TFTP(Trivial File Transfer Protocol)

PROJECT REPORT

Submitted by

AJAY.RADHAKRISHNAN 9927K0113

B.JAYARAM 9927K0126

P.MOHANRAJ 9927K0140

Under the guidance of

P-874

Mr.S.MOHANAVEL BE, MBA, DCPA
Senior Lecturer, Department Of Computer Science

In partial fulfillment of the requirements for the award of the Degree of
Bachelor of Engineering
in Computer Science and Engineering of Bharathiar University, Coimbatore.

April 2003

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
KUMARAGURU COLLEGE OF TECHNOLOGY,
Coimbatore – 641 006.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
KUMARAGURU COLLEGE OF TECHNOLOGY,
Coimbatore – 641 006.

PROJECT REPORT
April 2003

CERTIFICATE

This is to certify that the project work entitled **"TRIVIAL FILE TRANSFER PROTOCOL"**submitted by the following students:

AJAY.RADHAKRISHNAN	**B.JAYARAM**	**P.MOHANRAJ**
9927K0113	9927K0126	9927K0140

In partial fulfillment of the requirements of the award of the Degree of Bachelor of Engineering in Computer Science and Engineering of Bharathiar University. is a record of bonafide work carried out by them.

[illegible] 15/03/03
Guide

Prof. S. Thangasamy
Head of the Department

Place : Coimbatore

Certified that the candidate with Register no 9927K0140

was examined by us in the project viva voice held on 18.3.03

[illegible]
Internal Examiner

[illegible]
External Examiner

ACKNOWLEDGEMENT

We owe our deepest gratitude to our patron and our Principal, **Dr K.K. Padmanabhan**, Kumaraguru College of Technology, Coimbatore for having permitted us to use the facilities in the college to do the project.

We also express our sincere thanks to the Head of the Department, **S. Thangasamy,** B.E (hons) PhD, who has been a constant source of support and encouragement, providing us with the materials necessary for the successful completion of the project.

We are also extremely thankful to our project guide, **Mr.S.Mohanavel,** B.E., MBA, DCPA Senior Lecturer, project coordinator, **Ms. S.Rajini** and our class advisor **Mr. M.N. Guptha,** Department of Computer Science and Engineering, for their invaluable guidance, support, continuous encouragement, confident words, constructive criticisms and timely suggestions rendered to us.

We also wish to all non-teaching staff members and student friends of Computer Science and Engineering department for their constant support.

SYNOPSIS

Our project Trivial File Transfer protocol, is a simple protocol to transfer files between two systems. It is used for boot strapping diskless machines and for loading the initial Operating Systems or configuration for many network devices like routers, switches, hubs, X-terminals, printers, etc. The protocol is a very basic protocol and it can read and write files to and from the remote system. This was implemented in most of the earlier operating systems and was used as the initial O/s loader for X terminals. This is now survived by the improved version Ftp which is implemented in all the recent Operating Sytems. The only difference it has from its newer version is User Authentication and Security which was not taken to much consideration in the olden days. The protocol specifies various formats for read and write requests and has been improvised with data, error and acknowledgment packets. The main advantage of the protocol is that every single packet is acknowledged separately. Our project is implemented in the Microsoft Windows platform in Java using Socket Programming. Since the protocol is the very basic of its kind lots of enhancements can be implemented. Flow control needs to be provided to maintain the connection between the TFTP client and server. It is implemented efficiently to store it in the firmware itself. Its implemented to work on any Windows platform with the same or higher capacity of 64 Mb RAM. Also many enhancements like multicast option, block size option, time out interval are designed to be added to the packets to make the protocol versatile.

TABLE OF CONTENTS

Chapter 1

INTRODUCTION

1.1 Purpose

TFTP Server Overview

TFTP stands for "Trivial File Transfer Protocol". Many network devices require a TFTP Server to load their initial operating system or configuration. Many routers, switches, hubs, X-terminals, printers, terminal servers, etc need a TFTP server in order to load their initial configuration.

The project "Trivial File transfer Protocol" is concerned with bootstrapping diskless machines and to transfer files from remote server in different modes like ASCII binary and raw bytes of data . This is implemented on top of UDP for connectionless service and TCP for Connection oriented service. This Software Specifications documents the features and constraints of this system.

1.2 Scope

Context: The project is developed such that it fits to the Linux network environment. It deals with transfer of files between two remote machines. This involves the usage of the various RFC standards and Protocols for transferring of files and various flow control techniques . It demands the use of sockets for connecting the server with the users and to run multi-users at the same time using con-current server.

Information Objectives: The requested application is the customer-visible data object and it is invoked automatically. The user needs to input the name of the application as the input data object.

Function and Performance: The software searches for the specified application over the network, transfers it to the host system and invokes it using the overlay concept.

1.3 Definitions, Acronyms and Abbreviations

Protocol: A set of rules that governs the operation of functional units such as system, terminals, etc. to achieve communication.

Connection-oriented data transfer: A protocol for exchanging data in which a logical connection is established between the end points (e.g., virtual circuit).

Connectionless data transfer: A protocol for exchanging data in which a logical connection is not established previously between the end points .

Socket: A socket is a data structure maintained by a BSD-Unix system to handle network connections. A socket is created using the system call ``socket". It returns an integer that is like a file descriptor: it is an index into a table and ``read" and ``write" to the network using this socket file descriptor.

System Call: The direct entry points provided by the kernel through which an active process can obtain services from the kernel.

Inter Process Communication (IPC): The facilities provided by the operating system for two processes to communicate with each other (pipes, FIFO, message queues, semaphores, shared memory).

IP Address: Internet Protocol Address (a 32-bit address).

TCP: Transmission Control Protocol.

UDP: User Datagram Protocol

TID: Transfer Identifiers.

Chapter 2

2.1 SYSTEM STUDY

2.1.1 Introduction:

TFTP is a very simple protocol used to transfer files. It is from this that its name comes, Trivial File Transfer Protocol or TFTP. Each nonterminal packet is acknowledged separately. This document describes the protocol and its types of packets. The document also explains the reasons behind some of the design decisions.

Purpose

TFTP is a simple protocol to transfer files, and therefore was named the Trivial File Transfer Protocol or TFTP. It has been implemented on top of the Internet User Datagram protocol (UDP or Datagram) [2] so it may be used to move files between machines on different networks implementing UDP. (This should not exclude the possibility of implementing TFTP on top of other datagram protocols.) It is designed to be small and easy to implement. Therefore, it lacks most of the features of a regular FTP. The only thing it can do is read and write files (or mail) from/to a remote server. It cannot list directories, and currently has no provisions for user authentication. In common with other Internet protocols, it passes 8 bit bytes of data.

