, and the server initiates the data connection and transfer in accordance with the specified parameters. It should be noted that the data port need not be in the same host that initiates the FTP commands via the control connection, but the user or the user-FTP process must ensure a "listen" on the specified data port. It ought to also be noted thatthe data connection may be used for simultaneous sending and receiving.

In another situation a user might wish to transfer files between two hosts, neither of which is a local host. The user stes up control connections to the two servers and then arranges for a data connection between them. In this manner, control information is passed to the user-PI but data is transferred between the server data transfer processes. The model of the server - server interaction is shown in fig-2.2.

The protocol requires that the control connections be open while data transfer is in progress . It is the resposibility of the user to request the closing of the control connections when finished using the FTP service, while it is the server who takes the action . The server may abort data transfer if the control connections are closed without command .

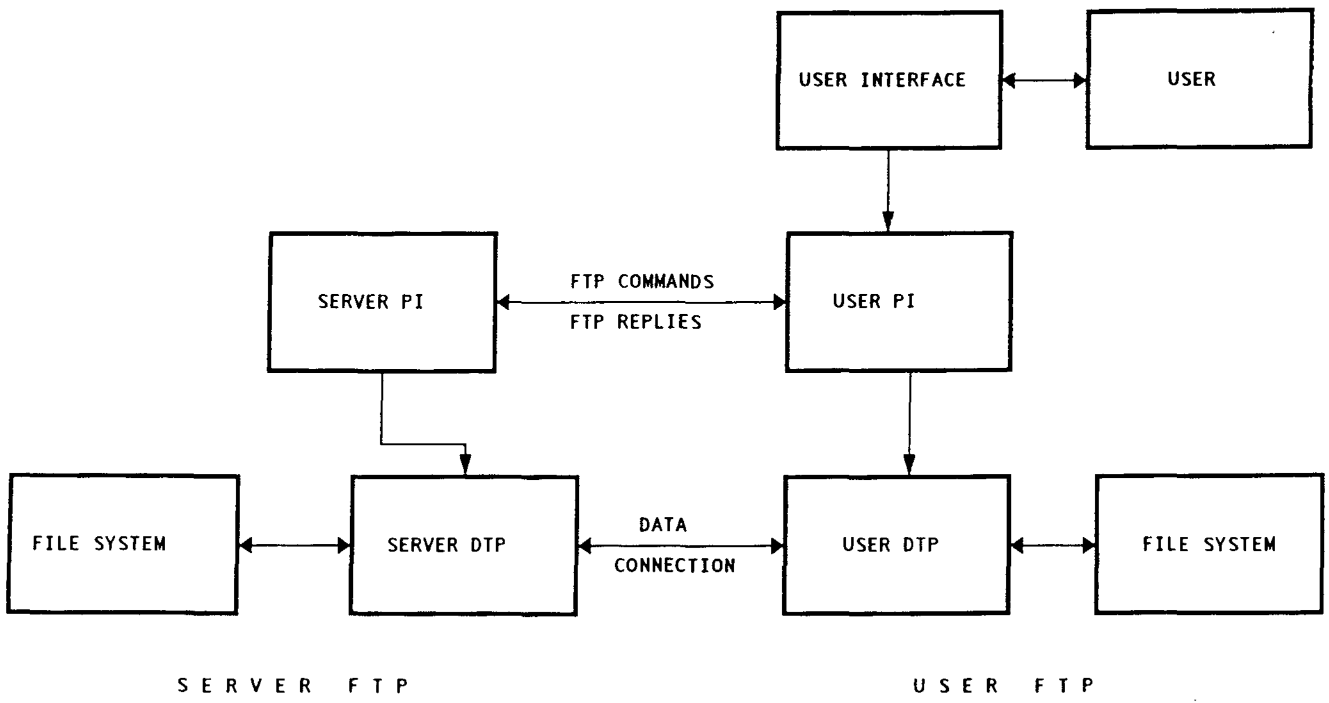


Fig. 2.1 : Mode for FTP use

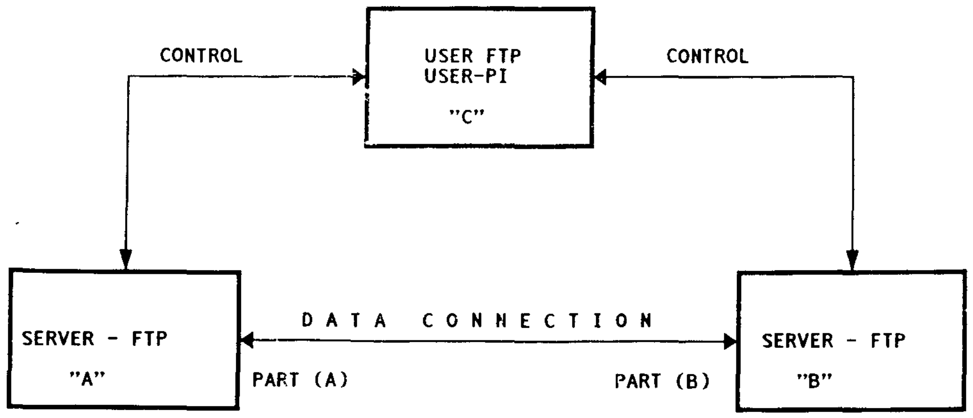


Fig. 2.2 : Model for Server-Server Interaction

2.2. DIFFERENT PROTOCOLS SUPPORTING FTP AND THEIR RELATIONSHIP :

TFTP (Trivial File Transfer Protocol) :

TFTP is a simple method of transferring files between two systems. It is designed to be small and easy to implement. So,as to provide inexpensive and unsophisticated service . Tftp provides transfer of files via UDP datagrams, so it requires very little communication software --- only IP and UDP need be running on computer. TFTP uses port number 69 for server to receive the request. Some essential features of TFTP are :

* sends 512 -octet blocks of data (expect for the last)
* Numbers the block starting with one
* supports ASCII or binary transfer
* can be used to read or write a remote file
* uses a simple header
* has no provision for user authentication.

Window NT supports both the FTP and TFTP under implementation of transferring files across the INTERNET. The differences between the two protocols are explained below :

* FTP is a complete session- oriented general purpose file transfer protocol.

TFTP is used as a barebone special pupose file transfer protocol .

* FTP can be used interactively . TFTP allows only unidirectional transfer of files.

* FTP depends on TCP, is connection -oriented, and provide reliable control.TFTP depends on UDP requires less overhead and provides no control .

* FTP provides user authentication .TFTP does not.

* FTP uses well -known TCP port numbers : 20 for data and 21 for connection dial e.g. TFTP uses UDP port number 69 for its file transfer activity.

* The windows NT FTP-server services does not support TFTP because TFTP does not support authentication.

* Windows 95 and TCP\IP for windows do not include a TFTP client program.

There are five types of packets used by TFTP. Every packets begins with a 2-byte opcode.

	opcode	string	EOS	string	EOS
	02	filename	0	mode	0
read request (RRQ)	2 bytes	n bytes	1 byte	n bytes	1 byte
	01	filename	0	mode	0
write request (WRQ)	2 bytes	n bytes	1 byte	n bytes	1 byte

	opcode	block	data
	03	block #	data
data	2 bytes	2 bytes	n bytes, 0 <= n <= 512

	opcode	block
	04	block #
acknowledgement (ACK)	2 bytes	2 bytes

	opcode	block	string	EOS
error	05 2 bytes	errcode 2 bytes	errstring n bytes	0 1 byte

TCP (Transport Control Protocol) :

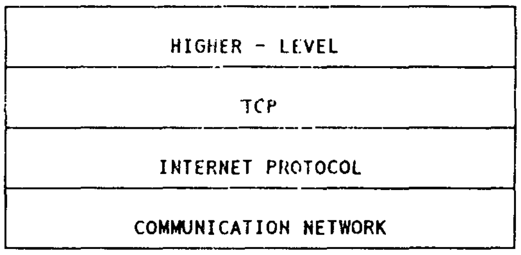
The TCP is intended for use as a highly reliable host-to-host protocol between hosts in packet-switched compute communication networks and in interconnected ststems of such networks .

The TCP is connection-oriented, end-to-end reliable protocol designed to fit into a layered hierarchy of protocols which support multi-network applications.

The TCP provides reliable inter-process communication between pairs of processesin host computers attached to distinct but interconnected computer communication networks. TCP assumes it can obtain a simple potentially unreliable datagram service from the lower level protocols. In principle, the TCP should be able to operate hard-wired connections to packet-switched or circuit-switched networks.

The TCP fits into a layered protocol architecture just above a basic internet protocol which provides a way for the TCP to send and receive variable length segments of information enclosed in internet datagram envelopes.

The internet datagram provides a means for addressing source and destination TCPs in different networks. The internet protocol also deals with any fragmentationor reassembly of the TCP segments required to achieve transport and delivery through multiple networks and interconnecting



RELATION TO THE OTHER PROTOCOL :

DIAGRAM ILLUSTRATES THE PLACE OF THE TCP IN THE PROTOCOL HIERARCHV

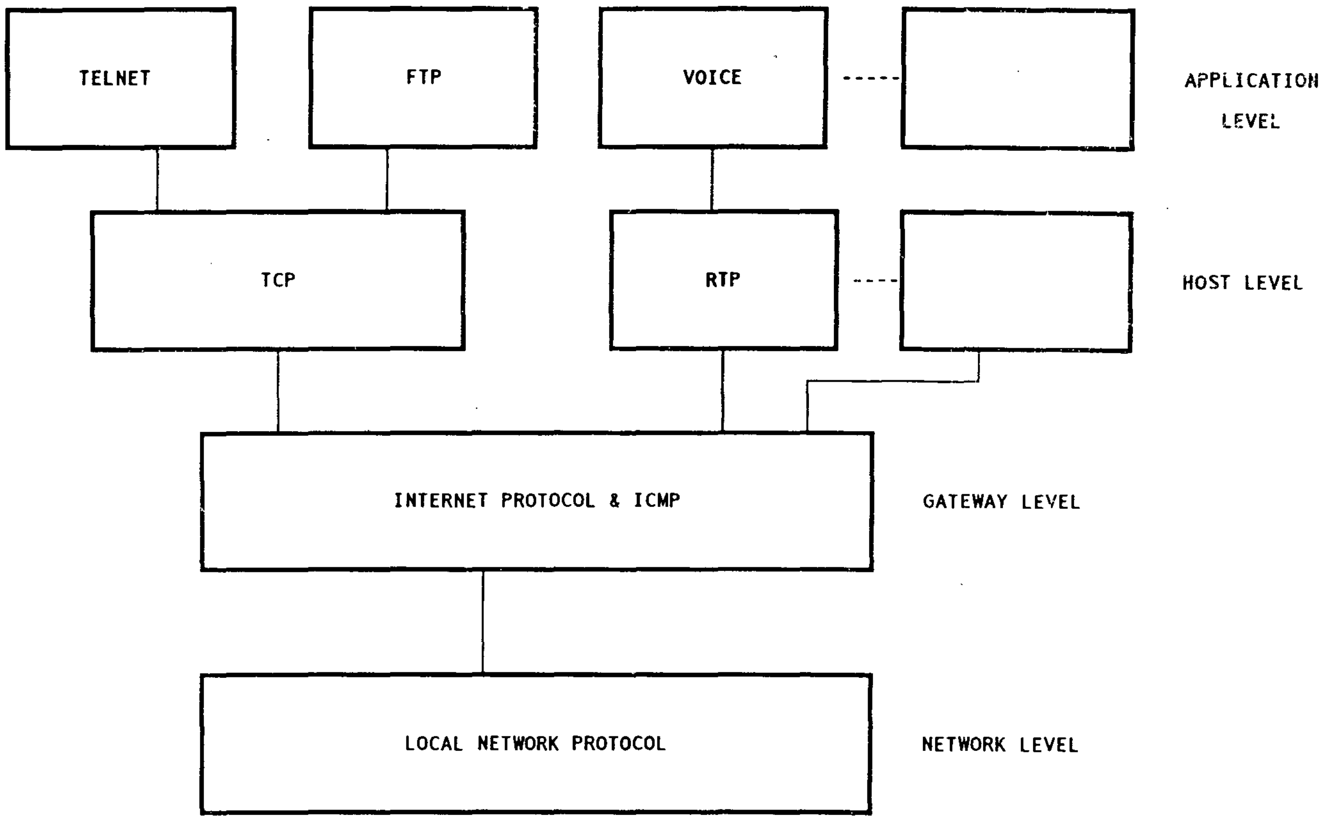


Fig. 2.3 Protocol Relationships

gateways. The Internet Protocol also carries information on the TCP segments. So, this information can be communicated end-to-end across multiple networks.

RELATION TO THE OTHER PROTOCOL :

The following fig-2.3.diagram illustrates the place of the TCP in the Protocol hierarchy.

source port	destination port	sequence number	acknowledge ment number	flags	window size	check sum	urgent pointer	options	data

2 byte 2 byte 4 byte 4 byte 2 byte 2 byte 2 byte 2 byte varies varies

fig-2.4. TCP datagram format

TCP segments are sent as internet datagrams. The Internet Protocol header carries several information fields includung the source and destination host addresses. A TCP header follows the Internet header supplying information specific to the TCP protocol. This division allows for the existence of host level protocols other than TCP.

TCP provides a reliable, correctly sequenced,flow controlled stream delivery service between the transport end-points (i.e. specific program) for any two machines on a connected internet does not preserve message boundaries .

TCP uses a technique called positive acknowledgement with retransmission in which the sending end waits to receive an acknowledgement of each segment before sending the next. The maximum communication bandwidth achievable with this scheme is maximum packet size / round trip size

which is almost certainly much lower than the network is actually able to sustain. To overcome this problem, TCP uses a " SLIDING WINDOW PROTOCOL " which allows several unacknowledged segments to be present in the network. As shown in fig. the window gratually slides down the data stream as the transmission proceeds.

Bytes behind the trailing edge of the window have been both transmitted and acknowledged. Bytes in front of the leading edge of the window have not been seen yet, to control this sliding window, there are three fields within TCP header sequence number, acknowledgement number, and the third field is used to control the size of the window. The sequence number is placed in the header by the sender and indicates the byte offset within the data stream at which this segment begins. The data is used by the recipient ensure that misordered segments are re-assembled correctly,and to reject duplicate segments. The acknowledgement number is used in the acknowledgements returned by the recipient, it indicates which segment is being acknowledged. A third packet to control size of the window is used in acknowledge packets and is filled in by the recipient to indicate how many more bytes of data the recipient is willing to accept before further acknowledgement are sent .