Three modes of transfer are currently supported: **netascii** (This is ascii as defined in "USA Standard Code for Information Interchange" Also it is 8 bit ascii. The term "netascii" will be used throughout this document to mean this particular version of ascii.); **octet** (binary transfer) . Raw 8 bit bytes; **mail**, netascii characters sent to a user rather than a file. Additional modes can be defined by pairs of cooperating hosts.

2.1.2 Overview of the Protocol :

Any transfer begins with a request to read or write a file, which also serves to request a connection. If the server grants the request, the connection is opened and the file is sent in fixed length blocks of 512 bytes. Each data packet contains one block of data, and must be acknowledged by an acknowledgment packet before the next packet can be sent. **A data packet of less than 512 bytes signals termination of a transfer.** If a packet gets lost in the network, the intended recipient will timeout and may retransmit his last packet (which may be data or an acknowledgment), thus causing the sender of the lost packet to retransmit that lost packet. The sender has to **keep just one packet** on hand for retransmission, since the lock step acknowledgment guarantees that all older packets have been received. Notice that both machines involved in a transfer are considered senders and receivers. One sends data and receives acknowledgments, the other sends acknowledgments and receives data. Most errors cause termination of the connection. An error is signaled by sending an error packet. This packet is not acknowledged, and not retransmitted (i.e., a TFTP server or user may terminate after sending an error message), so the other end of the connection may not get it. Therefore timeouts are used to detect such a termination when the error packet has been lost. Errors are caused by three types of events:

- Not being able to satisfy the request (e.g., file not found, access violation, or no such user),
- Receiving a packet which cannot be explained by a delay or duplication in the network (e.g., an incorrectly formed packet)
- Losing access to a necessary resource (e.g., disk full or access denied during a transfer).

TFTP recognizes only **one error condition** that does not cause termination, the source port of a received packet being incorrect. In this case, an error packet is sent to the originating host.

This protocol is very restrictive, in order to simplify implementation. For example, the fixed length blocks make allocation straight forward, and the lock step acknowledgement provides flow control and eliminates the need to reorder incoming data packets.

2.1.3 Relation to other Protocols :

As mentioned TFTP is designed to be implemented on top of the datagram protocol (UDP). Since Datagram is implemented on the Internet protocol, packets will have an Internet header, a Datagram header, and a TFTP header. Additionally, the packets may have a header (LNI, ARPA header, etc.) to allow them through the local transport medium. As shown in the Figure, the order of the contents of a packet will be: local medium header, if used, Internet header, Datagram header, TFTP header, followed by the remainder of the TFTP packet. (This may or may not be data depending on the type of packet as specified in the TFTP header.) TFTP does not specify any of the values in the Internet header. On the other hand, the source and destination port fields of the Datagram header are used by TFTP and the length field reflects the size of the TFTP packet. The transfer identifiers (TID's) used by TFTP are passed to the Datagram layer to be used as ports; therefore they must be between 0 and 65,535. The initialization of TID's is discussed in the section on initial connection protocol.

The TFTP header consists of a 2 byte opcode field which indicates the packet's type (e.g., DATA, ERROR, etc.) These opcodes and the formats of the various types of packets are discussed further in the section on TFTP packets.

2.1.4 Initial Connection Protocol

A transfer is established by sending a request (WRQ to write onto a foreign file system or RRQ to read from it), and receiving a positive reply, an acknowledgment packet for write, o the first data packet for read. In general an acknowledgment packet will contain the block numbe of the data packet being acknowledged. Each data packet has associated with it a block number block numbers are consecutive and begin with one. Since the positive response to a write reques is an acknowledgment packet, in this special case the block number will be zero. (Normally, sinc an acknowledgment packet is acknowledging a data packet, the acknowledgment packet will contain the block number of the data packet being acknowledged.) If the reply is an error packet. then the request has been denied.

In order to create a connection, each end of the connection chooses a TID (Transfer Identifiers) for itself, to be used for the duration of that connection. The TID's chosen for a connection should be randomly chosen, so that the probability that the same number is chosen twice in immediate succession is very low. Every packet has associated with it the two TID's of the ends of the connection, the source TID and the destination TID. These TID's are handed to the supporting UDP (or other datagram protocol) as the source and destination ports. A requesting host chooses its source TID as described above, and sends its initial request to the known TID 69 decimal (105 octal) on the serving host. The response to the request, under normal operation, uses a TID chosen by the server as its source TID and the TID chosen for the previous message by the requestor as its destination TID. The two chosen TID's are then used for the remainder of the transfer.

As an example, the following shows the steps used to establish a connection to write a file. Note that WRQ, ACK, and DATA are the names of the write request, acknowledgment, and data types of packets respectively

1. Host A sends a "WRQ" to host B with source= A's TID, destination= 69.

2. Host B sends a "ACK" (with block number= 0) to host A with source= B's TID, destination= A's TID.

At this point the connection has been established and the first data packet can be sent by Host A with a sequence number of 1. In the next step, and in all succeeding steps, the hosts should make sure that the source TID matches the value that was agreed on in steps 1 and 2. If a source TID does not match, the packet should be discarded as erroneously sent from somewhere else. An error packet should be sent to the source of the incorrect packet, while not disturbing the transfer. This can be done only if the TFTP in fact receives a packet with an incorrect TID. If the supporting protocols do not allow it, this particular error condition will not arise.

A SIMPLE EXAMPLE

The following example demonstrates a correct operation of the protocol in which the above situation can occur. Host A sends a request to host B. Somewhere in the network, the request packet is duplicated, and as a result two acknowledgments are returned to host A, with different TID's chosen on host B in response to the two requests. When the first response arrives, host A continues the connection. When the second response to the request arrives, it should be rejected, but there is no reason to terminate the first connection. Therefore, if different TID's are chosen for the two connections on host B and host A checks the source TID's of the messages it receives, the first connection can be maintained while the second is rejected by returning an error packet.

2.1.5 TFTP Packets :

TFTP supports five types of packets, all of which have been mentioned above:

opcode	operation
1	Read request (RRQ)
2	Write request (WRQ)
3	Data (DATA)
4	Acknowledgment (ACK)
5	Error (ERROR)

The TFTP header of a packet contains the opcode associated with that packet.

RRQ and WRQ packets (opcodes 1 and 2 respectively) have the format shown in the Figure. The file name is a sequence of bytes in netascii terminated by a zero byte. The mode field contains the string "netascii", "octet", or "mail" (or any combination of upper and lower case, such as "NETASCII", NetAscii", etc.) in netascii indicating the three modes defined in the protocol. A host which receives netascii mode data must translate the data to its own format.

Octet mode is used to transfer a file that is in the 8-bit format of the machine from which the file is being transferred. It is assumed that each type of machine has a single 8-bit format that is more common, and that that format is chosen. For example, on a DEC-20, a 36 bit machine, this is four 8-bit bytes to a word with four bits of breakage. If a host receives a octet file and then returns it, the returned file must be identical to the original.

Mail mode uses the name of a mail recipient in place of a file and must begin with a WRQ. Otherwise it is identical to netascii mode. The mail recipient string should be of the form "username" or "username@hostname". If the second form is used, it allows the option of mail forwarding by a relay computer.

The discussion above assumes that both the sender and recipient are operating in the same mode, but there is no reason that this has to be the case. For example, one might build a storage server. There is no reason that such a machine needs to translate netascii into its own form of text. Rather, the sender might send files in netascii, but the storage server might simply store them without translation in 8-bit format. Another such situation is a problem that currently exists on DEC-20 systems. Neither netascii nor octet accesses all the bits in a word. One might create a special mode for such a machine which read all the bits in a word, but in which the receiver stored the information in 8-bit format. When such a file is retrieved from the storage site, it must be restored to its original form to be useful, so the reverse mode must also be implemented. The user site will have to remember some information to achieve this. In both of these examples, the request packets would specify octet mode to the foreign host, but the local host would be in some other mode. **No such machine or application specific modes have been specified in TFTP, but one would be compatible with this specification.**

It is also possible to define other modes for cooperating pairs of hosts, although this must be done with care. There is no requirement that any other hosts implement these. There is no central authority that will define these modes or assign them names.

Data is actually transferred in DATA packets depicted in the Figure. DATA packets (opcode = 3) have a block number and data field. The block numbers on data packets begin with one and increase by one for each new block of data. This restriction allows the program to use a single number to discriminate between new packets and duplicates. The data field is from zero to 512 bytes long. If it is 512 bytes long, the block is not the last block of data; **if it is from zero to 511 bytes long, it signals the end of the transfer**

All packets other than duplicate ACK's and those used for termination are acknowledged unless a timeout occurs . Sending a DATA packet is an acknowledgment for the first ACK packet of the previous DATA packet. The WRQ and DATA packets are acknowledged by ACK or ERROR packets, while RRQ and ACK packets are acknowledged by DATA or ERROR packets. The block number in an ACK echoes he block number of the DATA packet being acknowledged. **A WRQ is acknowledged with an ACK packet having a block number of zero.**

An ERROR packet can be the acknowledgment of any other type of packet. The error code is an integer indicating the nature of the error. A table of values and meanings is given in the appendix. (Note that several error codes have been added to this version of this document.) The error message is intended for human consumption, and should be in netascii. Like all other strings, it is terminated with a zero byte.

Normal Termination

The end of a transfer is marked by a DATA packet that contains between 0 and 511 bytes of data (i.e., Datagram length < 516). This packet is acknowledged by an ACK packet like all other DATA packets. The host acknowledging the final DATA packet may terminate its side of the connection on sending the final ACK. On the other hand,dallying is encouraged. This means that the host sending the final ACK will wait for a while before terminating in order to retransmit the final ACK if it has been lost. The acknowledger will know that the ACK has been lost if it receives the final DATA packet again. The host sending the last DATA must retransmit it until the packet is acknowledged or the sending host times out. If the response is an ACK, the transmission was completed successfully. If the sender of the data times out and is not prepared to retransmit any more, the transfer may still have been completed successfully, after which the acknowledger or network may have experienced a problem. It is also possible in this case that the transfer was unsuccessful. In any case, the connection has been closed.

Premature Termination

If a request cannot be granted, or some error occurs during the transfer, then an ERROR packet (opcode 5) is sent. This is only a courtesy since it will not be retransmitted or acknowledged, so it may never be received. Timeouts must also be used to detect errors.

Initial Connection Protocol for reading a file

1. Host A sends a "RRQ" to host B with source= A's TID, destination= 69.

2. Host B sends a "DATA" (with block number= 1) to host A with source= B's TID, destination= A's TID.

Error Codes

Value	**Meaning**
0	Not defined, see error message (if any).
1	File not found.
2	Access violation.
3	Disk full or allocation exceeded.
4	Illegal TFTP operation.
5	Unknown transfer ID.
6	File already exists.
7	No such user.

Source Port : Picked by originator of packet.

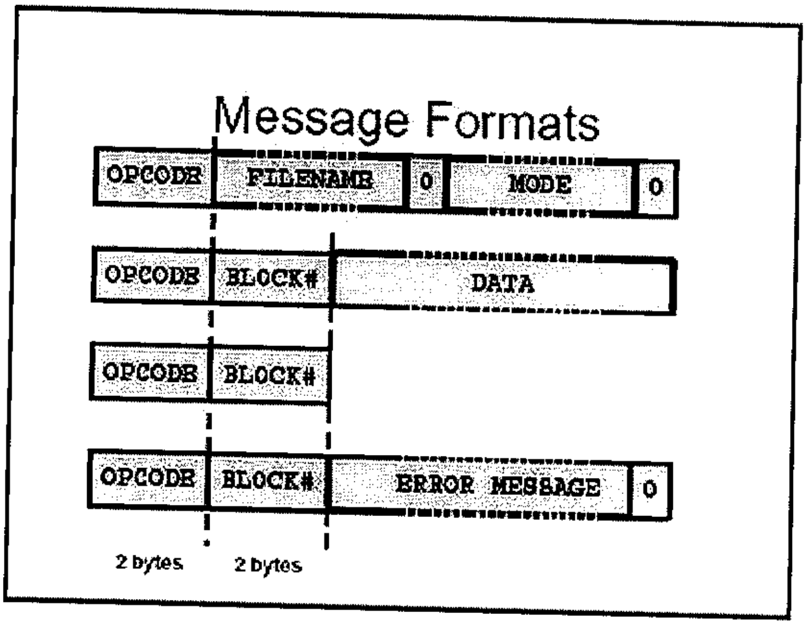
Dest. Port : Picked by destination machine (69 for RRQ or WRQ).