Consider the scenerio in fig.-2.6. A sliding window, the recipient has acknowledged receipts of bytes 1 to 6, and specified a window size of 11 . This gives the sender licence to send as far as, but not beyond, byte 17. At the instant shown on the diagram, the sender has sent up to byte 13. Bytes 14-17

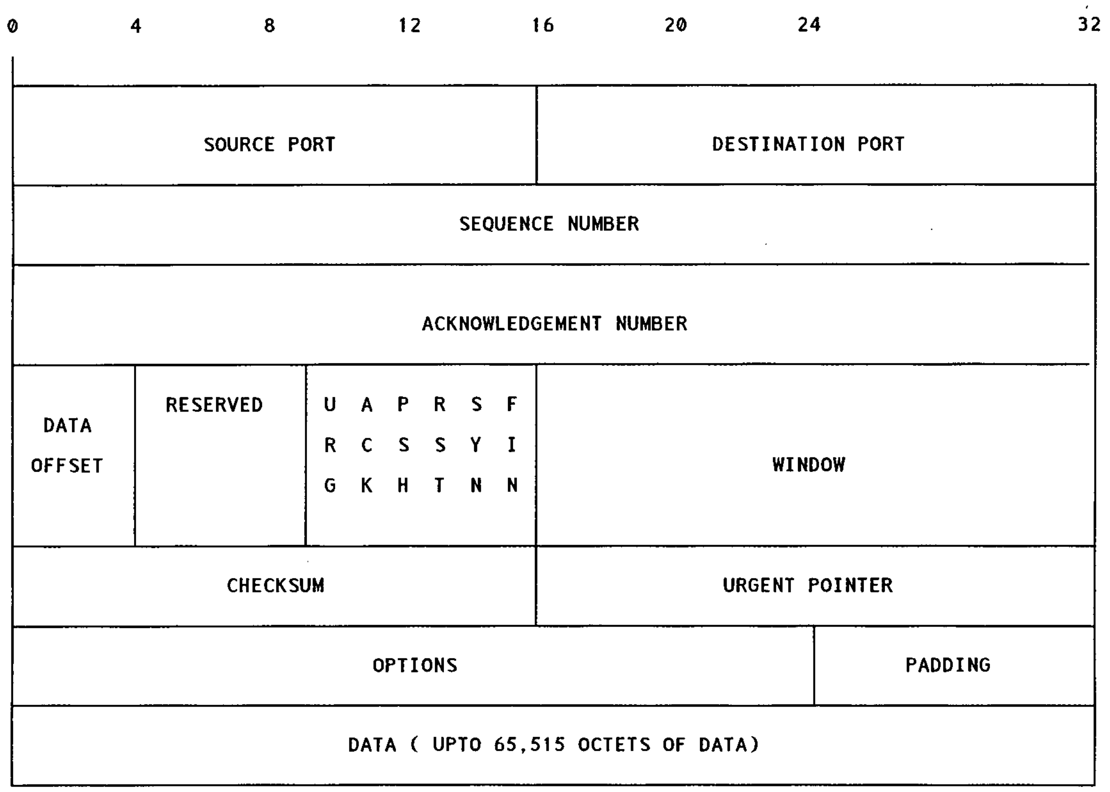


Fig. 2.5 : TCP header format

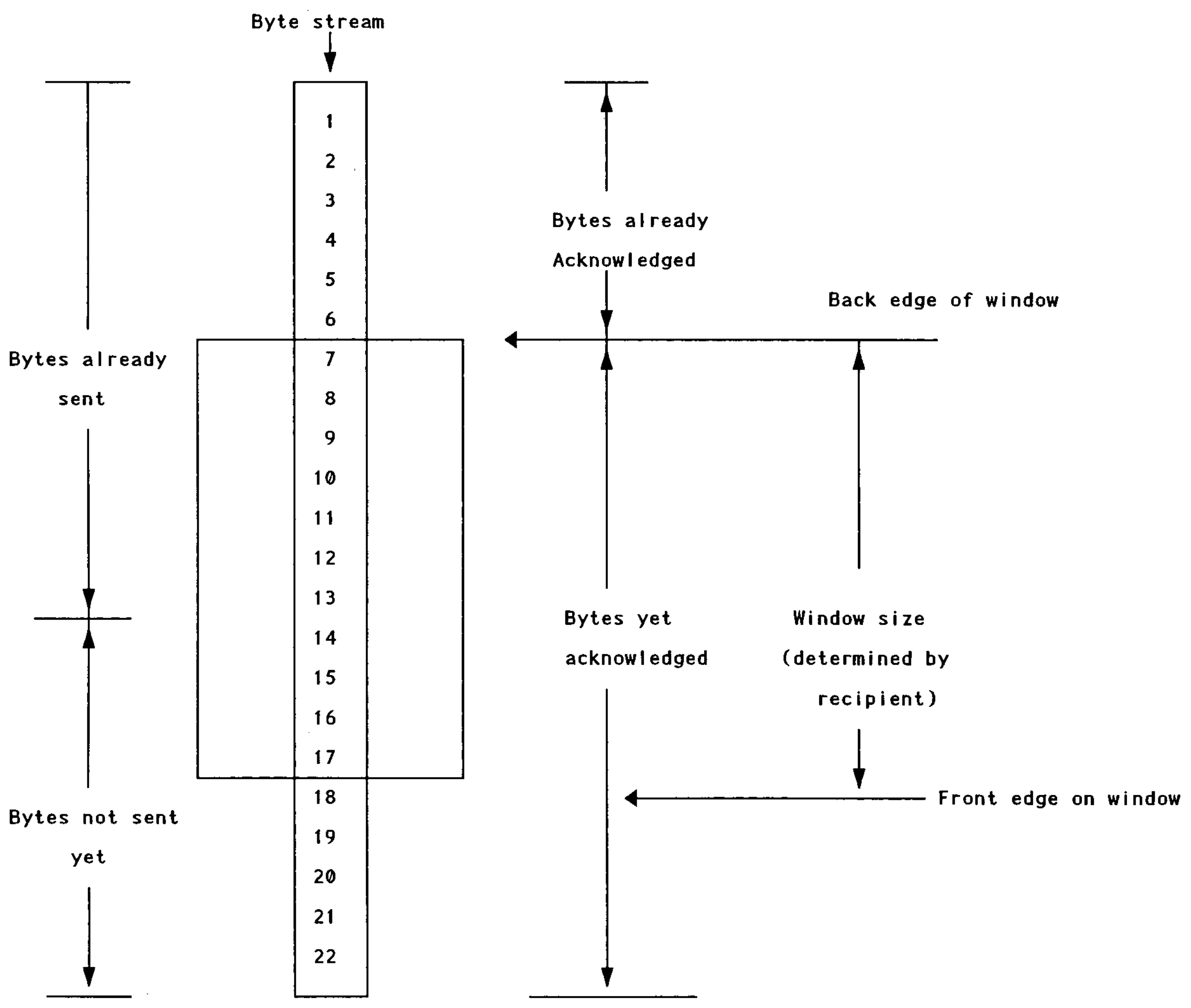


Fig. 2.6 : A sliding window

may be sent without waiting for further acknowledgements. The sliding window protocol allows TCP to flow-control the data stream accurately .

A number of important utilities were developed to take advantage of TCP\IP .Especially, well used were TELNET which allowed you to login on a remote machine and FTP which allowed transfer of files to and from remote machine .

IP (INTERNET PROTOCOL) :

Internetworking refers to any technology that joins independent networks together into a single virtual network called an INTERNET. Gateways are one of the fundamental components of the INTERNET. Gateways accept packets from one network and forward them to hosts or gateways on another. IP provides a connectionless and unreliable delivery sustem. It provides the packet delivery services for TCP, UDP, ICMP. It is connectionless because it considers each IP datagram independent

of all others. Every IP datagram contains the source address and the destination address so that each datagram can be delivered and routed independently.

IP datagram is unreliable because it does not guarantee that IP datagram ever get delivered or that they are delivered correctly.

INTERNET ADDRESSING :

The INTERNET architecture is based on the idea that every host has its own unique address. Internet addresses are composed of two parts, a network portion and a local portion. The network part of the addess specifies which network the host resides on, and the local part identifies the specific host on the network. INTERNET addess closely follows the model where hosts connect to networks and networks are combined into internets.

The fig.-2.7. shows the relationship of the protocol in the protocol suite with their approximate mapping into the OSI model. An Internet address occupies 32 bits and encodes both a network ID and host ID. A 32-bit IP address has one of the four formats shown in fig.

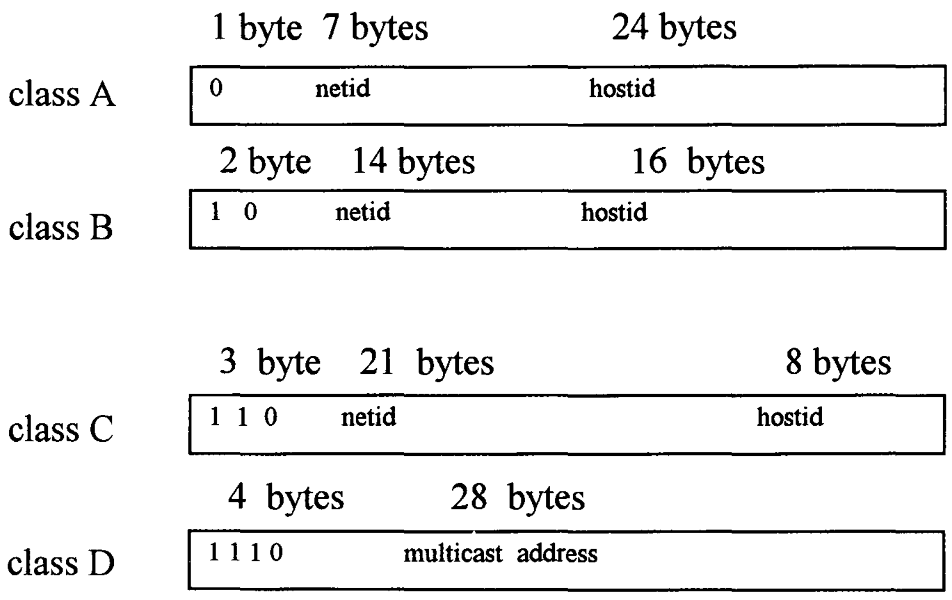


fig-2.8. Internet address formats

IP DATAGRAM :

IP datagram differ from the NETWORK FRAMES in several important ways.The header format is given in fig.-2.9. Like network frames,datagrams contain both the sender's address and the address of the intended recipient. A 16-bit checksum provides error detection for the datagram header. The checksum does not include the data portion of the datagram and uses 16-bit

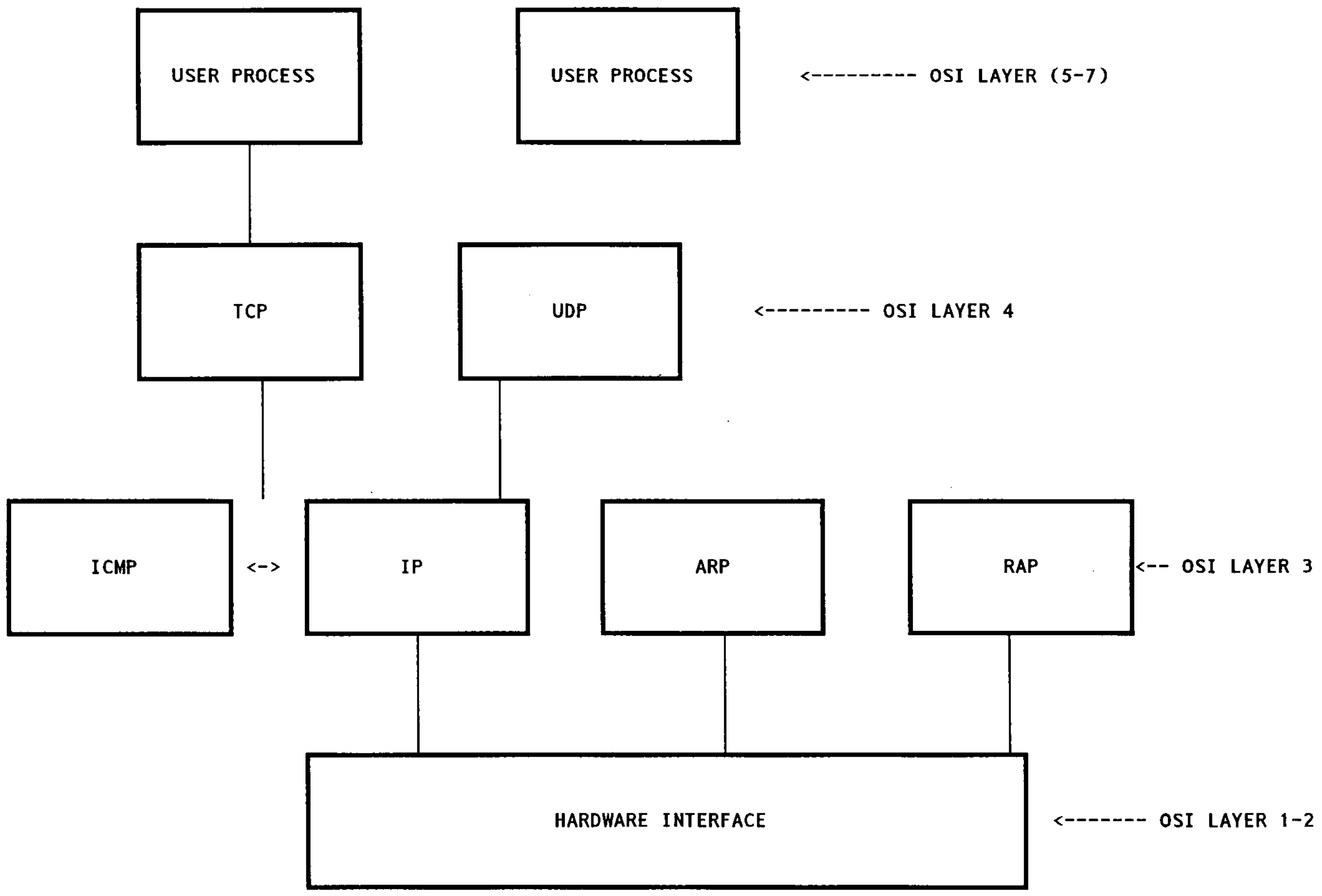


Fig. 2.7 : Relationship in protocol suit

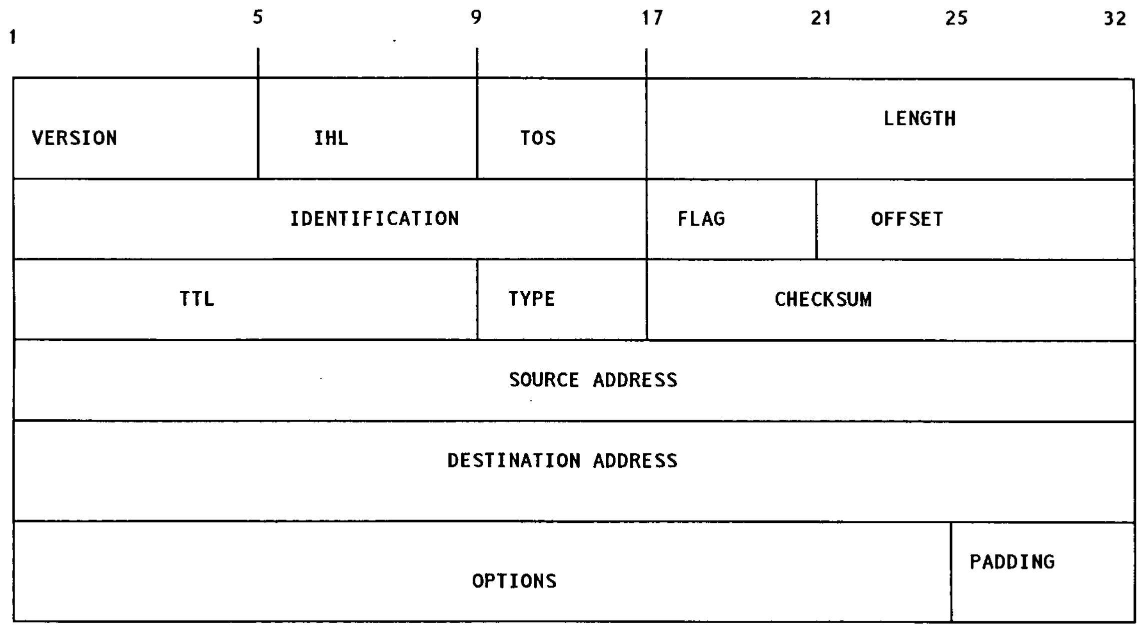


Fig. 2.9 : Format of IP datagram header

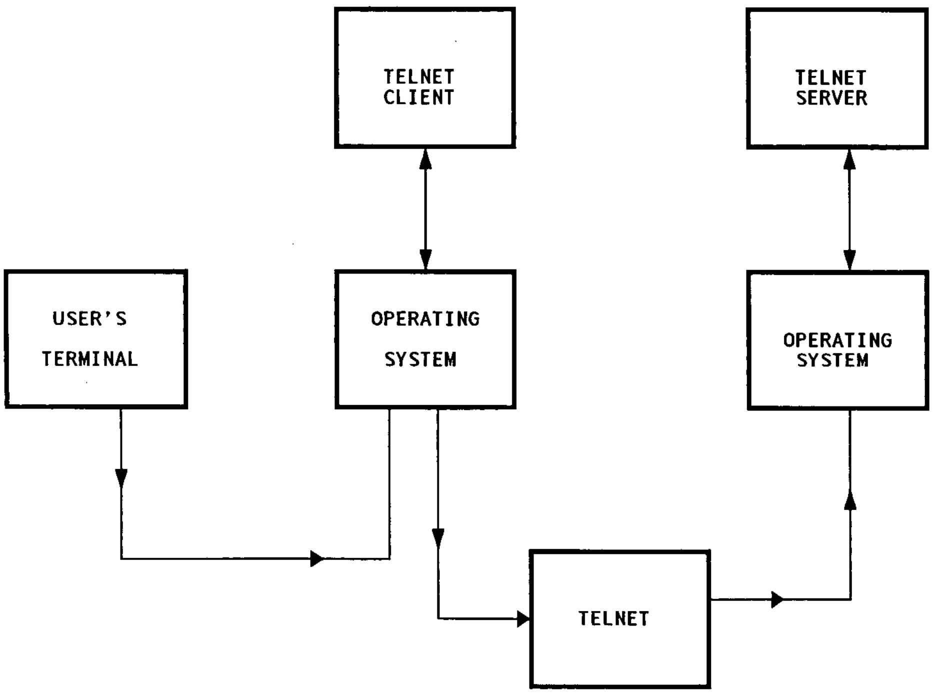


Fig. 2.11 : Path of Data in Telnet remote terminal session.

arithmetic to compute the one's complement of the "one's complement sum of the header".

An 8-bit type field identifies the type of data that the packet carries and is used to demultiplex the datagram to higher level protocol software. An 8-bit TTL field (i.e. "time to live" entry) prevents looping of datagrams endlessly around between gatways with bad routing data . Finally the header can include a variable length list of options. Thus, the header is not of fixed size and its length is given in the length field . The 16 bit length field allows datagrams to be upto 65,535 octets in size, although in practice they are usually much smaller.