Length : Number of bytes in UDP packet, including UDP header.

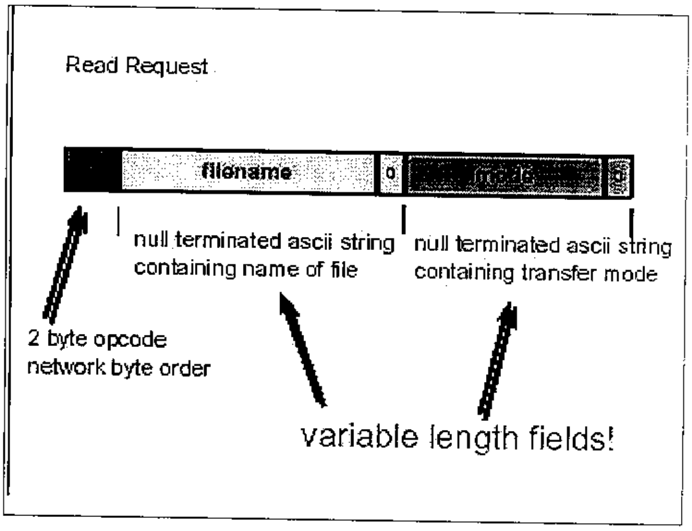
Checksum : (The implementor of this should be sure that the correct algorithm is used here.) . Field contains zero if unused.

Since TFTP includes no login or access control mechanisms, care must be taken in the rights granted to a TFTP server process so as not to violate the security of the server hosts file system. TFTP is often installed with controls such that only files that have public read access are available via TFTP and writing files via TFTP is disallowed.

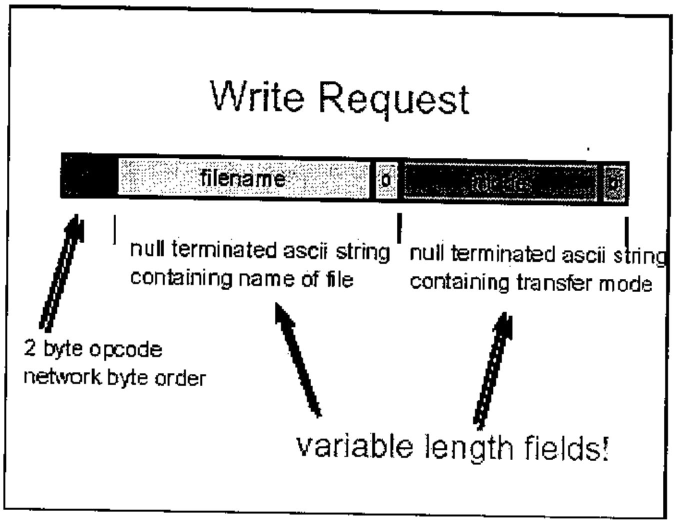
2.1.6 Protocol



Read Request


Read Request
filename
0
mode
0
null terminated ascii string containing name of file
null terminated ascii string containing transfer mode
2 byte opcode network byte order
variable length fields!

write request


Write Request
filename
0
mode
0
null terminated ascii string containing name of file
null terminated ascii string containing transfer mode
2 byte opcode network byte order
variable length fields!

Data formats

There are two data formats supported by TFTP :

- Ascii
- Binary

TFTP transfer modes

***"netascii"** : for transferring text files.*

1. all lines end with \r\n (CR,LF).
2. provides standard format for transferring text files.
3. both ends responsible for converting to/from netascii format.

***"octet"** : for transferring binary files.*

no translation done.

. receiving a file

you need to remove '**\r**' before storing data.

sending a file

you need to replace every '**\n**' with "**\r\n**" before sending

2.1.7 Connections :

1. The client BINDS the local address that forms a unique socket - 3 tuple(protocol , local address and local process) on the client's system.

2. The client sends it's first datagram (an RRQ or a WRQ) to the server at the servers well-known port. This datagram consist of the client's address so that server can send response.

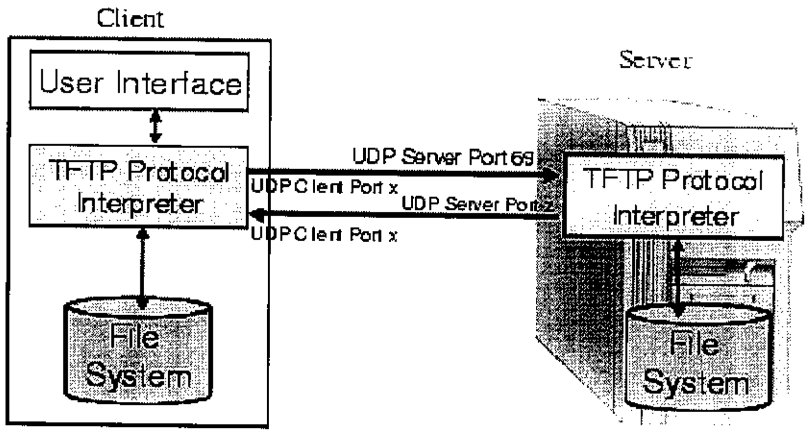
3. The server receives the datagram from the new client and spawns a new child process to handle the request. This child process BINDS an unique address.

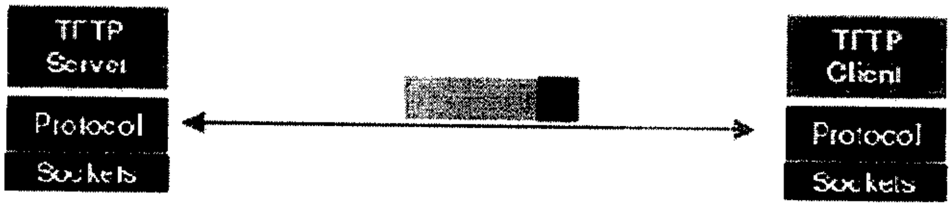
4. The child process sends the response to the RRQ or WRQ packets that the client sent. This datagram is send to the client's address from step 1 above. The return address in the datagram is the new port that the child process has obtained in step 3

5. The client and server continue exchanging packets , with client sending to the server.

2.2 SYSTEM MODEL

TFTP Connections





TFTP Transfer example:

1. Client sends read request. (RRQ)

2. Server responds with data packet of 512 bytes length (if file exists) with block number „1"

3. Client sends Acknowledgement for block number „1"

4. Server responds with next data packet.

5. ...

6. When client receives data packet with less than 512 bytes of data, it knows that this is the final packet.

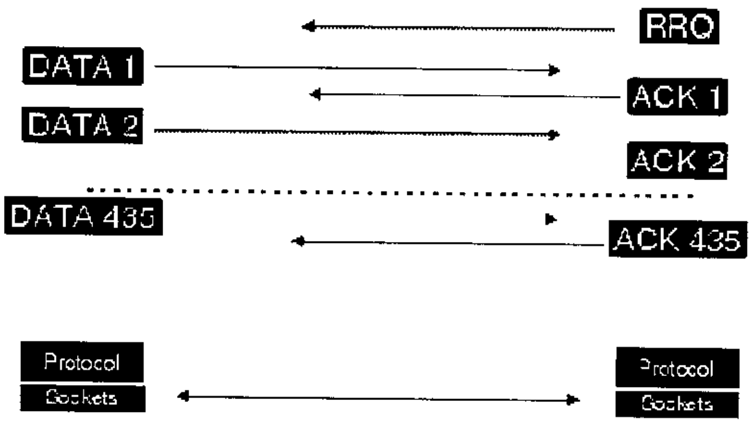
2.3 Proposed System:

As mentioned TFTP is designed to be implemented on top of the Datagram protocol. Since Datagram is implemented on the Internet protocol,packets will have an Internet header, a Datagram header, and a TFTP header. Additionally, the packets may have a header (LNI, ARPA header, etc.) to allow them through the local transport medium. The order of the contents of a packet will be: local medium header, if used, Internet header, Datagram header, TFTP header, followed by the remainder of the TFTP packet. (This may or may not be data depending on the type of packet as specified in the TFTP header.) TFTP does not specify any of the values in the Internet header. On the other hand, the source and destination port fields of the Datagram header (its format is given in the appendix) are used by TFTP and the length field reflects the size of the TFTP packet.

The proposed system is one in which the TFTP is implemented on top of the TCP. This results in a connection oriented service. As a result the synchronization between the client and the server is improved. Also the existing system has no user identification. As a result only publicly accessible files can be transferred. But the proposed system includes ways to improve the situation.

TFTP: Protocol Procedures

- Stop and wait data transfer (a block at a time)
- Recover from packet loss: ACK + time-out



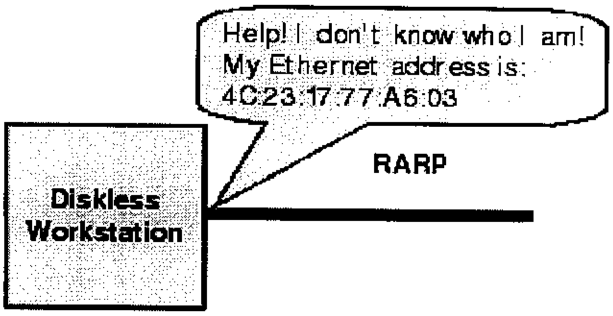
. The new system is one that could transmit and receive multiple configuration files at the same time. The currently available TFTP servers do not satisfy our requirements. A couple would allow multiple connections, but no more than two and none would transmit AND receive at the same time.

Note: A TFTP Server is NOT an FTP server. TFTP and FTP are different protocols. You will not be able to connect to the TFTP Server with an FTP client.

- Transfer files between systems.
- Minimal Overhead.
- Design for UDP as well as TCP
- Very efficient and compact. Possible to include in firmware itself.
- Used to bootstrap workstations and network devices.

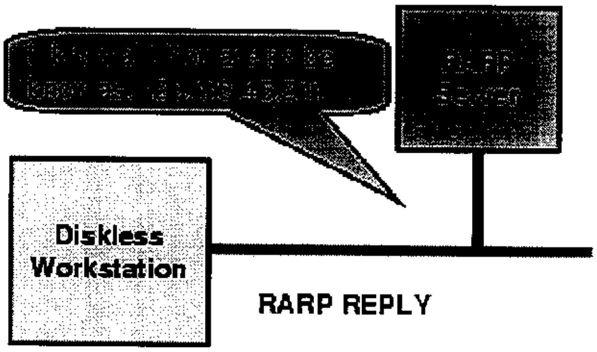
Diskless Workstation Booting 1

The call for help


Help! I don't know who I am! My Ethernet address is: 4C:23:17:77:A6:03
Diskless Workstation
RARP

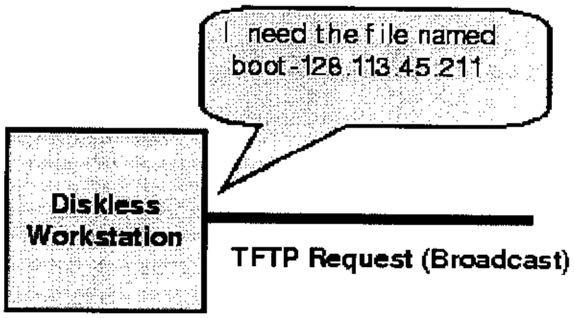
Diskless Workstation Booting 2

The answer from the all-knowing



Diskless Workstation Booting 3

The request for instructions



RARP is Reverse Address Resolution Protocol.:

If we have an Ethernet LAN consisting of hosts using TCP/IP protocols, we have 2 types of addresses: 32 bit Internet addresses and 48-bit Ethernet addresses (typically assigned by the manufacturer of the interface board and are all unique). We have the following address resolution problems.