To indicate where the header ends and data begins, the header includes a field containing the length of the Internet Header (IHL). the header length specifies the number of 4-octets units in the header. Thus, the data begins at an offset of 4* IHL octets from the beginning of the datagram. Because IP headers consume a minimum of 20 octets, the data portions can contain no more than 65,515 octets of data.

TRANSPORTING DATAGRAM :

IP datagrams are the universal packet of the INTERNET, and all Internet hosts and gatways understand how to process them. To send an IP datagram,the sending machine encapsulates the datagram inside a network for transmission across a directly connected network.

MAPPING IP ADDRESS into PHYSICAL ADDRESS :

One problem arises when a host wishes to send datagrams to a host or gaeway on its directly connected network : what physical address should the sender use to send datagrams to a specific INTERNET ADDRESS ? A

TH-6394

mechanism that maps Internet addresses to physical address is needed. The Address Resolution Protocol (ARP) is a special protocol designed to solve the problem of mapping between INTERNET and PHYSICAL addresses .ARP maps the INTERNET address into hardware address and RARP maps a hardware address into an internet address .

UDP (USER DATAGRAM PROTOCOL) :

Not every application finds it convenient to use the reliable stream oriented service of TCP. Some applications wantdatagram-oriented communication that understand record boundaries.The UDP provides connectionless, unreliable delivery service using IP to carry messages among machines. Indeed, UDP provides little more destinations on a machine and provides checksum to ensure data integrity. It does not use acknowledgement or retransmit lost datagrams.

Unlike TCP,UDP is not a complete transport protocol. Most important,it does not provide congestion control or flow control.Applications typiclly use UDP as a building block for more specilised protocols. The most important function of UDP is to deliver datagrams between transport end-points (i.e. specific programs) for any two machines on a connected internet. UDP preserves message boundaries but does not guarantee delivery or correct sequencing of the datagrams.

UDP provides only two features that are not provided by IP : Port numbers and an optional checksum to verify the contents of the UDP datagram. Like TCP, UDP uses 16-bit port numbers to identify the destination of a datagram . UDP port numbers completely independent from

TCP ports because the type field in the IP diagram header distinguishes TCP segments from UDP datagrams.

Source port	Destination port	Length	Checksum	Data
SS				
2 bytes	2 bytes	2 bytes	2 bytes	varies

fig- 2.10. UDP datagram format

The 5-tuple that defines an association in the internet suite consists of :-

* The protocol (TCP or UDP),
* The local host's Internet address,
* The local port number (a 16-bit value),
* The foreign host's Internet address (a 32-bit value),
* The foreign port number (a 16-bit value).

TELNET (APPLICATION LAYER PROTOCOL) :

The INTERNET provides a simple remote terminal protocol called " TELNET " which provides remote login facility. It allows an interactive user or a client system to start a login session on a remote system. Once a login session is established, the client process passes the user's keystrokes to the server process. TELNET provides three main services : first, it defines a network virtual terminal interface standard upon which application programs can be built. Second, it provides a way for a client and server to negotiate options and provides a standard set of options. Finally, TELNET treats each end of the connection symmetrically. Instead of designing one end of the connection as the terminal, either end can be a program.Interestingly, TELNET protocol can be used to access services other than the standard remote login service like connect to a different ports .

Conceptually, a client application communicates with the TELNET process on the local machine that transfer that to the remote TELNET server,and the remote application accepts data from the TELNET server on its machine.

RELATIONSHIP BETWEEN FTP AND TELNET :

The FTP uses the TELNET protocol on the control connection, this can be achieved in two ways : First, the user-PI or the server-PI may implement the rules of the TELNET protocol directly in their own procedures; or, second, the user-PI or THE server-PI may make use of the existing Telnet module in the system. Ease of implementation, sharing code, and modular programming argue for the second approach. Efficiency and independence argue the first approach.

PROTOCOL COMPARISION :

	IP	UDP	TCP
connection - oriented ?	NO	NO	YES
message boundaries ?	YES	YES	NO
data chaecksum ?	NO	OPT	YES
positive acknowledge ?	NO	NO	YES

timeout and retransmit	?	NO	NO	YES
duplicate detection	?	NO	NO	YES
sequencing	?	NO	NO	YES
flow - control	?	NO	NO	YES
full - duplex	?	NO	NO	YES
fragmentation and re-assembly of packet	?	YES	NO	NO

CHAPTER .3

THEORETICAL BACKGROUND

3.1. STREAMS:

A stream is a collection of units called “modules”, providing at one end, the head,a user/kernel interface following a prescribed protocol, and at the other end,the driver providing what is usually very similar to a traditional device driver. Device drivers are the routines that know about (and,for the most part hide) all the low level details of talking to the real hardware. The modules between the head and the driver serve as filters,transforming data as required,as servers,implementing particular data communication protocols; or as routers,choosing lower streams for routing data in and out as various devices and choosing upper streams for routing data to and from various processes.Any module may send and receive messages of its own,representing control requests or responses or error indications.

The head and driver ends of a stream are fixed when it is opened , but modules may be dynamically pushed onto a stream as needed or popped one stream to be linked under another, with the lower stream running at its head special routines defined within the upper stream,with the advent of networking, a form of driver, often called a software driver,is used to stand between user system calls, or even kernel calls and various network devices. It often impl-ements one or more layers of a networking protocol family, or acts as a filter to convert between data representations on different machines or devices.

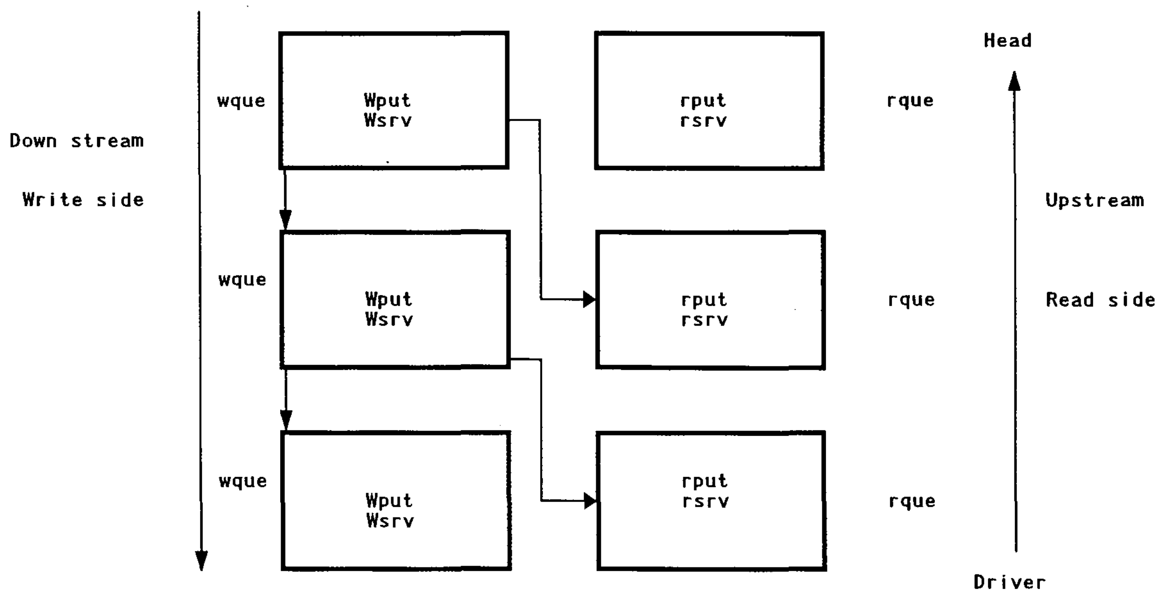


Fig. 3.1 : Module Linkage in a stream

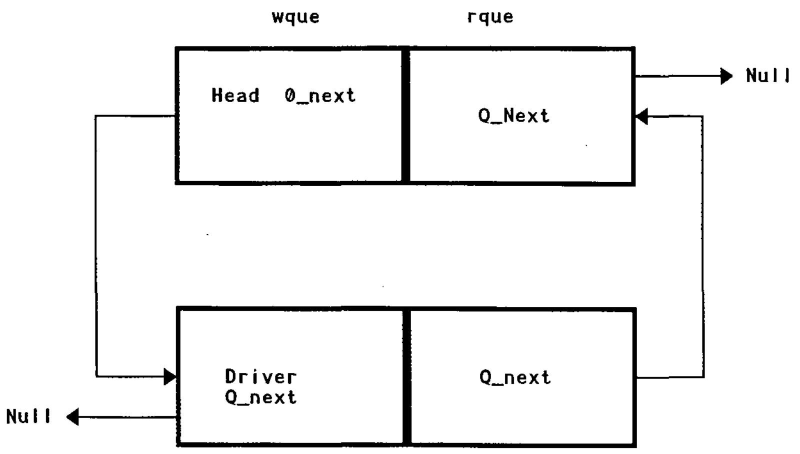


Fig. 3.2 : The Link between Queue pairs

CONSTRUCTING A STREAM:

A field dstr (driver stream) has been added to cdevsw[].(standard character device switch table). The connection between the system calls and these function is made through the standerd character device switch table cdevsw[] which is indexed by the major number of a device. The system calls use cdevsw[] to find C function thar should be used for that device .

The 'dstr' intended to point to a structure that describes the construction of a stream. When the kernal 'open' sees a non zero pointer, it sets up to invoke special fixed 'STREAM' version of 'open' and 'close' (and infact, of read,write,and ioctl). The value of dstr is the location of a STREAMTAB structure :

The STREAMTAB structure itself need be the only variable accessible by nameoutside the driver module itself. In other words,the remainder of the variables may be declaried static or may be local to perticular routines.The special STREAMS Open() procedure now carries out the following actions :-

It allocates a pair of QUEUES for the stream head, and fills these with standard stream head QINIT structure .

It allocates a pair of QUEUES for the driver and fills these with the QINIT structure described in the STREAMTAB .

3.2. QUEUE

The basic structural description of a stream is that it is a collection of pairs of queues, doubly linked in a special way. Each queue is an instance of the data structure :

```
struct QUEUE      {
                  QINIT   *q_qinfo ;
                  MSGB    *q_first ;
                  MSGB    *q_last ;
                  QUEUE   *q_next ;
                  QUEUE   *q_links ;
                   Caddr_t *q_ptr ;
                   ushort  qflag ;
                  short   q_minpsz ;
                  short   q_maxpsz ;
        }
```

The queue pairs are made up of a read side and a write side . The memory is allocated in one double sized chunk with the read member at lower add ress. Message queue belongs to either a read queue which we call an rque or the write queue which we call wque.

Message queue is IPC (inter process communication) mechanism generally known as Sysytem V IPC. A message queue is an ordered list of messages held in the kernel.H has some features in common with a shared memory segment , it is identified by a numeric key value , it has own chip and access nodes and it exists quite independently of any particular user process. Subject to usual access permissions any process that knows the key of a message queue can send the message to it and retrieve the messages from it.

Each message on queue carries user data and it is used to give some control over the order in which the messages are retrieved from queue. In term of functionality message queue are sub-ways between named-pipes and shared memory of IPC (inter process communication) . They offer more flexible access then the strictly first-in first-out manner of a pipe. But they do not offer the completely random access provided by shared memory.

3.3. SIGNAL :

A signal is an asynchronous event which delivers to process . Asynchronous means that the event can occur at any time ,it is not related to or synchronised with any particular operation in program. Broadly speaking a signal is usually a notification to a process that Somethig unexpected or unusual has happened which may require the process to leave what it is doing for a moment and take special action . Signals are sometimes called software interrupts. Signal can be sent :-

By one process to another process (to itself also)

By kernel to process

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);
```

Signals have a type (a small integer) and the types have symbolic names.

Signal name	Default Action	Descriyption
SIGHUP	Terminate process	Hangup
SIGINT	Terminate process	Interrupt character typed
SIGQUIT	Create core image	Quite character typed
SIGKILL	Terminate process	Result of kill

SIGSEGV	create core image	Invalid memory reference
SIGALAM	Terminate process	Clock time expired
SIGURG	Discard signal	Urgent condition on socket
SIGUSR1	Terminate process	User defined signal type

How a process responds to a signal :-

A process may choose how it will respond when a signal of a particular type is delivered. This choice is known as disposition of signal. There are three possibilities :-

1. The process can choose to ignore the signal
2. The process can choose to execute a special signal handler function when the signal is delivered and then continue where it lefts off
3. The process can accept default behaviour of the signal.

The disposition of a signal is set using system() system call. The genral form is : signal(sig, Handler) where sig is the signal type and handler is the address of the handler function.

3.4. IOCTL (system call/function) :

The ioctl system call is used to change the behaviour of an opened file like(fcntl).

```
#include<sys/octl.h>
int ioctl(int fd, unsigned long request, char *arg);
```

This system call performs the variety of control functions on terminal devices sockets (BSD) and streams. The greatest use for ioctl is to change the terminal characteristics, the baud rate, parity, number of bits per character, etc. The main difference between fctl and ioctl is that fctl is intended for any open file and ioctl is intended for devices specific operations.

The ioctl I_LINK system call is the system call that creates the lik.

ioctl(op_stream_fd,I_LINK,lan_stream_fd)

The op_stream_fd is becomes the Control Stream for multiplexed configuration and when it is closed the configuration is automatically dismounted.

3.5. SOCKET :

A socket *is an end point for communication . It is where the applications programs and the transport provider meet (operating system side). Sockets can only interacts with socket of same type . The system call system creates a socket (a data structure withi opeating system) i.e. creates a TSAP of given type.Comparision of Sockets TLI(Transport Layer Interface),Message queue*

	Socket	TLI	Message Queue
SERVER allocate space	---	t_aloc()	---
create endpoints	socket()	t_open()	mesg_get()
bind address	bind()	t_bind()	---
specify queue	listen()	---	---
wait for connection	accept()	t_listen()	---
get pre fd	---	t_open()	---
CLIENT allocate space	---	t_alloc()	---

create endpoints	socket()	t_open()	msgget()
bind address	bind()	t_bind()	---
connect to server	connect()	t_connecto	---
transfer data	read()	read()	msgrcv()
terminate	close()	t_close()	msgctl()

3.6. DESCRIPTION OF FSM (FINITE STATE MACHINE):

A key concept used in many protocol models is the finite state machine. With the technique ,each protocol machine (i.e. sender or receiver) is always in a specific stage at every instant of time.Its state consists of all the values of its variables,including the program counter. In most cases,a large number of state can be grouped together for purposes of analysis.For example,considering the receiver in protocol -3 ,we could extract out from all the possible states two important ones: waiting for frame 0 or waiting for frame 1.All other states can be thought of as transient,just steps on the way to one of the main states.Typically the states are chosen to be those instants that the protocol machine is waiting for the next event to happen(i.e. executig the procedure call or wait event in this examples).All this point of state of the protocol machine is completely determined by the states in the variables. The number of states is then 2**n,where n is the number of bits needed to represent all the variables combination. The state of the complete system is the combination of all the states of the two protocol machines and the channel.The state of the channel is determined by its contents.Using Protocol 3 ,the channel has four possible states:a zero frame,or a one frame moving from sender to receiver ,an acknowledgement frame going the other way or an empty channel.If we model the senderand receiver as each having two states,the complete system has 16 distinct states.

From each state ,there are zero or more possible transitions to ther states.Transition occurs when some event happens. FOr a protocol machine a transition might occure when a frame is set,when a frame arrives,when a timer goes off,when an interrupt occurs etc.For the channel,typical events are intertioon of a new frame onto the channel by a protocol machine,delivery of a frame to a protocol machine ,or loss of frame due to noise burst.

Given a complete description of the protocol machine and the channel characteristics,it is possible top draw a directed graph showing aa the states as nodes and all the transitions a directed arcs.Formally,a finite state machine(FSM)model of a protocol can be regarded as a quadruple (S,M,I,T) where :--

S—is the set of states of processes and channel can be in

M—is the set of frames that can be exchanged over

I—is the set of initial states of the processes

T—is the set of transitions between states

At the begining of time ,all processes are in the initialtates.Then events begin to happen,such as frame becoming available for transmissions or timers going off.Each event may cause one of the process of the channel to take an action and switch to new state.If there exists a set of states from which there is no exit and from which no process can be made(correct frames received), we have an error(deadlock). A less serious error protocol specification that tells how to handle an event in a state in which the event cannot occur(extraneous transition).

CHAPTER . 4

ISSUES IN DESIGN

4.1 FTP DATA STRUCTURE & DATA REPRESENTATION :

Data is transferred from a storage device in the receiving host. Often it is necessary to perform certain transformations on the data because data storage representations in the two systems are different. The sending and receiving sites would have to perform the necessary transformations between the standard representation and their internal representations.

A different problem in representation arises when transmitting binary data (no character codes) between hosts with different word lengths. It is not always clear that the sender should send data, and the receiver store it. For example when transmitting binary data (not character codes) between host systems with different word lengths. It is not always clear how the sender should send data, and the receiver store it. For example when transmitting 32-bit bytes from a 32-bit word length systems to a 36-bit word length systems, it may be desirable (for reason of efficiency and usefulness) to store the 32 bit bytes right justified in a 36 bit word in the latter system.

Data Types : data representations are handled in FTP by a user specifying a representation type

Ascii Type :This is the default type and must be accepted by all FTP implementations. It is intended primarily for the transfer of text files, except when both hosts would find the EBCDIC type more convenient.

EBCDIC type :This type is intended for efficient transfer between hosts which use EBCDIC for their internal character representation .

Image type :Image type is intended for efficient storage and retrieval of files and for transfer of binary data.

Local type :The data transferred in logical bytes of the size specified by the obligatory second parameter , Byte size. The logical byte size is not necessararily the same as the transfer byte size. If there is a difference in byte sizes, then the logical bytes should be packed continuously disregarding transfer byte boundaries and with any necessary padding at the end.

When data reaches the receiving host it will be transformed in a manner dependent on the logical byte size and the particular host.

Format Control : The types ASCII and EBCDIC also take a second(optional) parameter, that is to indicate what kind of vertical format control, if any is associated with a file. The following data representation types are defined in FTP:-

Non print, telnet format control, carriage control.

Data structure :

Three types of data structure:

1) File structure
2) Record structure
3) Page structure

To transmit files that are discontinuous, FTP defines a page structure to provide for various page sizes and associated information , each page is sent costs a page header. the page header has the following defined fields

1. Header length :-

The no. of logical bytes in the page header including this byte. The minimum header length is 4, Page index - The logical page number of this section of the file.

Data length- The number of logical bytes in the page data.

Page type - the following page types are defined :-

0=last page

1=simple page

2=descriptor page

3= ascess controlled page

The function msgget(key,flag) creates a message-queue by setting flag-value to the IPC_CREAT | PERMS , and by setting the flag argument to zero, a handle on the existing message-queue.

id = msgget(key,flag);

where id --> Handle for message-queue,

key --> Numeric key identifying the queue,

flag --> IPC_CREAT | 0644, to create new queue with access (rw-r—r--),or 0 to get an existing queue.