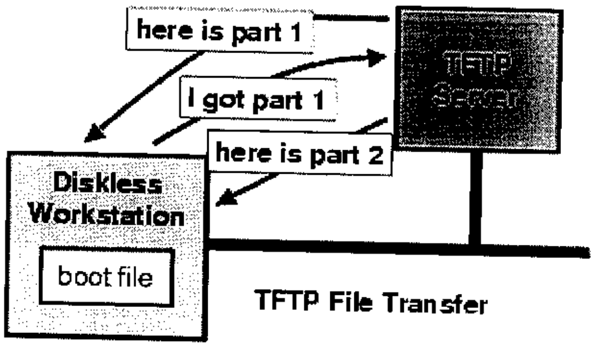
- If we know the Internet address of the other host that we want to communicate with, how does the IP layer determine which Ethernet address corresponds to that host? This is the *address resolution problem.*

- When a diskless workstation is initialized (bootstrapped), the OS can usually determine its own 48-bit Ethernet address from its interface hardware. But we do not want to embed the workstation's 32-bit Internet address into the OS image, as this prevents us from using the same image for multiple workstations. How can the diskless workstation determine its Internet address at bootstrap time ? This is the *reverse address resolution problem*

ARP solves the former while RARP solves the latter. RARP is intended for LANs with diskless workstations. One or more systems on the LAN are the RARP servers and contain the 32-bit Internet address and its corresponding 48-bit Ethernet address for each workstation. This allows the Os for each workstation to be generated without having to have its internet address as part of its configuration. When the workstation is initialized it obtains its 48-bit Ethernet address from the interface hardware and broadcasts an Ethernet RARP packet containing its Ethernet address and asking for its Internet address. Every host on the Ethernet LAN receives this broadcast, but only the RARP servers should respond.

Diskless Workstation Booting 4

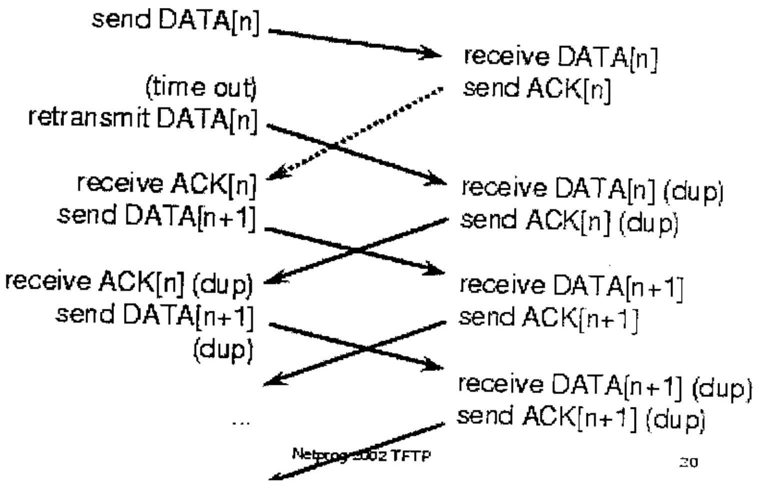
The dialog



LOST DATA PACKETS - ORIGINAL PROTOCOL SPECIFICATION

- ❖ Sender (Client or server) uses a time out with retransmission.
- ❖ Duplicate data packets must be recognized and ACK retransmitted.
- ❖ This original protocol suffers from "Sorcerer's Apprentice Syndrome".

Sorcerer's Apprentice Syndrome



THE FIX:

- ❖ Sender should not resend a data packet in response to duplicate ACK.
- ❖ Is sender receives ACK[n] - don't send DATA[n+1} if the ACK was a duplicate.

Process Model

The process model is concerned with establishing connection between systems. The connection-oriented protocol in the transport layer i.e. TCP is used for communication between systems. The fundamental mechanism for the communication between a client and server is shown in the following figure.

Connection oriented (TCP)

One process (server) makes its socket known to the system using ``bind". This will allow other sockets to find it. It then ``listens" on this socket to ``accept" any incoming messages. The other process (client) establishes a network connection to it, and then the two exchange messages. As many messages as needed may be sent along this channel, in either direction.

Server Model

There are two types of Server. They are Concurrent server and Iterative server.

Concurrent Server: A concurrent server invokes another process to handle each client request so that the original server process can go back to sleep, waiting for the next client request. Naturally, this type of servers requires an operating system that allows multiple processes to run at the same time. Most client requests that deal with a file of information (e.g., print a file, read or write a file) are handled in a concurrent fashion by the server. As the amount of the processing required handling each request depends on the size of the file.

Iterative Server: When a client's request can be handled by the server in a known, short amount of time, the server process handles the request by itself. We call this iterative server. A time-of-day service is typically handled in an iterative fashion by the server.

Interprocess Communication

Since network programming involves the interaction of two or more processes. we must look carefully at the different methods that are available for different processes to communicate with each other. In traditional single process programming different modules within the single process can communicate with each other using global variables, functions calls, and the arguments and results passed back and forth between functions and their callers. But when, dealing with separate processes, each executing within its own address space, there are more details to consider. When time-shared, multiprogramming operating systems were developed over 20 years ago, one design goal was to assure certain that separate processes wouldn't interfere with each other. For two processes to communicate with each other, they must both agree to it, and the operating system must provide some facilities for the Interprocess Communications (IPC).

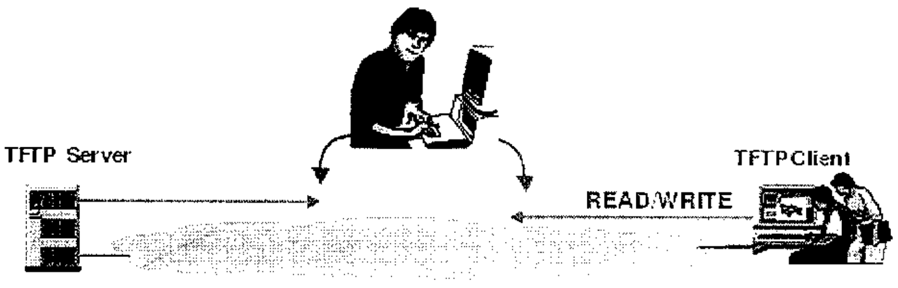
While some form of IPC is required for network programming, the use of IPC is by no means restricted to processes executing on different systems, communicating through a network of some form. Indeed, there are many instances where IPC can and should be used between processes on a single computer system

PERFORMANCE CONSIDERATIONS:

- **Transferring file blocks one at a time can be slow**
  - Move only small files.
  - Large time-outs = slow recovery from loss.
  - Small time-outs = duplicated copies.
- **Example: How long does it take to transfer a file of 1000 blocks?**
  - Assume round-trip latency is on average 5ms, time out is 25ms and 20% of the blocks are lost.
  - What is the average throughput as a function of round-trip latency, time-out, and loss probability.
- **The sorcerer's Apprentice syndrome:**
  - Protocol rules preserves duplicates.
  - Compute the impact of this syndrome on throughput.

Security Considerations

- Security risks
  - ❑ No authentication
  - ❑ Read: can be used to access sensitive files
  - ❑ Write: can be used to write sensitive files
- TFTP servers must limit access to files
  - ❑ TFTP directory must be restricted to include files that can be shared at no risk



CRITICAL ASSESSMENT:

The good news.