```
struct msgbuf{
long mtype;
char mtext; }
struct msgbuf *ptr;
```

The function msgsnd() call is used to place a message on the queue. The user is required to assemble a message with a type field at the beginning followed by the data. The syntax of msgsnd() :

msgsnd(id,ptr,size,flag);

A message is read from the message-queue using the msgrcv system call.

int msgrcv(int msqid, struct msgbuf *ptr, int length,long msgtype,int flag);

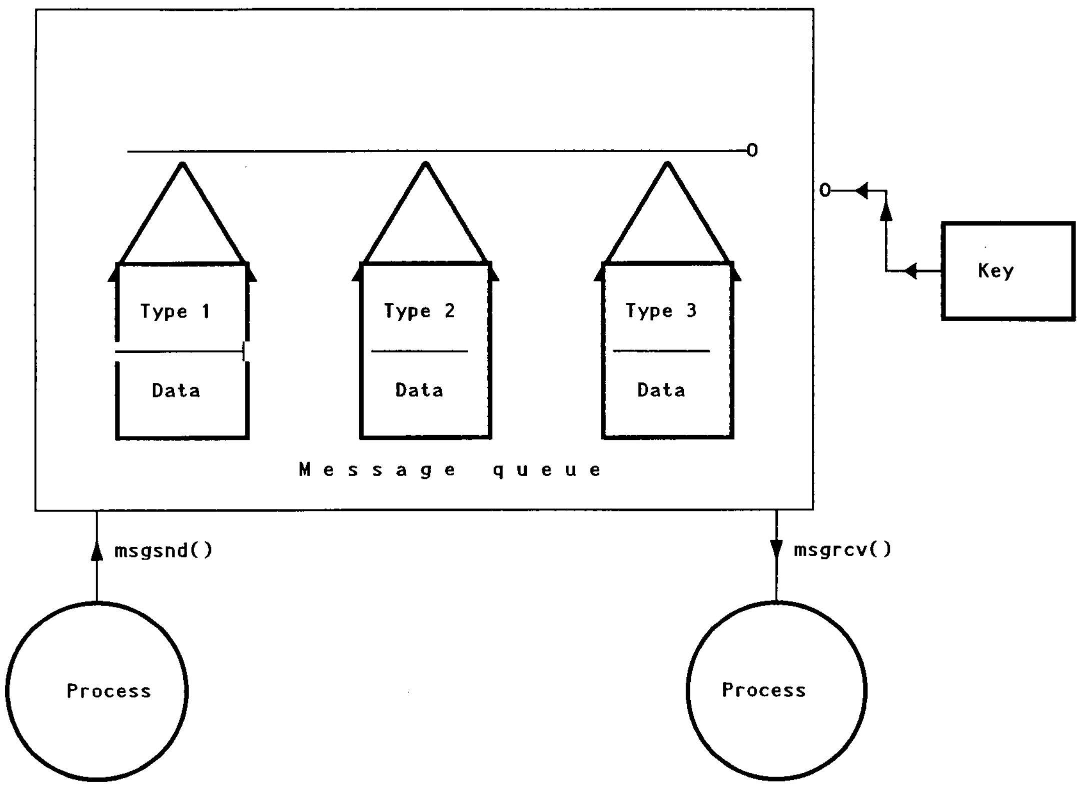


Fig. 4.1: Message queue

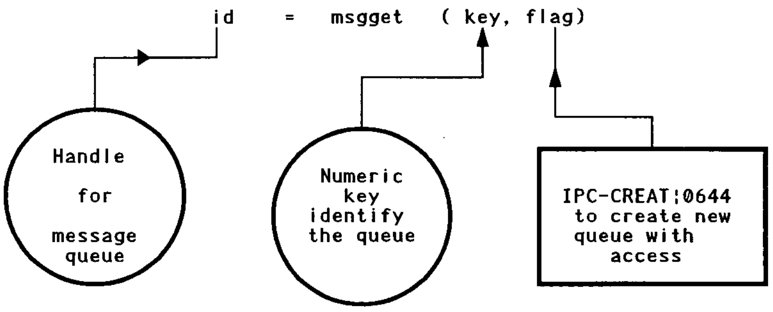

id = msgget (key, flag)
Handle for message queue
Numeric key identify the queue
IPC-CREAT¦0644 to create new queue with access

Fig. Creating a message queue

4.3. DESIGN OF SOCKET ADDRESS STRUCTURE :

Unix domain address structure :-

```
struct sockaddr_un {
                short sun_family;          /* tag : AF_UNIX */
                char  sun_path;            /* path name     */
                };
```

Internet domain address structure :

```
struct sockaddr_in {
           short sin_family;                  /* tag : AF_INET */
                u_short sin_port;            /* port number   */
                struct in_addr sin_addr;     /* IP address    */
                char sin_zero;                /* padding      */
                };
truct in_addr {
u_long s_addr;
           };
```

generic socket address structure :

```
struct sockaddr {
u_short sa_family;
char sa_data[14];
          };
```

socket types :

```
SOCK_STREAM    /* A connection-oriented transport  e.g. TCP          */
 SOCK_DGRAM    /* A connectionless transport, e.g. UDP               */
```

```
SOCK_RAW        /* Used on occasion, to talk directly to the IP Layer  */
SOCK_SEQPACKET  /* Sequenced packet socket                             */
SOCK_RDM        /* Reliably delivered message socket                   */
```

Combination of socket family,type, and protocol :

```
------------------------------------------------------------------------------
family      type              protocol      actual protocol
------------------------------------------------------------------------------
AF_INET   SOCK_DGRAM    IPPROTO_UDP   UDP
AF_INET   AOCK_STREAM   IPPROTO_TCP   TCP
AF_INET   SOCK_RAW      IPPROTO_RAW   (raw)
```

4.4 DESIGN OF CONNECTION-ORIETED CLIENTs :

FTP is a connection-oriented file-transfer protocol, uses connection-oriented clients & server mechanism . The client application allocates a socket and connects it to a server. It then sends requests across the connection and receives replies back.

Algorithm of Client :

(1.) Identify the IP address and protocol port number of the server with which communication is desired.

(2.) Allocate a socket.

(3.) Specify that the connection needs an arbitrary, unused protocol port on the local machine, and allow TCP to choose one.

(4.) Connect the allocated socket to the server.

(5.) Communicate with the server using the application-level protocol (this usually involves sending requests and awaiting replies).

(6.) close the connection.

Find the IP address and protocol port-number :

The address of srver-machine includes the IP address of the host corresponding to the server, and port number for the server. The client program accepts server's hostname as an argument and uses it to find the server's IP address. The socket interface includes library routines (inet_addr and gethostbyname) that perform the conversion descibed ahead . inet_addr takes an ASCII string that contains a dotted decimal address and returns the e quivalent IP address in binary. gethostbyname takes an ASCII string that contains the domain name for a machine. It returns the address of a hostent structure that contains, among other things the host's IP address in binary. The hostent structure is declared in the include file netdeb.h :

```
struct hostent {
            char *h_name;        /* official hostname  */
            char **h_aliases;    /* other aliases      */
            int h_addrtype;      /* address type       */
            int h_length;        /* address length     */
            char **h_addr_list;  /* list of addresses  */
            };
#define h_addr h_addr_list[0];
```

To obtain the IP address client calls 'gethostbyname' as in :

```
struct hostent *hptr;
if(hptr = gethostbyname("host")) {
/* IP address is now in hptr->h_addr   */
            }
```

```
else {
/* no entry for host in /etc/hosts file    */
                        }
```

If the call is successful, gethostbyname returns a pointer to a valid hostent structure. If the hostname can not be mapped into an IP address.

Looking up a well-known port by name :

To find the port number, the client invokes library function getservbyname, which takes two argument : a string that specifies the desired service and a stringthat specifies the desired service and a string that specifies the protocol being used. It maps the service name to port number. It returns a pointer to a structure of type servent also defined in the include file netdb.h :

```
struct servent {
        char *s_name;        /* official service name */
        char **s_aliases;   /* other aliases         */
        int  s_port;        /* port for the service  */
        char *s_proto;       /* protocol to use       */
        };
```

The client calls getservbyname in the following way :

```
struct servent *sp;
if(sp = getservbyname(service, "tcp")){
/* port number is now in sp->s_port           */
}else
                {
/* no entry for service in the  /etc/services  */
        }
```

Both getservbyname & gethostbyname returns the port-number & IP address in network byte order.

Looking up a protocol by name :

The socket interface provides a mechanism that allows a client or server to map a protocol name to the integer constant assigned to that protocol.Library function "getprotobyname" performs the look up :

```
struct protent {
        char *p_name;       /*  official protocol name     */
        char **p_aliases;  /*  list of aliases allowed    */
        int  p_proto;       /*  official protocol number  */
        };
struct protent *pptr;
if(pptr = getprotobyname("udp")) {
/* official protocol number is now in pptr->p_proto */
} else {
/*  error occurred-handle it  */
          }
```

(2.) Creating a socket :

```
#include<sys/types.h>
#include<sys/socket>
int sd;   /* socket descriptor */
sd = socket(AF_INET,SOCK_STREAM,0);
```

(3.) Setting a local address :

Unlike the server, the client does not need explicitly to bind an address to this socket. The system automatically bind an address for you, choosing an arbitrary port number. There is an important execption for clients which must bind a reserved port number (< 1024) as a verifier to the server. Thefunction 'rresvport()' is used to obtain a socket with a reserved port bound.

4. Connecting the socket to the server :

connect() system call is used to connect the client's socket to the server's.

```
retcode=connect(sd,(struct sock_addr_in*)&tcp_svr_addr),
                        sizeof(tcp_svr_addr);
```

One of the parameter to this call is a "sock_addr_in" structure which must be filled in with the address of the server i.e. remote end-point to which connection is established.

4.5 DESIGN OF CONNECTION-ORIENTED SERVER :

Connection-oriented design of server requires a separate socket for each connection, while connectionless designs permit communication with multiple hosts from a single packet. Each server follows a simple algorithm :- It creates a socket and binds the socket to the well-known port at which it desires to receive requests. It then enters an infinite loop in which it accepts the next request. Formulates a reply and sends the reply back to the client. In the concurrent connection-oriented server, the master server process accepts incoming connections and creates a slave process to handle each. Once the slave finishes, it closes the connection.

Algorithm :

master processes :

1. Create a socket and bind to the well-known address for the service being offered. Leave the socket unconnected.
2. Place the socket in passive mode, making it ready for use by server.

3. Repeatedly call accept to receive the next request from a client, and create a new slave.

slave processes :

1. Receive a connection request (i.e. socket for the connection) upon creation.

2. Interact wit the client using the connection : read request(s) and send back response(s).

3. Close the connection and exit. The slave process exits after handling all requests from one client.

Connection-oriented server operation design:

Establishing the connection :

(1.) CREATE socket :

A socket of the required address family and type is created.

```
int sock;
sock = socket(AF_INET, SOCK_STREAM,0);
```

(2.) BIND a 'well-known' port number to the socket :

An address is bound to the socket. The address consists of an IP address and a port number, and it has to be placed in a sock_addr_in structure along with the tag value AF_INET to say what kind of address this is. This structure is passed to the bind() system call.

```
#define server_port
struct sockaddr_in server;
server.sin_family = AF_INET          /* tag value */
server.sin_addr.s_addr = INADDR_ANY;
server.sin_port = htons(server_port);
```

bind(sock,(struct sock_addr*)&server, sizeof(server));

The functions htons converts the port-number from host to network byte order, which all machines can understand.

(3.) Establish a LISTEN queue for connections :

Inform the kernel to accept the connections on the socket by listen(sock,n); where n is the no. of pending connection requests
the system should queue.

(4.) Accept a connection :

The final stage in establishing communication is to wait for, and accept a connection from a client. The code is typically as follows :

```
struct sockaddr_in client;
int fd,client_len;
client_len = sizeof(client);
sfd = accept(sock, &client, &client_len);
```

The return value from accept() is a new descriptor relating to the connection now established to the client.

There are five items defining the connection between the server and client :

(a) The protocol in use,

(b) The client's IP address,

(c) The client's port-number,

(d) The server's IP address,

(e) The server's port-number.

The significance of this connection is that if any of the five items is different, it is a different connection. If another client is connected to the same server machine than one of the five items defining the connection will be

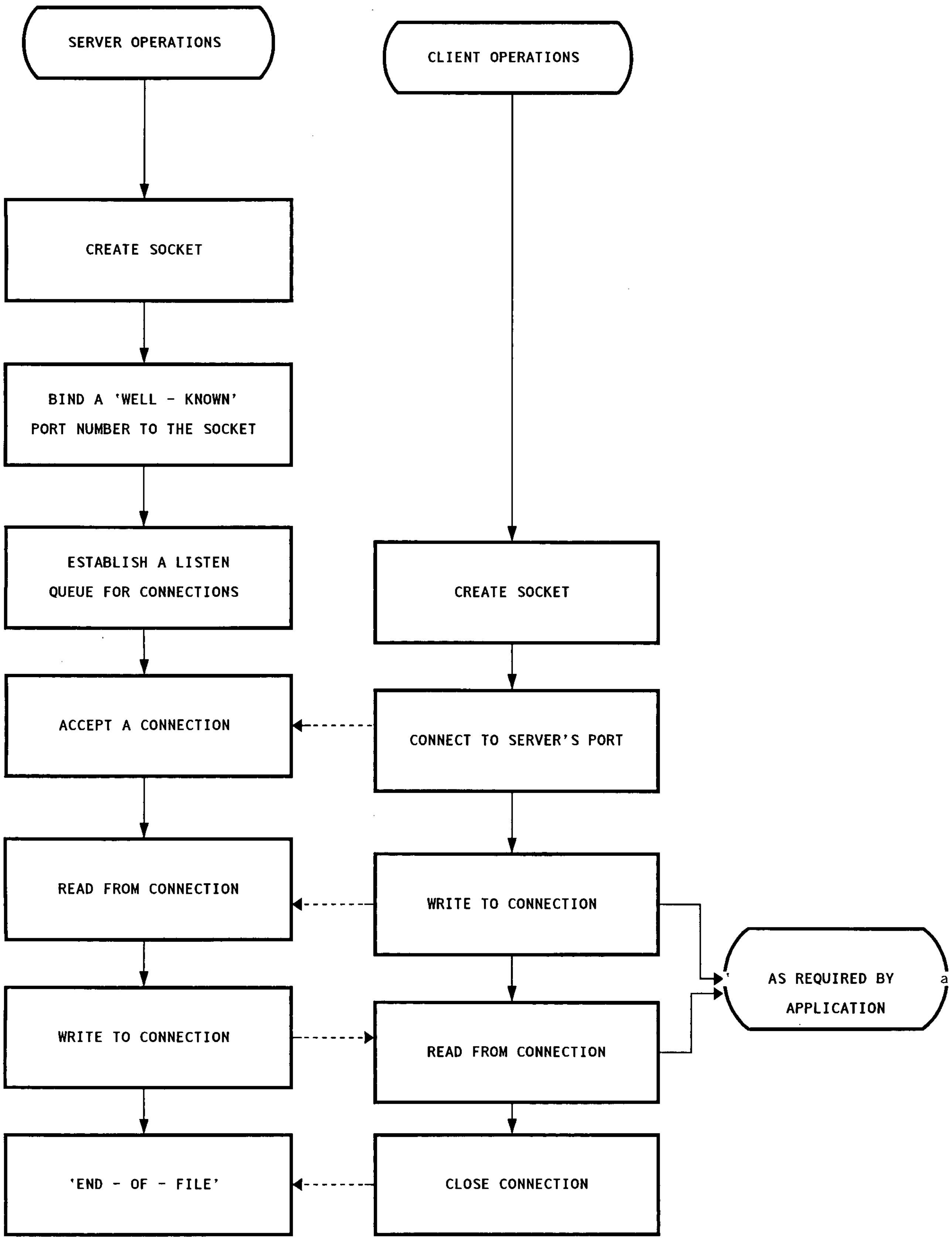


Fig-4.2 Connection-Oriented Client and Server Operations different.This approach makes the server spawn a child process to deal with each client.

Data transfer (read from a connection/write to connection) using socket :

Once the connection is estalished, the desriptor sd behaves like any other file desriptor. The Dialogue between client and server depends on the application protocol. It consists of request from the client followed by responses from the server. Process the data in inbuf[] and Placing the result in outbuf[] :

```
read(sfd,inbuf,sizeof(inbuf));
write(sfd,outbuf,sizeof(outbuf));
close(sfd);
```

Finally, when the server will finish with one client, it will usually close that connection and loop around to accept the next one.

Allocation and bindng of a server socket using TCP or UDP in passive mode :The function Passivesock(service,protocol,qlen) contains the socket allocation details, including the use of portbase. The first argument specifies the name of a service associated with a desired port,the second argument specifies the name of the protocol, and the third (used only for TCP sockets) specifies the desired length of the connection request queue. Passivesock alloctes either a datagram or stream-socket, binds the socket to the well-known port for the service, and returns the socket descriptor to its caller. When a server binds a socket to a well-known port, it must specify the address using structure sock_addr_in which includes an IP address as well as a protocol port-number. Passivesock uses the constant INADDR_ANY instead of a specific local IP address, enabling it to work either on hosts that have a single IP address or on gateways and multi-homed hosts that have a single IP address or on gateways and multi-homed hosts that have multiple IP address.

Algo :

```
struct servent *pse;     /* pointer to service information entry    */
struct protent *ppe;     /* pointer to protocol information entry   */
struct sockaddr_in sin;  /* Internet end-point address              */
memset((char*)&sin,0,sizeof(sin));
sin.sin_family = AF_INET;
sin.sin_addr.s_addr INADDR_ANY;
1.      Map servicename to port-number.
pse = getservbyname(service,protocol);
sin.sin_port = htons(ntohs(u_short)pse->s_port + portbase);
2.      Map protocol name to protocol number.
ppe = getprotobyname(protocol);
3.      Use protocol to choose a socket.
if(strcmp(protocol, "UDP") = 0) then
type = SOCK_DGRAM;
else
type = SOCK_STREAM;
4.      Allocate a socket :
s = sock(PF_INET,type,ppe->p_proto);
5.      Bind the socket :
bind(s, (struct sockaddr*)&sin, sizeof(sin));
if(type == SOCK_STREAM && listen(s,qlen) < 0)
then "can't listen on service port";
```

```
return s; /* socket-descriptor */
```

Passive socket for use in a TCP server is created as:

```
int PassiveTCP(service,qlen)
{
return Passivesock(service, "tcp", qlen);
}
```

Passive socket for use in a UDP server :

```
int PassiveUDP(service)
{
return Passivesock(service,"UDP",0);
}
```

FSM (finite state machine) Implentation for use in server :

After getting connect request server works as a FSM (finite state machine) as the server maintains information about the status of ongoing interactions with the ients. Keeping a small amount of information in the server reduces the size of messages that the client and server exchange, and allows the server to respond to requests quickly. Essentially, state information allows a server to remember the history of requests made till then by the client.

The server maintains a table that holds state information about the file currently being accessed. To transfer a file from one machine to another machine struct fsm_method is desired which elements are functions to handle various state transitions during file-transfer.

```
struct fsm_method {
void (*f)(void);
```

```
            }
struct fsm_method method_tab[] = {
            fsm q0,      /* recognise command and to machine-2    */
        fsm q1,      /* get port-number of machine-1            */
        fsm   q2,    /*   to   get   reply   from   buf   &   retrieve   file
                           from machine-1 & store on machine-2   */
        fsm qw1,     /* for ACK reply from machine-1 & wait      */
        fsm q3,      /* for ACK reply from machine-2             */
        fsm qw2,      /* for ACK wait reply from machine-2       */
        fsm q4,       /* to give transfer complete               */
        fsm qr,        /* to give transfer error                 */
              }
struct fsm_entry {
int next_state;
int method_index;
            }
struct fsmentry fsmtab[][] = {  }
```

A function to drive FSM should be designed to handle change of states during file transfer from 'q0' to 'q4' including error handling.

Syntax of drive_fsm() :

```
  {
struct fsmentry p;
present_state = Q0;
do {
p = fsmtab[present_state][rep_char - '1'];
present_state = p.next_state;
(*method_tab[p.method_index],f)();
}while(present_state != QERR && present_state != QOK);
```

To transfer file in chunks from one machine to another machine :

```
struct gf_fsm {
int next_state;
void(*f) (char*, char*, char*, int, int, int) }
```

is defined to handle different functions to control state changes.

First argument of the function should be username; second argument should be password of the user; third argument pointer to file being transferred; forth argument, socket-descriptor of the machine to which file is to be transferred; fifth argument, index of message-queue; and sixth,number of chunks.

Different states are :

1. sends a password to the remote host during connection.
2. sends a command to set the transfer type.
3. to handle an error occurrence.
4. to retrieve a file from the remote host.
5. to signal an end of file transfer.
6. to quit from file-transfer process.

DESIGN OF BFTP :

In FTP-model, server-PI will listen on a well-known port 21. The user PI will initiate the establishment of control connections by connecting to the server. Standard FTP commands are generated by the user PI to the server.

The server PI takes actions on these commands and based on the results of these actions a character string known as the FTP reply. The FTP replies are described later.Actual data is transferred over the data connections between the user DTP and the server DTP. The protocol specifies that the user DTP shall "listen" on a port, specified to the server beforehand by the PORT command.

The server-DTP initiates establishment of the data connection. The protocol requires that the control connection remains open while the data transfer is in progress.The model for BFTP is given in figure-4.3. The BFTP-model is very similar to that of the standard FTP-model.The only difference is that in the two FTP sessions which are initiated, user commands are generated from different sources. In the first FTP session,the user cmmands come from the user, while in the second FTP session the commands are generated from the queue. The two connections are shown in fig.- 4.4 .

Control connections in BFTP :

The first FTP session is started when the user first logs on. The BFTP server PI sends the standard FTP commands to the server-PI or the queue if the user so desires. This session is iterative and replies sent by the server are shown to the user. Whenever a user generates a file retrieval request, he is presented with an option to execute it in background. If the user wishes to transfer the fils in background, this request is enqueued.

After the user exits the first FTP session,the second session is opened. This FTP session has no direct involvement of user. All the commands are generated from the queued user commands. The replies sent by the server PI

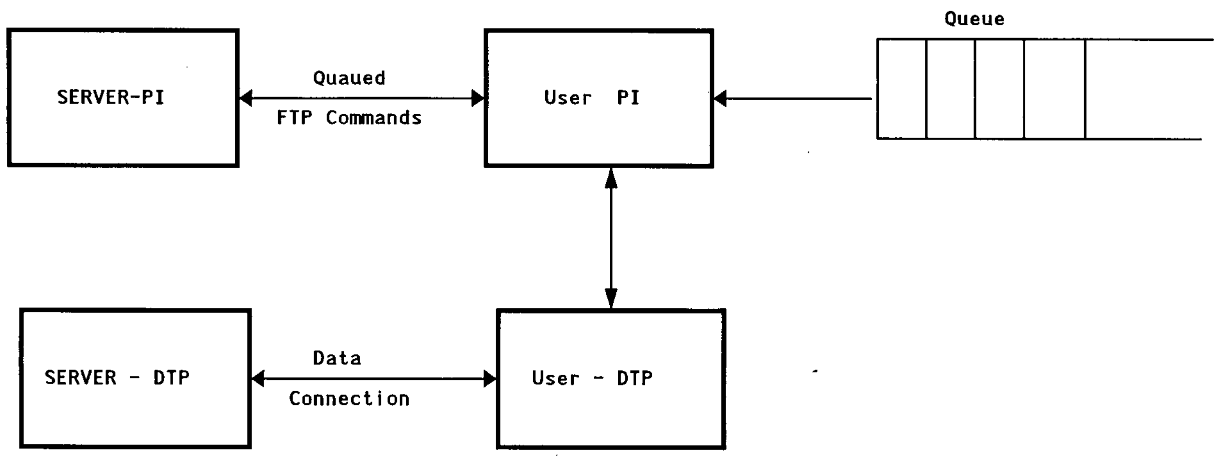


Fig. 4.3 : Model for Background FTP

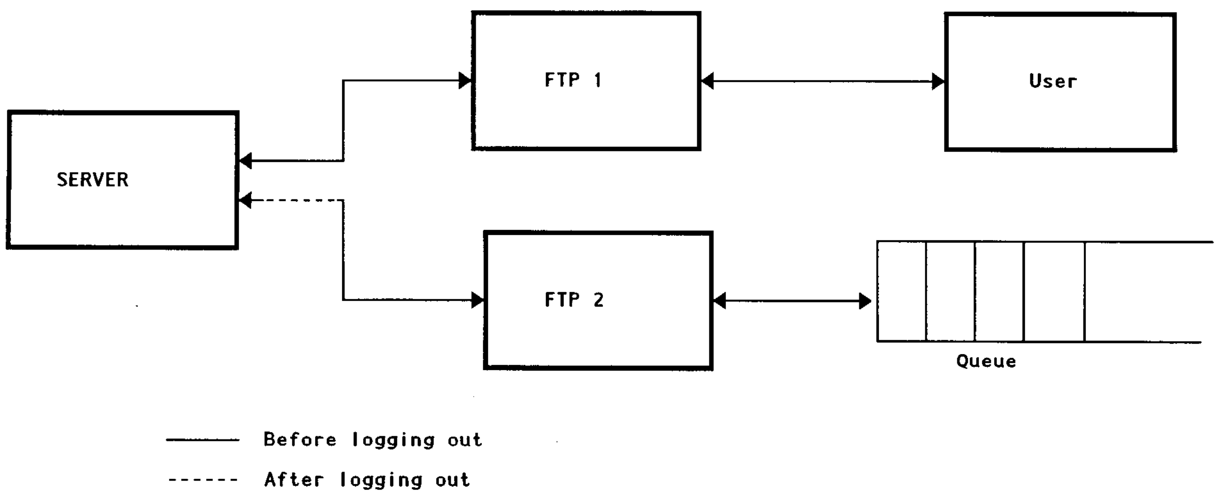


Fig. 4.4 : Control Connection in Background FTP

are not displayed to the user. The user FTP process handling this second FTP session is run in background and is dissociated from the control terminal. This process remains in existence as long as queue is non-empty.

Queueing of commands :

The second FTP session executes a set of FTP commands which are enqueued by the first FTP session. The commands are enqueued in such a way that in case of a file-retrieval, the pathname, the mode, the type of data transfer, and the user information is stored. This ensures that the file-transfer takes place correctly.

The queue of commands is maintained as a linked-list of nodes having the following structure :

1. The command which is to be executed. This command is coded in the form of an integer.
2. The local pathname : this string is NULL if not required.
3. The remote pathname : This string is null if not required.
4. The pointer to the next element in the list.

The queue is maintained in such a manner that all the file retrieval requests generated by the user are carried out correctly in background. The following commands are enqueued :

All 'cd' commands which change the currnt working directory of the server file system.

Type and mode related commands which determines the nature of file transfer.

All file retrieval and store commands which the user wishes to be processed in background.

User related information which dtermine the account to which file is transferr ed.

BFTP module :

The command Interpreter stores all the FTP functions in a table structure which consists of command-name and a pointer to the associated function. Whenever the user enters a commands, the command table is searched. If the command is found, the corresponding function is called and executed otherwise an error message is generated.

The BFTP model stores all the standrd FTP functions. This module is responsible for sending FTP commands to the server and interpreting the replies. This module opens the first FTP session at the user's request, carries out all the commands generated by the user, enqueues the ones which are relevant and opens another FTP session after the user has logged out.

After the user logs out, the BFTP module spwans a child process. This child process opens the second FTP ession with the server. The parent process exits,while the child process keeps on executing after the close of the first FTP session. As far as the user is concerned, FTP session is over. The child process which handles the second FTP session is now disassociated from it's control terminal. This is accomplished by the use of setpgrp() system call.

CHAPTER .5

ISSUES IN IMPLEMENTATION

5.1. ESTABLISHING DATA CONNECTION :

The mechanics of transferring data consists of setting up tha data connection to the apprpriate ports and choosing the parameters for transfer. Both the user and the server-DTPs have a default data port. The user process default data port is the same as the control connection port(i.e U). The server-process default data port is the port adjacent to the control connection port (i.e L-1).

The transfer byte size is 8-bit bytes.This byte size is relevant only for the actual transfer of the data; it has no bearing on representation of the data within a host's file system.

The passive data transfer process (this may be a user-DTP or a second server-DTP) shall "listen" on the data port prior to sending a transfer request command . The FTP request command determines the direction of the data transfer .The server, upon receiving the transfer request, will iniate the data connection to the port.When the connection is established , the data transfer-begins between DTP's ,and the server-PI sends a confirming reply to the user-PI.

It is possible for the user to specify an alternate data port by use of the PORT command. The user may want a file dumped on a TAC line printer or retrieved from a third party host. In the latter case,the user-PI sets up control connections with both server-PI's. One server is then told(by an FTP command) to "listen" for a connecton which the other will iniate. The user-PI

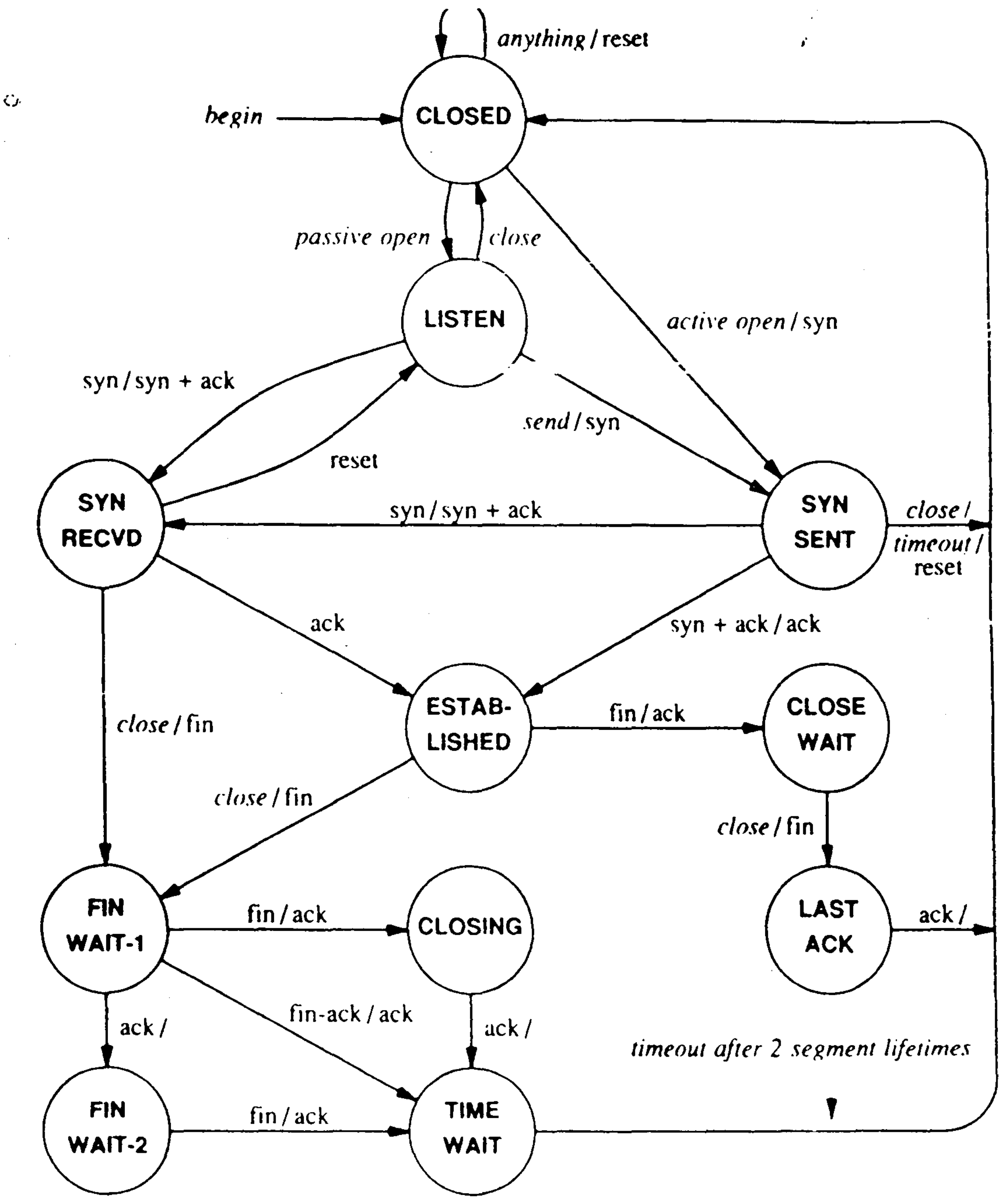


Figure-5.1. The TCP Finite State Machine that controls processing.

sends one server-PI a PORT command indicating the data port of the other. Finally ,both are sent the appropriate transfer commands. This model is shown in the figure

5.2. ESTABLISHING CONTROL CONNECTIONS :

FTP connections are reliable and connection-oriented. If successfully connected, the server sends a successful 220 reply. A two way control connection is opened between the client and server. The user-PI interpreter the commands generated by the user and sends the corresponding FTP commands over this connection. FO example, a 'dir' by the user is actually sent to the server as a LIST command. This command is then executed by the server and corresponding directory listing is sent over the data connection. Usually, in this project, the control processing of the data transfer is done by FSM (finite state machine).

5.3. FTP REPLY :

An FTP reply consists of a three digit number (transmitted as three alphanumeric characters) followed by some text. The number is intended for use by automata to determine what state to enter next; the text is intended for the human user. It is intendedthat the three digits contain enough encoded information that the user-process(the User-PI) will need to examine the text and may either discard it or pass it on to the user,as appropriate. In particular, the text may be server-dependent,so there are likely to be varying texts for each reply code.

A reply is defined to contain the 3-digit code followedd by space SP, followed by one line of text (where some maximum line length has been specified), and terminated by the Telnet end-of-line code. There will be cases however,where the text is longer than a single line. In these cases the complete text must be bracketed so the user-process knows when it may stop reading the reply (i.e stop processing input on the control connection) and go to other things.This requires a special format on the first line to indicate that more than one line is coming, and another on the last line to designate it as the last. At least one of these must contain the appropriate reply code to indicate the state of the transaction. To satisfy all functions,it was decided that both the first and last line codes should be the same.

Thus the format for multi-line replies is that the first line begin with the exact required reply code,followed immediately by a hyphen,"-"(also known as Minus), followed by text. The last line will begin with the same code,followed immediately by space SP , optionally some text,and the Telnet end-of-line code.

The three digits of the reply each have a special significance. This is intended to allow a range of very simple to very sophisticated responses by the user-process. The first digit denotes whether the response is good,bad or incomplete. (Reffering to the state diagram), an unsophisticated user-p[rocess will be able to determine its next action (proceed as planned,redo,retrench,etc.) by simply examining this first digit. A user-process that wants to know approximatelywhat kind of error occured)e.g file system error, command syntax error) may examine the second digit,reserving the third digit for thr finest gradation of information)e.g RNTO command without a preceding RNFR).

There are five values for the digit of the reply code :

1yz. Positive Preliminary Reply

The requested action is being iniated;expect another reply before proceeding with a new command. (The user-process sending another command before the completion reply would ne in violation of protocol; but server-FTP processes should queue any commands that arrive while a p[receding command is in progress.) This type of reply can be used to indicate that the command was accepted and the user-process may now pay attention to the data-connections ,mplementations where simultaneous monitoring is difficult. The server-FTP process may send at most, one 1yz reply.

2yz Positive Completion reply

The requested action has been successfully completed. A new request may be iniated.

3yz. Positive Intermediate reply

The command has been accepted, but the requested action is being held in abeyance, pending receipt of futher information. The user should send another commands specifying this information. This reply is used in command sequence groups.

4yz Transient Negative Completion reply

The command was not accepted and the requested action did not take place,but the error condition is temporory and the action may be requested again. The user vshould return to the beginning of the command sequence,if any.It is difficult to assign a meaning to "transient", particularly when two distinct sites(Server- and user-processes) have to agree on the interpretation.

Each reply in the 4yz category might have a slightly different time value,but the intent is that the user-process is encouraged to try again. A rule of thumb in determining if a reply fits into the 4yz or the 5yz(Permanent Negative) category is that replies are 4yz if the commands can be repeated without any change in command form or in properties of the User or Server)e.g the command is spelled the same with the same arguments used; the user does not change his file access or user name; the server does not put up a new implementation.)

5yz PermanentNegative Completion reply

The command was not accepted and the requested action did not take place.

The user-process is discouraged from repeating the exact request (in rthe same sequence) Even some "permanent" error conditions can be corrected, so the human user may want to direct his user process to reiniate the command sequence by direct action at some point in the future (e.g after the spelling has been changed,or the user has altered his directory status.)

In the project only the first digits are used for the purpose of driving the finite state machine.

FTP replies are explained in terms of REPLY CODES.

Reply codes by Function groups as example :

200	command okay.
500	syntax error, command unrecognized.
501	syntax error in parameters or arguments.
502	command not implemented.
214	help message.
125	data connection already open; transfer starting.
225	data connection open; no transfer in progress.

425	can't open data connection.
226	closing data connection.
426	connection closed; transfer aborted.
etc.	

Numeric order list of reply codes :

120	restart marker reply
200	command okay.
214	help message.
257	pathname created.
331	Username okay, need password.
332	Need account for login.
etc.	

CHAPTER .6.

DESCRITION OF SOFTWARE

The FTP program allows a user to log into two remote host at a time. Thus, a user can work with a two FTP servers. The interface function host is there to switch functioning between the two remote host. The connetions are to two different host with different names. That is the criteria for differentiating the remote host.