- Sockets enable simple and relatively uniform access to transport services.

The bad news.

- Distributed applications are too complex to build and maintain.
- Distributed applications require custom protocols.
- Protocol design requires bit-level encoding of PDUs.
- Design decisions can have significant non-trivial performance ramifications.
- Design decisions can lead to non-trivial security holes.

Chapter 3

REQUIREMENT ANALYSIS

The software requirement can be broadly classified into functional, hardware. software, resource and testing requirements.

3.1 Functional Requirements

Functional Requirement 1:

Introduction: Facility to connect to the remote server
Input: Host name and request for read or write .
Process: The server reply with an acknowledgement that it has accepted the connection
Output: The server responds with the corresponding packet based on the finite machine state transition.

Functional Requirement 2:

Introduction: Facility to request for a file.

Input: Input may be a local file name to be written to the server or a remote filename to be send to the client

Process: The server checks for the file access and process the request and send acknowledgement.

Output: The server responds with the corresponding packet based on the finite machine state transition.

Functional Requirement 3:

Introduction: Facility to check the status of the connection.

Input: The client socket id

Process: get the information about the status of the connection like whether it is connected and if so what mode it is connected(ASCII or binary) and also implements various other facilities like options to trace and set all warnings.

Output: Status message is displayed.

3.2 Hardware and Software Requirements

Hardware: Establishing a network requires RS232 cables, network hub, LAN cards, and two or more PCs. A UPS is needed for regulated power supply.

Software: Windows operating system has to be installed in the PCs used. Java Environment should be installed with the compiler and interpreter.

3.3 Resource Requirements

The resources such as Windows manual pages for system calls and DOS commands are needed for reference during implementation. As to satisfy the client's request, the server must be loaded with ample amount of applications.

3.4 Testing Requirements

Testing Requirements includes the following things:

- Three to four systems in a network
- The client programs are run on other systems.
- The authentication provided by the server is tested by requested various kinds of files.

Chapter 4

SYSTEM ANALYSIS

In the analysis phase, the project is divided into many stages or phases and each phase is carried out in a sequential fashion.

4.1 Constraints, Dependencies and Assumptions

The constraints and dependencies of the system are as follows

- .No User Authentication is done.
- Only Publicly accessible files can be retrieved.

The assumptions are as follows

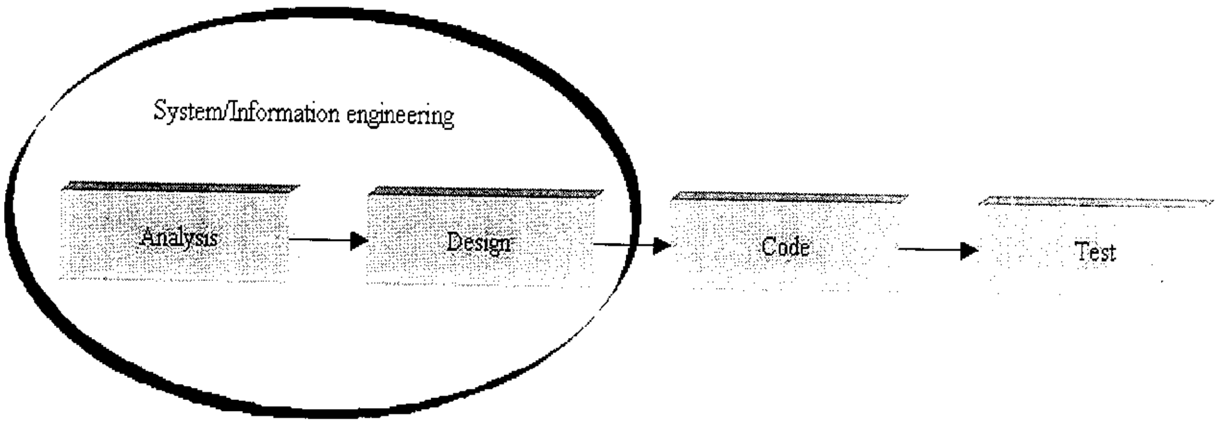
- The user is believed that he knows the syntax and commands for the file transfer.

4.2 Software Process Model

Linear Sequential Model

The software process model followed by us to develop the project is linear sequential model. Sometimes called the classic life cycle model or waterfall model, the linear sequential model suggests a systematic, sequential approach to software development that begins at the system level and progresses through analysis, design, coding, testing and support. The following figure illustrates the linear sequential model for software engineering.

The classic life cycle paradigm has a definite and simple structure. It provides a template into which methods for analysis, design, coding, testing, and support can be placed.



Chapter 5

SYSTEM ANALYSIS:

5.1 Design Constraints

Portability: The project has to be designed in such a way that it can be ported to other Windows systems.

Reliability: The project has to be designed in such a way that it works for all sorts of input from the system environment. For example, the server performs the searching for the local or remote file, which is requested by the client. If the request is not present in the server file system then failure message code has to be sent to the client. This ensures reliability in the file transfer.

Performance: The system performance is measured by its execution time and space occupied. Hence in order to ensure performance, the system has to be designed efficiently.

Scalability: Any number of clients can be added to or removed from the network without exceeding the network load. The scalability can be ensured when the system supports any number of clients without any significant decrease in performance. Also the presence of a concurrent server ensures that many clients can be handled at a time.

Standardization: The project has to be designed to follow the universal standards of TFTP. So RFC 783 standards of TFTP are considered for standardization. For example the TFTP server can send and receive files, to and from any other standard TFTP servers across network.