```
open jnu1 < enter >
```

Enter the login-name and password for jnu1.

```
open jnu2 < enter >
```

Enter the login-name and password for jnu2.

Thus three machines are opened : one working platfom machine, and another two host machines.

A function 'copy' has also been added which allows a user to copy (read transfer) a file from one remote host to another. The syntax of this command is

```
copy this-file that-file
```

from JNU1 to JNU2. Then the command will be

```
copy jnu1:rfc109.txt jnu2:rfc109.txt
```

The mode of the file 'ascii' or 'binary' is required to be changed corresponding the file by using command 'ascii' or 'binary' at the 'myftp' prompt before transferring the file.

2. Transfer file in CHUNKs :

The most elegant feature of the program is to transfe a file in chunks. The command for this is

getL

Its syntax is

getL file-name

It assumes that one has a shell account in the remote machine and is logged into it through this program. When the getL command is issued it asks for the size which should be provided correctly for correct transfer. The program then uses the exec service to log into the remote machine and splits the file into five chunks. All these chunks are then transferred concurrentely using different connections. The choice of five chunks is made for there are many FTP servers which do not handle more than five connections concurrentely. It is also assumed that the splitting program is installed in the remote machine. The program produces two files - myftp.split.log and myftp.log to report the status of the file transfer.

DESCRIPTION of PROJECT's PROGRAM MODULES :

Different HEADER files defined :

cmd.h :

```
#define USER   0
#define PASS   1
#define QUIT   2
#define PORT   3
#define PASV   4
#define TYPE   5
```

```
#define STRU   6
#define MODE   7
#define RETR   8
#define STOR   9
#define APPE   10
#define ALLO   11
#define RNFR   12
#define RNTO   13
#define ABOR   14
#define DELE   15
#define CWD    16
#define XCWD   17
#define LIST   18
#define NLST   19
#define HELP   20
#define NOOP   21
#define MKD    22
#define XMKD   23
#define RMD    24
#define XMD    25
#define PWD    26
#define XPWD   27
#define CDUP   28
#define XCUP   29
#define STOU   30
```

```
#define REIN   31


PARGET.H
#include <stdio.h>
#include <sys/types>
#include <sys/ipc.h>
#include <sys/msg.h>
#include "netdef.h"
#define NUM OF CHUNKS   5
#define ID OF MSGQ      (( key t ) 0 * 10)
#define T COMPLETE      10
#define T ERROR         20
#define T REPLY         30
#define MAXMESGSIZE     (MAXBUFF + 2 * (sizeof(int )))
struct my msgbuf {
            int type;
            char err str[MAXBUFF];
            };
extern FILE *logfileP;
extern char ofile[];
netdef.h
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
```

```
#include <netdb.h>
#ifdef MAXHOSTNAMELEN
#undef MAXHOSTNAMELEN
#endif
#define MAXHOSTNAMELEN 128
#define MAXBUFF       2048
#define MAXDATA        512
#define MAXFILENAMELEN 128
#define MAXLINE       512
#define HOSTNAME      "202.41.10.01"
```

Main.c :

The program module 'Main.c' defines a function

main(int argc, unsigned char *argv[])

which uses the function 'Mainloop()' to give prompt 'myftp' when value of argc

is 1, else 'socket open error' .Finally, it interfaces to the command function array to allow to execute different commands like 'open' , 'ls' ,' copy' ,'quit' 'get' , 'put' etc .

Interface.c :

The program module 'Interface.c' defines an array of commands

```
    char *cmd str[];
```

correspondingly, an array of function

```
          struct LocalFunc {
                    int (*cmd Fun) (char*); }
```

struct LocalFunc FuncArray[];

and also, an array of help related to the commands char *Help cmd str[];

```
char *cmd str[] = {
                                        "ascii",
                                        "bi",
                                        "cd",
                                        "close",
                                        "copy",
                                        "del",
                                        "dir",
                                        "get",
                                        "getL",
                                        "help",
                                        "host",
                                        "lcd",
                                        "ldel",
                                        "lls",
                                        "lmd",
                                        "lpwd",
                                        "ls",
                                        "md",
                                        "open",
                                        "put",
                                        "pwd",
                    "quit",
                    "rd",
                    "rhelp",
                    "status",
                    "user"
                    };
```
The corresponding function array :

```
    struct LocalFunc Funcarray[] = {
            Ascii,          /   int Ascii(char*);       /
            Binary,         /   int Binary(char*);      /
            ChangeDir,      /   int changeDir(char*);   /
                Close,          /   int Close(char*);       /
            Copy,           /   int Copy(char*);        /
            Delete,         /   int Delete(char );      /
            Dir,            /   int Dir(char*);         /
            GetFile,        /   int GetFile(char*);     /
            GetL,           /   int GetL(char*);        /
            Help,           /   int Help(char*);        /
            Host,           /   int Host(char *s);      /
            Lcd,            /   int Lcd(char*);         /
            Ldel,           /   int Ldel(char*);        /
                Lls,            /   int Lls(char*);         /
            Lmd,            /   int Lmd(char*);         /
            Lpwd,           /   int Lpwd(char*);        /
            List,           /   int List(char*);        /
```

```
MakeDir,      /      int MakeDir(char*);     /
Open,         /      int Open(char*);        /
StoreFile,    /      int StoreFile(char*);   /
          PresentDir,   /      int PresentDir(char*);  /
Quit,         /      int Quit(char*);        /
RemoveDir,    /      int RemoveDir(char*);   /
Rhelp,        /      int Rhelp(char*);       /
Status,       /      int Status(char*);      /
User,         /      int User(char*);        /
};
```

Related array of HELP to the commands :

```
char *help Cmd str[] = {
            "Sets transfer type to ascii "
            "Sets transfer type to binary"
            "Changes remote working directory"
            "closes a connection"
                 .
                 .
               "Logs in a user in the remote host" }
```

PROGRAM MODULES in REMOTE LOGIN :

Netopen.c

Netopen.c defines two variables, socketdescriptor1 and socketdescriptor2, to handle two machines host1 and host2.

Algorithm :

```
(1.) open the machine by using command 'open'
                    syntax : prompt> open (machine-name)
  if (socketdescriptor1 > 0 && socketdescriptor2 > 0) then
                  print "two connection are opened : host1 & host2,close one
and
                              try again. "
    else
                     mysd = Tcpopen(s,"ftp",0);    /* s is host-machine name
*/
                              to connect to server.
      if (socketdescriptor1 < 0) then strcpy(host1,s)
                           print "present host is host1"
                           socketdescriptor1 = sd = mysd;
    else if(socketdescriptor2 < 0) then strcpy(host2,s)
                           " print present host is host2"
                                    socketdescriptor = sd = mysd;
          else
         "print close one machine & try again ".
(2.) to close one machine, compare the host-name to two opened machine-
name
     if(sd == socketdescriptor1) then
         *username1 = '\0';
         *password1 = '\0';
         send_cmd(QUIT,(char*)NULL,sd);
         close(sd);
```

```
        *host1 ='\0';
        sd = socketdescriptor2;
        socketdescriptor1 = -1;
        if(sd > 0) then "print present host is host2" ;
        else
            if(sd =  socketdescriptor2)
                /* similarly try as before */
```

Tcpopen.c :

Tcpopen.c module defines a function Tcpopen(host,service,port) which connects to the server machines by opening TCP connection using 'socket' and creates 'stream-socket' to read and write n bytes on the stream socket using the functions
readn(sd,ptr,nbytes) and writen(sd,ptr,nbytes).

```
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 IP address */
      char     sin_zero[8];      /*  padding                  */
      };
```

This structure is defined in INTERNET family header file <netinet/in.h>.

```
struct servent {
      char    *s_name;          /*  official service name      */
      char    *s_aliases;       /*  alias list               */
      char    *s_port;         /*  port number, network byte order*/
      char    *s_proto;         /*  protocol to use            */
```

```
        };
```

The function getservbyname() looks up a service by getting port nummber.

```
        #include <netdb.h>
        struct  servent  *getservbyname(char *servname, char *protname);
    struct  hostent  *gethostbyname(char *hostname);
        struct hostent {
                    char *h_name;     /* name of the host-machine    */
                    char *h_aliases   /* alias list                  */
                    int  *h_addtype;  /* host address type           */
                    int  *h_length;   /* length of address           */
                    char *h_addr_list /* list of addresses from      */
                                                    /*        name        address
*/
        };
    #define h_addr  haddr_list[0]  /* first address in list */
```

does address to name mapping of the INTERNET address of client by server. The function gethostbyname() function returns a pointer to a hostent structure that is defined in <netdb.h>. The function gethostbyaddr(char *addr,int len,int type)

Algorithm of the function Tcpopen() :

This function returns socket descriptor if ok else -1 on error.

(1.) Initialize the server's Internet address structure store the actual 4-byte Internet address and the 2-byte port# below.

```
      struct sockaddr_in tcp_srv_addr;
struct servent     tcp_srv_info;
```

```
          struct hostent     tcp_host_info;
        bzero((char *) &tcp_srv_addr, sizeof(
(2.) if (service != null) then
                      if ((sp = getservbyname(service,"tcp" ==null) then
                                          error(tcp-open : unknown service )
                       else
                     tcp_srv_info = *sp;  /* structure copy  */
    if (port > 0) then it is the port of server (caller's value)
           tcp_svr_addr.sin_port = htons(port);
           else
                tcp_svr_addr.sin_port = sp->s_port;  (service-value)
    if (port <= 0) then "error : must specify either service or port"
(3.) convert the host-name as a dotted decimal number
    if ((inaddr = inet_addr(host)) != INADDR_NONE) then
                            memcpy((void*)     &tcp_svr_addr.sin_addr,     (void*)
&inaddr.sizeof(inaddr));
                    tcp_host_info.h_name = null;
        else
                     if((hp = gethostbyname(host)) == null) then "error : host-name"
          tcp_host_info = *hp;
                         memcpy((void*)    &tcp_svr_addr.sin_addr,    (void*)    hp-
>h_length);
(4.) get a reserved port
           if (port < 0) then resvport = IPPORT_RESERVED - 1.
             if ((sd = rresvport(&resvport)) < 0)
```

```
                    then "error : can't get a reserved port ".
    if((port >= 0) && (sd = socket(PF_INET, SOCK_STREAM,0) < 0)
            then " error :can't create TCP socket "
 (5.) connect to server
                                        if              ((connect(sd,(struct
sockaddr*)&tcp_svr_addr,sizeof(tcp_svr_addr)<0)
                then "error : can't connect to server ";
                close (sd);
      else return (sd);
```

cmd.c :

This program module defines an array of command

```
        static unsigned char *cmd_str[];
```

and uses a function

send_cmd(int cmd,char *data,int sfd) who sends a command and reads in only one reply.

```
 Algo of send_cmd() :
 *buf = '\0';
 strcat(buf,cmd str[cmd]);
 if (data != null) then strcat(buf,data);
 strcat(buf,"\r\n");
 if(written(sfd,buf,strlen(buf)) < 0) then "error :send_cmd socket write";
 if(ReplyToBeRead) then n = ReadReply(sfd,buf,MAXBUFF);
         if(n < 0) then "error : send_cmd socket read";
              else printf(buf);
```

Login.c :

Login.c program module logs to (i.e. connect to server machine) by using socket descriptor(sd).

syntax of function : int login(unsigned char *dummy).

Algo :

```
 (1.) if (sd > 0) then get username and match it in the user's login file-name
                         ReplyToBeRead = false;
                         send_cmd(USER,username,sd);
                         ReplyToBeRead = true;      /* check authority of user */
                         ReadReply(sd,Reply,MAXLINE);
                         /* it gives the socket-descriptor no. of the machine to be
                              connected  */
(2.)  check  the  user's  authority and  then connect the user to the machine.
        switch(rep_char)
        case 2 : if (sd = socketdescriptor1 or socketdescriptor2)
                              then match the username and get password to check
authority
           ttynoecho();
                              gets(password);
                              ttyrestore();
                              send_cmd(PASS,password,sd);
      ttynoecho() {
           ioctl(0,TCGETA, &ttyold);
                ttynew = ttyold;
      ioctl(0,TCSETA, &ttynew);
                return(0) }
```

```
   ttyrestore() { ioctl(0,TCSETA, &ttyold);  }
```

(3.) if password is correct then connect the user to that machine.

PROGRAM MODULE in BACKGROUND FILE TRANSFER :

Copy.c :

This program module transfers a file from one machine to the another machine in

Bacgronund mode.

```
struct fsm_method { void (*f)(void)  }
struct fsm_method method_tab[] = {
                                  fsm qr,
                                  fsm q1,
                                  fsm q2,
                                  fsm qw1,
                                  fsm q3,
                                  fsm qw2,
                                  fsm q4,
                                  s};
struct fsmentry{
            int next_state;
                 in method_index;
                 };
struct fsmentry fsmtab[6] [5] = {
                                {
                                 { QERR, FERR },
                    { QONE, FONE },
                    { QERR, FERR },
                    { QERR, FERR },
                    { QERR, FERR }
        },
                    {
                     { QERR, FERR },
                     { QTWO, FTWO },
                     { QERR, FERR },
                     { QERR, FERR },
                     { QERR, FERR }
        },
                  {
                     { QWAIT1, FWAIT1 },
                     { QERR,   FERR   },
                     { QERR,   FERR   },
                     { QERR,   FERR   },
                     { QERR,   FERR   }
        },
                    {
                     { QERR,   FERR   },
                     { QTHREE, FTHREE },
                     { QERR,   FERR   },
                     { QERR,   FERR   },
                     { QERR,   FERR   }
        },
                        {
                         { QWAIT2, FWAIT2 },
                         { QERR,   FERR  },
                         { QERR,   FERR  },
                         { QERR,   FERR  },
                         { QERR,   FERR  }
        },
                        {
                         { QERR, FERR },
                         { QOK,  FOK },
                         { QERR, FERR },
                         { QERR, FERR },
                         { QERR, FERR }
        }
                        };
```

Different FSM functions used:

fsm_q0(void) --> allocate socket to the machine-2 (dest-machine)

fsm_q1(void) --> get port number and connect to machine-1 source-machine

fsm_q2(void) --> rtrieve the file into buffer from machine-1 and store it
on machine-2 from buffer, get aknowledgement
from mach-1

fsm_qw1(void) --> wait till file is retrieved from machine-1

fsm_q3(void) --> transfer the file frim buffer to machine-2

fsm_qw2(void) --> wait till file on machine-2 is to transfered fsm_q4(void) --> to give acknowledgement of transfer complete

fsm_qr(void) --> to give acknowledgement of transfer error

int drive_fsm(void) --> which changes the state from present state to
final state to transfer file .

getport(char *cret, char *sret) --> get port for the machine connection

```
int drive_fsm(void)
{
  struct fsmentry p;
  present state = q0;
  do {
          p = fsmtab[present_state][rep_char - '1'];
          present_state = p.next_state;
          (*method_tab[p.method_index].f)();
    }While(present_state != QERR && present_state != QOK);
   return ((present_state == QERR) ? -1 : 0); }
 }
```

Algorithm of 'copy' function :

syntax of the function 'copy' : Copy(char *s);

```
      /*  Copy host1:source file  host2:destination file    */
```

(1.) First the syntax of the filenme along with hostname is checked.

(2.) corresponding to the hosts, the socket-descriptor is getten.

(3.) new connection will have to be opened.

(4.) File transfer process is done.

```
        t_start();  /* time at the beginning of transfer  */
        Echo = false;
```

```
fsm_q0();
drive_fsm();
Echo = true;
t_stop;
print " the time taken to transfer by function t_getrtime()"
```

TRANSFER FILE IN CHUNKs :

getL.c :

Program module transfers the file from one machine to another machine by breaking the file in 5 parts. The parted file-chunks are transferred concurrently by different 5 ports from source - machine to destination - machine.There these files are stored in a queue and finally, cocatenates all the files into original file transferred.

Algorothm of getL() :

(1.) check connectivity whether the source - machine and destination-machine areopened or not.

if (sd <0) then " not connected, use open ".

(2.) set file-name by entering the name of file.

i.e. gets(file);

(3.) get file-size.

(4.) set the username,password and hostname of the source and destination machines.

(5.) start time counter module

start = times(&tms_start);

(6.) Now fork() and let the child process to handle the large file transfer.

```
        P = fork();
(7.) close all the socket-connection (i.e. soket-descriptor) of parent process.
(8.) open the 'logfile' for the status write during file transfer.
(9.) split the remote file
        mysplit(usr, passwd,file,file-size,host);
(10.) transfer the splitted file in parallel.
        getFileInParallel(usr,passwd,host);
(11.) close the logfile and host-machine by function
        RemoveRemote(usr,passwd,host).
        struct servent *pse;
        pse = getservbyname("exec" , "tcp" );
        sockd = rexec(&host, pse->s_port,usr,passwd,"\\rm ftp.1.*,(int*)0);
        close(sockd);
(12.) give the ti
me taken to transfer the file.
```

mysplit.c :

The program module 'mysplit.c' defines a function

```
        mysplit(char *user,char *passwd, cahr *file,int file-size,char *host)
```

split the large file in 5 parts by calculating the size of each files.

Algo :

```
(1.) FILE *fp;
        struct servent *pse;
        open the file torecord the splitting session
        fp = fopen("/tmp/myftp.split.log","w");
```

(2.) give the remote execution of the split command with the appropriate arguments.It is assumed that that the remote host has the bsplit program present.

(3.) make cmd string and split infile into num 0f chunks i.e. 5 parts by bsplit(),

```
        sockfd = rexec(&host,pse->s_port,usr,passwd,cmd,(int*)null);
```

(a) define the infile and outfile as an array,

```
        char ifile[MAX];
        char ofile[MAX];
        char outfile[MAX];
```

(b) calculate the size of each and the size of the last chunk-file,

```
            size_of_each = size/count;
            size_of_last = size_of_each + size % count;
```

(c) open these files to output the corresponding files with chunk-filename.

```
            for (i = 0;i < count;i++){
                    ofp = fopen(outfile,"wb");
                    this_size = (i == count-1) ? size_of_last : size_of_each;
                    for(i=this_size; i>= MAXbuff; i -= MAXBUFF){
                     fread(buf,MAxBUFF,sizeof(char,ifp);
                     fwrite(buf,MAXBUFF,sizeof(char),ofp);}
                     if(i > 0) {
                                            fread(buf,i,sizeof(char),ifp);
                                            fwrite(buf,i,sizeof(char),ofp); }
                                fclose(ofp);
                }
```

Parget.c :

This program module defines a function

getFileInParallel(char *usr, char *passwd, char *host)

which uses the functions gf_drivefsm() (i.e. FSM implementation in GfFsm.c)

and getFileChunk() . This module get a file in parallel by opening certain number of connections to the remote host.

Algo:

(1.) Establish a message-queue here for communication with all the childs,

```
        msgqid = msgget(ID_OF_MSGQ,PERMS | IPC_CREAT );
```

(2.) Now fork the parent process into children process.

```
    for(i=0;i<NUM_OF_CHUNKS;i++)
            p = fork();
```

error conition : if a few processes have been forked and then the fork)() failed then what happens ? How to handle it ? It must be specified in the communication protocol. This protocol as yet does not handles it.

```
      if (p = 0)  then "report fork error";
              else if (p == 0) then get files in chunks .
            i.e. getFileChunk(usr,psswd,i,msgqid,host);
```

(3.) Read the message queue to check the status of the file-transfer.

```
        for( i=0;i<NUM_of_CHUNKS;i++)
                if
(msgrcv(msgqid,&msg,MAXMESGSIZE,(long)(T_REPLY),0)    <0)    then
"error reading queue.
                else
```

msgctl(msgqid, IPC_RMID);

and write the reporting transfer status to "logfileP".

(4.) close the mssage-queue.

    msgctl(msgqid, IPC_RMID);

getFileChunk.c :

This program-module defines a function

    getFileChunk(char *usr,char *passwd,int number,int msgqid,char *host);

to get the splitted files in chunks.

Algo:

(1.) open a TCP connection, sd = Tcpopen(host,"ftp",0);

(2.) Read the socket for an FTP-reply, n = ReadReply(sd,buf,MAXBUFF);

(3.) if (rep_char =='2') then call funtion 'gf_drivefsm()' to drives FSM to retrieve a file,

```
    gf_drivefsm(USER,passwd,filename,sd,msgqid,number);
    else "error during file transfer";
    msg.type = T_REPLY;
    msgsnd(msgqid,&msg,strlen(msg.err_str), 0);
```

gfFsm.c :

In this program module, various functions are defined to send a passwd to the remote host, send a command to set the transfer mode,error handling, signals an end of receive and retrieve of a file from remote host

different defined terms :

```
#define GF_ERROR -1
#define GF_START  0
```

#define GF_SENDP 1

#define GF_SETTT 2

#define GF_GETFC 3

#define GF_QUIT 4

#define GF_OVER 5

deifferent defined functions :

(1.) gf_sendp() sends a password to the remote host,

gf_sendp(char *usr,char *pass, char *file,int sockd,int msgqid,int chunk_num); send_cmd(PASS,pass,sockd);

(2.) gf_settt(char *usr,char *pass,char *file,int sockd,int msgqid,int chunk_num) sends a command to set the transfer mode.

i.e. send_cmd(TYPE,((type==ASCII) ? "A" : "I"),sockd);

(3.) gf_error(char *usr,char *pass,char *file,int sockd,int msgqid,int chunk_num) handles error during file transfer.

(4.) gf_over(char *usr, char *pass, char *file,int sockd,int msgqid,int chunknum) signals an end of receive.

(5.) gf_getfc(char *usr,char *pass, char *file, int sockd,int msgqid, int chunk_num) rtrieves a file from the remote host.

(6.) gf_quit(char *usr,char *pass,char *file,int sockd,int msgqid,int chunk_num) quits from the file transfer process,

.e. send_cmd(QUIT, (char*)NULL, sockd);

(7.) gf_drivefsm(char *usr,char *pass,char*file,int sockd,int msgqid,int chunk_num) drives the FSM for retrieving a file.

```
struct gf_fsm p;
Bool oldEcho = Echo;
```

```
        Echo = false;
        send_cmd(USER,usr,sockd)
        present_state = GF_START;
        while(present_state != GF_OVER && present_state != GF_ERROR)
        {
         p = gf_fsmtab[present_state][rep_char-'1'];
         present_state = p.next_state;
         (*p.f) (usr,pass,file,sockd,msgqid,chunk_num);
         } /* end of while */
         Echo = oldEcho;
   /* end of drive_fsm function */
```

get.c :

This program module defines getFile(char *s) to get a file from machine.

put.c :

This program module defines a function StoreFile(char *s) to store a file on a machine.

new.c :

This program module defines a function host(char *s) to switch from one machine to another machine if both machines are opened.

```
INCLUDE = cmd.h netdef.h parget.h Global.h
LIB =
myftp :  ls-dir.o Local.o Main.o New.o Login.o Readin.o Tcpopen.o Cmd.o
Tty.o
time.o Netopen.o Copy.o Dir.o Interface.o Error.o getFileChunk.o getL.o
parget.o
```

```
 gFfsm.o mysplit.o list.o put.o get.o getpassive.o list.o
cc   -o  myftp  ls-dir.o  time.o  Error.o  New.o  Login.o  Main.o  Readin.o
Tcpopen.o
Cmd.o Tty.o Local.o Netopen.o  Dir.o Interface.o Copy.o getFileChunk.o
gFfsm.o
mysplit.o parget.o getL.o get.o put.o getpassive.o list.o $(LIB)
ls-dir.o : ls-dir.c $(INCLUDE)
        cc -g -c ls-dir.c
getFileChunk.o : getFileChunk.c $(INCLUDE)
        cc -g -c getFileChunk.c
gFfsm.o : gFfsm.c $(INCLUDE)
        cc -g -c gFfsm.c
mysplit.o : mysplit.c $(INCLUDE)
        cc -g -c mysplit.c
parget.o : parget.c $(INCLUDE)
        cc -g -c parget.c
getL.o : getL.c $(INCLUDES)
        cc -g -c getL.c
time.o : time.c
        cc -g -c time.c
Copy.o : Copy.c $(INCLUDE)
        cc -g -c Copy.c
Error.o : Error.c $(INCLUDE)
        cc -g -c Error.c
Cmd.o : Cmd.c $(INCLUDE)
```

```
        cc -g -c Cmd.c
Tcpopen.o : Tcpopen.c $(INCLUDE)
        cc -g -c Tcpopen.c
Readin.o : Readin.c $(INCLUDE)
        cc -g -c Readin.c
Main.o : Main.c $(INCLUDE)
        cc -g -c Main.c
Tty.o : Tty.c $(INCLUDE)
        cc -g -c Tty.c
Dir.o : Dir.c  $(INCLUDE)
        cc -g -c Dir.c
Netopen.o : Netopen.c $(INCLUDE)
        cc -g -c Netopen.c
Interface.o : Interface.c $(INCLUDE)
        cc -g -c Interface.c
Login.o : Login.c $(INCLUDE)
        cc -g -c Login.c
dtp.o : dtp.c $(INCLUDE)
        cc -g -c dtp.c
file.o : file.c $(INCLUDE)
        cc -g -c file.c
Local.o : Local.c $(INCLUDE)
        cc -g -c Local.c
New.o : New.c $(INCLUDE)
        cc -g -c New.c
```

```
get.o : get.c
        cc -g -c get.c
put.o : put.c
        cc -g -c put.c
list.o : list.c
        cc -g -c list.c
getpassive.o : getpassive.c
        cc -g -c getpassive.c
```

USER INTERFACE

It has been attempted to provide a "USER FRIENDLY" command interface to FTP & BFTP. The commands in both are similar. This interface provides extensive prompting, defaulting, and help facilities for every command. For the list of all FTP commands, the user may enter "help <return>" at the 'myftp' prompt. To obtain information on a particular command, "help <command-name> <return>" should be entered. The 'quit' command will exit for FTP or BFTP. The user has to give at the prompt all the set of parameters with the command-name to run the command like, open <machine-name>, copy <host1:filename> <host2:filename> etc. The procedure for a BFTP user is to set up a set of parameters for the desired transfer and then submit the request to the FTC daemon. To give the user the maximum flexibility, BFTP supports two modes of submission:

To transfer a file by breaking the file in chunks, the user should run 'getL' command with its specified syntax : getL <filename> at the 'myftp' prompt. Also, a user can use 'get' and 'put' command to transfer a file with its specified file-name within different users (i.e. different login-name) on the same machine. If a user wants to see other person's file-directory by login into that person's login-name then user can use 'user' command at the 'myftp' prompt.It will ask for user-name and password of that user. The user can make new directory in that person's privilege, can delete the created directory, can change to any exiting directoryy using md,rd, and cd commands.At the same time, the user can also work into own- Background operation : To request a reliable background file transfer, the user will answer "no" to the 'myftp' prompt and "forground" when issueing a file transfer request. For the simple third party file-transfer, user can use 'copy' command with its specified syntax : copy <host1:filename> <host2:filename> at the 'myftp' prompt.

privilege.

7.1. TERMINAL SESSION :

From the beginning, a short description of working style on 'myftp'.

```
Local> c jnu1
Local -010- session 1 to jnu1 established.
DEC OSF/1 Version V2.0 (jnu1.jnu.ernet.in) tty0v.
login:s_nandan
password:
Last login : wed jan 1 05:25:18 in tty0v.
DEC OSF/1 V2.0(Rev. 240); Thu oct
....
....
Hello> cd project
Hello> ls
Copy.c     Main.c     get.o      Tty.c      myftp
.....
.....
Hello> myftp
myftp> help
ascii bi   cd   close   copy
```

```
del    dir   get    getL     help
host   lcd   ldel   lls      lmd
lpwd   ls    md     open     put
pwd    quit  rd     rhelp    status
user
myftp> rhelp
not connected.
myftp> open jnu1
connected to jnu1/ftp in FTP server (OSF/1 version 5.60) ready.
present host jnu1.
username : s_nandan ( typed by own )
331 password required for s_nandan.
password :  (type user's password of jnu1).
230 user s_nandan logged in.
myftp> ls
200 PORT command successfull.
150 opening ASCII mode data connection for /bin/ls (202.41.10.1, 1038).
total  727
-rw-r--r--   1   s_nandan  Mtech95    4618    Dec 24  16:07   Interface.c
-rw-r--r--   1   s_nandan  Mtech95    172710  Jan  1  03:47   RFC793.TXT
....
....
226 transfer complete.
time taken = 01.5200000 seconds.
myftp> open jnu2
```

```
connected to jnu2/ftp.
present host jnu2.
....
....
password :
230 user s_nandan logged in.
myftp> ls
-rw-r--r--   1   s_nandan  51    326992    Dec 24 15:17   RFC1147.TXT
-rw-r--r--   1   s_nandan  51      9425    Jan  1 03:52   r3
.....
.....
226 transfer complete.
Time taken = 0.900000 seconds
myftp> pwd
257 "/data1/users/s_nandan" is current directory.
myftp> host
Host changed to jnu2.
myftp> rhelp
214-The following commands are recognized (* =>'s unimplemented).
USER    PORT    STOR    MSAM*    RNTO    NLST    MKD    CDUP
.....
214 End of help
myftp> md report
550 report: File exits.
myftp> md remote
```

```
257 MKD command successfull.
myftp> cd remote
250 CWD command successfull.
myftp> rd remote
550 remote: No such file or directory.
myftp> cd ..
250 CWD command successfull.
myftp> rd remote
250 RMD command successfull.
myftp>host
Host changed to jnu1.
myftp> user
Username: s_rverma
331 password required for s_rverma.
password:
230 User s_rverma logged in.
myftp> ls
200 PORT command successfull.
150 Opening ASCII mode data connection for /bin/ls (202.41.10.1,1073).
total 40
-rw-r--r--  1  s_rverma  mtech95   18564  Oct 16  01:50  viewedit
.....
226 Transfer complete.
Time taken = 0.160000 seconds
myftp> lls
```

```
total 1561
-rw-r--r--  1  s_nandan  mtech95   28616  Dec  2  12:40  Interface.o
.....
myftp>get test
200 PORT command successfull.
550 test : No such file or directory.
myftp> get viewedit
200 PORT command successfull.
150 Opening ASCII mode data connection for viewedit (202.41.10.1,1076)
(18564 b.226 Transfer complete.
Time taken = 0.960000 seconds.
myftp> put Interface.o
200 PORT command successfull.
150 Opening ASCII mode data connection for Interface.o
(202.41.10.1,1077).
226 Transfer complete.
Time taken = 2.120000 seconds.
myftp> copy jnu1:RFC793.TXT jnu2:RFC793.TXT
Transfer complete
Time taken = 0.440000 seconds
myftp> copy jnu2:r3 jnu1:r3
Transfer complete.
Time taken = 0.140000 seconds
myftp>getL
Enter filename : RFC793.TXT
```

```
Please Enter file size :172710
Large file Transfer initiated.
IngetFileChunk1
Time taken = 0.520000 seconds.
Out of getFileChunk1
IngetFileChunk2
Time taken = 0.260000 seconds.
Out of GEtFileChunk2
IngetFileChunk3
Time taken = 0.440000 seconds.
Out of getFileChunk3
IngetFileChunk4
Time taken = 0.520000 seconds.
Out of getFileChunk4
IngetFileChunk5
Time taken = 0.200000 seconds.
Out of getFileChunk5
Toatl time taken = 0.870000 seconds.
Transfer over.
myftp> del RFC793.TXT
Delete RFC793.TXT ? y
250 DELE command successfull.
myftp> pwd
257 "/usr/users/s_nandan" is current directory.
myftp> status
```

```
Host1 = jnu1
Host2 = jnu2
Transfer type = Ascii
myftp> bi
200 Type set to I.
myftp> ascii
200 Type set to A.
 myftp> dir
 200 PORT command successfull.
 .cshrc
 .login
 ....
 ....
 report
 bsplit.c
 150 Opening ASCII mode data connection for file list (202.41.10.1, 1063).
 226 Transfer complete
 Time taken = 0.020000  seconds
myftp> close jnu2
closing jnu2.
221 Goodbye.
Connection to host jnu2 closed.
current host is jnu1.
myftp> open jnu2
....
```

```
myftp> quit
Sd1 = 3
Sending Quit
Closing jnu2
Sd2 = 4
Sending Quit
Closing jnu1
Goodbye
Hello>
```

CHAPTER .8

CONCLUSION AND SCOPE FOR FURTHER EXTENSION

8.1. CONCLUSION :

All the three main objective of this project remote login enhancement,background file transfer and transferring the files in chunk have been implemented and tested. The following conclusions have been drawn.

1. By opening two host machines along with working plateform machine one can switch from one unix machine to another unix machine easily by 'Host' command and he can see directory (ls command); delete any wrongly transferred files or available files on that machine. Also he can make directory or remove created directory etc.

2.a. In the background file transfer mode the third part file transfer has been done correctly without loss of any bytes of data of both 'ascii' and 'binary' type within the seperate sub-directory.
b. In the background file with deffered delivery give user the sense of navigating the server's file system reliable file-transfer is still not insured due to remote connectivity problem.

3. The transferring the files in chunks reduce the overall transfer time for the large size files but reliability is not ensured for small size-file.

8.2. SCOPE FOR FUTURE EXTENSION :

1) The present program -module is blocking in nature,this can be removed.

2) It handles only two hosts but the capability can be increased to any number of hosts. This will be quite useful.

3) Any editor can also be added to the program module so that user can see the file's content on the host machine of required irrespective of only listing of directory to see no. of byte transferred with the original number of byte to confirm correct transfer of files.

BIBLIOGRAPHY

1. Brown, Chris; " Unix Distributed Programming ", First edition, 1994; Prentice Hall Inc.

2. Stevens,W.Richards; " Unix Network Programming ", Fourth edition, Pentice Hall Inc.

3. Kochan, Steven G. & Hood, Patrick H.; " Unix Networking ", Prentice Hall Inc.

4. Comer, Douglas E. & Stevens, David L.; " Internetworking with TCP/IP ", VOL 1; Priciples, Protocols, and Architecture; Second Edition, PHI Pub.

5. Comer, Douglas E. & Stevens, David L.; " Internetworking with TCP/IP ", VOL II; Design, Implementation & Internals; Second Edition, PHI Pub.

6. Comer, Douglas E. & Stevens, David L. " Internetworking with TCP/IP", VOL III; Client-Server Programming and Applications; Second Edition,PHI Pub.

7. Feit,Sidnie; " TCP/IP " Third Edition, May'96, PHI Pub.

8. Patridge,Craig; " Innovations on Intenetworking "

9. Tanenbuam, A.S; " Computer Networks : Towards distributed Proccesing systems", PHI Pub; cliffs, New Jersy.

10. Postel, J. and Reynolds, j.; " FILE TRANSFER PROTOCOL (FTP) " RFC-959, USC/Information Science Institute; oct,1985.

11. Deschan, A. and Bradhan, R.; " BAVKGROUND FILE TRANSFER (BFTP) " RFC-1068, ISI, August 1958.

12. ISI,Univ. Of Southern California; " TRANSPORT CONTROL PROTOCOL (TCP) " RFC-793, Sept,1981.

13. Braden,R. & Postel,j.; ISI, " REQUIREMENTS FOR INTERNET GATEWAYS " RFC-1009; July,1987.

APPENDIX A

INTERNET SUPERVISOR :

With the interface of TCP/IP support into UNIX, and the development of real-time servers such as telnetd and ftpd, the number of services grew steadily, and after a while the number of daemon-processes in the system begain to get rather large. Of course, these daemons are not consuming processing time unless a client actually connects to them, because they are blocked awaiting the connection. However, they are using memory (or at the last, swap space) and they are consuming slots in the kernel's scheduling tables.

To overcome this problem, 4.3 BSD UNIX introduced the idea of a 'super-server', called inetd(or internet daemon). The inetd server waits for connection requests on behalf of many ordinary servers. When a connection request is received, inetd forks a new process and duplicates the network connection onto the appropriate server program.

Further to reduce the number of idle processes in the machines, inetd simplifies matters by taking care of details of establishing a network connection, allowing the server itself to be completely 'network-alive', and simply communicate with the client using its standard input and standard output. Thus, making server easier to write and smaller because it does not need to include any socket code. Using inetd, any UNIX filter program (i.e. one which communicates solely via its standard I/O) can be turned into a network server.

To identify which servers to listen on behalf of,and at which ports, inetd reads two files : /etc/inetd.conf and /etc/services. The /etc/inetd.conf file specifies which services to listen for, and the /etc/services file specifies at which port to listen. As example :

```
                echo      7/tcp,
                ftp_data  20/tcp,
                ftp       21/tcp,
telnet   23/tcp,
                login     513/tcp,
```

The different fields of inetd.conf are : service-name, socket-type, protocol,wait-flag, login-name, server-program, server-program arguments.

The steps performed by inetd :

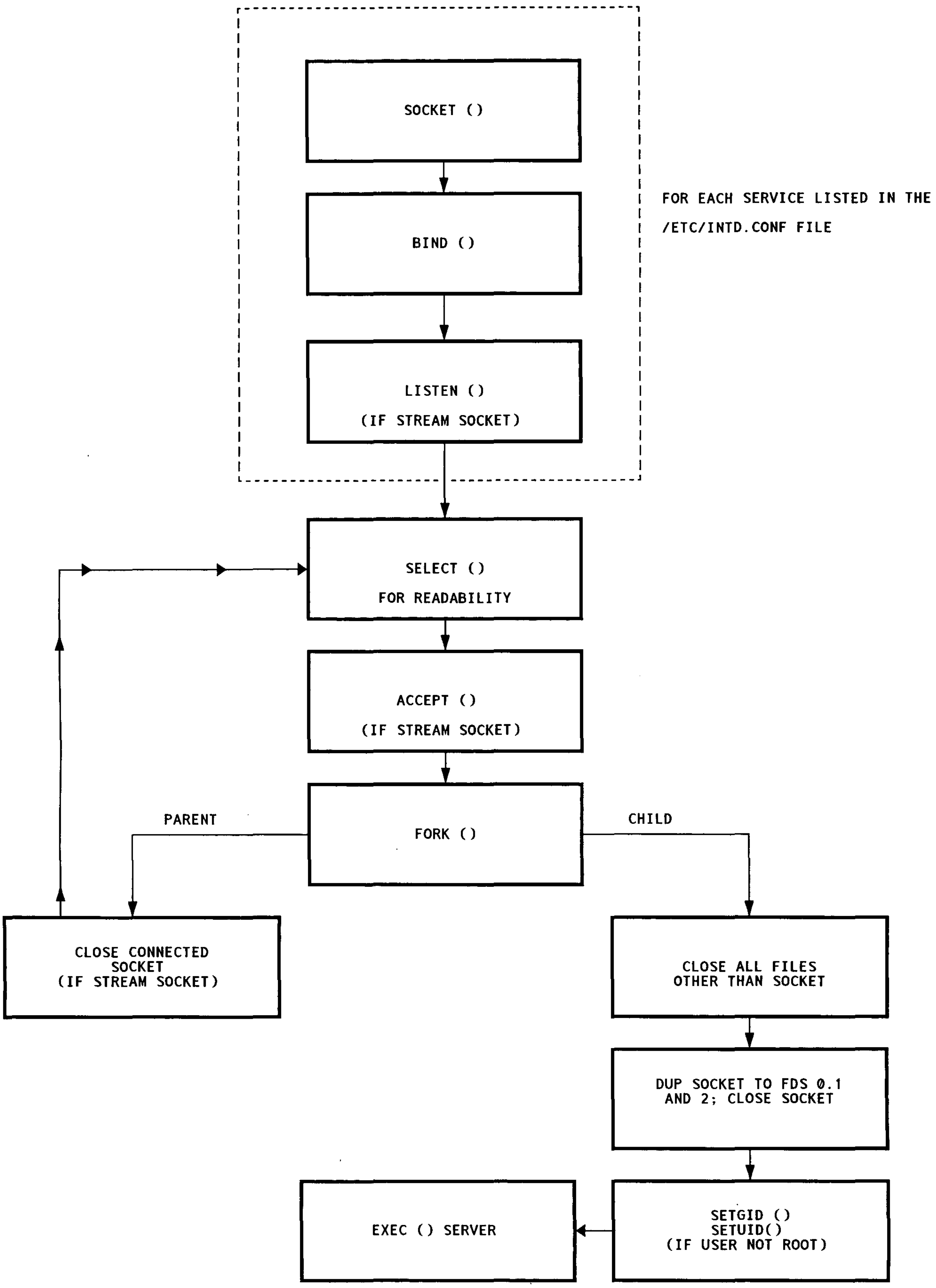


Fig. A.1 Steps performed by inetd.

SYNTAX OF FTP COMMANDS :

The command begin with a command code fallowed by an argument field. The argument feild consists of a variable length character string ending with the character string ending ending with the character string. < CRTF > (Carriage Return , Line Feed) for NVT-ASCII representation.

The following are the FTP Commands :

```
USER <SP> <username> <CRLF>
PASS <SP> <password> <CRLF>
ACCT <SP> <account-information> <CRLF>
CWD <SP> <pathname> <CRLF>
CDUP <CRLF>
SMNT <SP> <pathname> <CRLF>
QUIT <CRLF>
REIN <CRLF>
PORT <SP> <host-port> <CRLF>
PASV <CRLF>
TYPE <SP> <type-code> <CRLF>
STRU <SP> <structure-code> <CRLF>
MODE <SP> <mode-code> <CRLF>
RETR <SP> <pathname> <CRLF>
STOR <SP> <pathname> <CRLF>
STOU <SP> <CRLF>
APPE <SP> <pathname> <CRLF>
ALLO <SP> <decimal-integer>
   [<SP> R <SP> <decimal-integer>] <CRLF>
REST <SP> <marker> <CRLF>
RNFR <SP> <pathname> <CRLF>
RNTO <SP> <pathname> <CRLF>
```

```
ABOR <CRLF>
DELE <SP> <pathname> <CRLF>
RMD <SP> <pathname> <CRLF>
MKD <SP> <pathname> <CRLF>
PWD <CRLF>
LIST [ <SP> <pathname> ] <CRLF>
NLST [ <SP> <pathname> ] <CRLF>
SITE <SP> <string> <CRLF>
SYST <CRLF>
STAT [ <SP> <pathname> ] <CRLF>
HELP [ <SP> <string> ] <CRLF>
NOOP <CRLF>
```

DIFFERENT HEADER FILES:

<arpa/inet.h> ---> For internet ARP struct and functions to convert dotted decimal format and in_addr structure.

<netdb.h> ---> For struct hostent,protent & servent and functions gethostby name, getprotobyname, getservbyname.

<sys/ioctl.h> ---> For the function ioctl(stream-fd, I_STR, strioctl)
and ioctl(upstream-fd, I_LINK, lowstream-fd).

<sys/socket.h> ---> For socket address structure and related system calls as, struct sock_addr_in, and system calls socket, bind, connect, listen.

<sys/types.h> ---> Provides C definitions and data-types.

<sys/time.h> ---> Structure pointed to by the time-out argument, i.e. struct timeval; functions like t_start,t_stop & t_getrtime().

<sys/param.h> ---> For HZ, the number of clock ticks per second.

<sys/ipc.h> ---> For structure & other constants of IPC system call.

<sys/msg.h> ---> For the system call to create, open, control and IPC operations of the message-queue, like msgget(),msgrcv,msgtype.

<sys/stat.h> ---> For structure stat & file-access mode.

<netinet/in.h> ---> For internet-address family structure, and IPPROTO_xxx value.

<errno.h> ---> For system error number.

<stdarg.h> ---> For stepping through a list of function-arguments of unknown number and type, as va_start(va_list ap, last_arg)

<setjmp.h> ---> Provides a way to avoid the normal function call and return sequences, typically to permit an immediate return from a deeply nested function call.

SYSTEM CALL/FUNCTION :

rexec() : rexec() is a remote execution function on local host and uses server rexecd on remote host. The rexec() function does not need superuser privileges,since a reserved port is not required. The rexec() passes a login name and a cleartext password across the network to the server, for verification (i.e. authentication) on the remote host. The server takes the cleartext password and encripts it. The server then compares its encrypted version with the encrypted version inthe password file on the server's system. To pass the cleartext password to the fiction, the caller might have their password in the source file, which is another potential security hole.

The syntax of rexec() :

int rexec(char **ahost, int remport, char *servuname,char *password, char *cmd, int *sockfd);

This project assumes that rexec() remote function should be available on the both host machines between which file is to be transfered.

htons() & htonl() function :

These functions handle the potntial byte order differences between different computer architecture and different network protocls :

```
#include <sys/types>
#include <netinet/in.h>
u_long   htonl(u_long hostlong);  /* convert host to network,long integer   */
u_short  htons(u_long hostshort); /* convert host to network,short integer  */
u_long   ntohl(u_long netlong);   /* convert network to host, long integer  */
u_short  ntohl(u_long netshort);  /* convert network to host, short integer */
```