5.2 CLIENT USER INTERFACE DESIGN:

Serial Number	Command	Syntax	Meaning
1	Open	Open <port number>	Establishes connection with the port
2	Pwd	Pwd	Present Working Directory of the Remote System
3	Dir	Dir	Lists the directory contents of the Pwd
4	Get	Get<remote file><file name>	Retrieves a file from the remote system to the local host
5	Put	Put<local file><file name>	Writes file from the local host to the remote system
6	Mget	Mget *.*	Retrieves multiple files from remote system
7	Mput	Mput *.*	Writes multiple files to remote sytem
8	Send	Send<local file><file name>	Same as Put command
9	Recv	Recv<remote file><file name>	Same as Get command
10	Rename	Rename<file name>	Renames files in the remote system
11	Delete	Delete<file name>	Deletes the file in the remote system
12	Mdelete	Mdelete *.*	Deletes multiple files from the remote system
13	Cd	Cd<path>	Changes the directory in the remote system
14	Lcd	Lcd<path>	Changes the directory in the local host
15	Mkdir	Mkdir<name>	Creates directory in the remote system
16	Rmdir	Rmdir<name>	Removes directory from the remote system
17	Type	Type<mode>	Changes the mode of transfer
18	Close	Close	Command to quit

Chapter 6

SOURCE CODE

```
import java.io.*;

import java.net.*;

import java.util.*;

import java.awt.*;

public class tftp extends Thread

{

 static Frame f=new Frame();

static TextArea ta =new TextArea("");

 public static void main(String[] args)

    {

        f.setLayout(null);

         ta.setBounds(30,30,200,400);

         f.add(ta);

         f.setSize(500,500);

         f.show();

         if(args.length != 0) {

           System.out.println(args[0]);

             r = args[0];}

        else{ r = "f:\\";}

        int i = 1;
```

```
     try{
        ServerSocket s = new ServerSocket(21);


           for(;;)
           {
            Socket incoming = s.accept();
            new tftp(incoming,i).start();
            i++;
           }
       }
     catch(Exception e){}
}


public tftp(Socket income, int c)
   { incoming = income; counter = c; }




public void run()
{  int lng,lng1,lng2,i,ip1,ip2,ip = 1,h1;
  String a1,a2,di,str1,user="",host,dir;
  File f1=null;
  System.out.println(r);
  dir = r;
  InetAddress inet;
```

```
      InetAddress localip;
      try
       {
         inet = incoming.getInetAddress();
          ta.append("inet : " + inet.toString()+"\n");
         localip = inet.getLocalHost();
          ta.append("localip : " + localip.toString()+"\n");
         host = inet.toString();
          ta.append("host : " + host+"\n");
         h1 = host.indexOf("/");
         host = host.substring(h1 + 1);
         PrintWriter out = new PrintWriter(incoming.getOutputStream(),true);
         BufferedReader in = new BufferedReader(new
InputStreamReader(incoming.getInputStream()));
         out.println("220 MyTftp server.\r");


         boolean done = false;
         while(!done)
           {  a1 = "";
              a2 = "";
              String str = in.readLine();
              System.out.println("---------"+str);
      if(str.startsWith("RETR")){
                System.out.println("retr");
```

```
               out.println("150 Binary data connection");
               System.out.println(str);
               str = str.substring(4);
               str = str.trim();


               RandomAccessFile outFile = new
RandomAccessFile(dir+"\\"+str,"r");
                              System.out.println(dir+"\\"+str);
               Socket t = new Socket(host,ip);
               System.out.println("ip : " + ip);
               OutputStream out2
                 = t.getOutputStream();
               byte bb[] = new byte[1024];
               int amount;
              try{
               while((amount = outFile.read(bb)) != -1){
                                   System.out.println(new String(bb));
                  out2.write(bb, 0, amount);
                }
               out2.close();
               out.println("226 transfer complete");
               outFile.close();
               t.close();
              }
```

```
            catch(IOException e){}

        }

  if(str.startsWith("RNFR"))

 {

   String g=str.substring(5);

   f1=new File(dir+"\\"+g);

   if (f1.exists())

   {

    out.println("350 from File is present.");

   }

  else

  {

   out.println(" from File not exits.");

  }

 }


 if(str.startsWith("RNTO"))

 {

  String g1=str.substring(5);

  File f2=new File(dir+"\\"+g1);

  f1.renameTo(f2);

  out.println("530 File is renamed. \r\n");

 }

 if(str.startsWith("XMKD"))
 {
    String mkfile=str.substring(5);
    File f=new File(dir+"\\"+mkfile);
    if (f.exists())
    {
      out.println("Directory already exist");
   }else
      f.mkdir();
      out.println("Directory successfully Created");
}
if (str.startsWith("XRMD"))
{
   String removefile = str.substring(5);
   File f =new File(dir+"\\"+removefile);


   if (f.exists())
   {
          f.delete();
          out.println("Directory Successfully deleted");
  }else{
         out.println("Directory not found");
  }
```

```
}

if (str.startsWith("APPE"))

{

    System.out.println(str);

    out.println("150 Binary data connection");

    str = str.substring(4);

    str = str.trim();

    System.out.println(str);

    System.out.println(dir);




}

if(str.startsWith("STOR")){

        System.out.println("stor");

        out.println("150 Binary data connection");

        str = str.substring(4);

        str = str.trim();

        System.out.println(str);

        System.out.println(dir);

        RandomAccessFile inFile = new

        RandomAccessFile(dir+"\\"+str,"rw");

        Socket t = new Socket(host,ip);

        InputStream in2

          = t.getInputStream();
```

```
            byte bb[] = new byte[1024];
            int amount;
           try{
            while((amount = in2.read(bb)) != -1){
                inFile.write(bb, 0, amount);
             }
           in2.close();
           out.println("226 transfer complete");
           inFile.close();
           t.close();
          }
           catch(IOException e){}
       }



if(str.startsWith("TYPE")){
          out.println("200 type set");}
if(str.startsWith("DELE")){
          str = str.substring(4);
          str = str.trim();
          File f = new File(dir,str);
          boolean del = f.delete();
          out.println("250 delete command successful");}
```

```
if(str.startsWith("CDUP")){


          int n = dir.lastIndexOf("\\");
          dir = dir.substring(0,n);
          out.println("250 CWD command succesful");
                    }
if(str.startsWith("CWD")){
          str1 = str.substring(3);
          System.out.println(str1);
          if (str1.trim().equals(".."))
          {
           System.out.println("cd..");
           int lst=dir.lastIndexOf("\\");
           dir=dir.substring(0,lst);
           System.out.println(dir);
           out.println("250 CWD command succesful");
          }else{
          File f=new File(dir+"\\"+str1.trim());
          if (f.exists())
          {
               dir = dir+"\\"+str1.trim();
               System.out.println(dir);
               out.println("250 CWD command succesful");
          }else{
                out.println("250 Directory not Exist");
            }
          }


                    }
  if(str.startsWith("QUIT")) {
            out.println("GOOD BYE");
            done = true; }
  if(str.startsWith("USER")){
            user = str.substring(4);
            user = user.trim();
            if (user.equals("project"))
                  out.println("331 E-mail ID");
            else{
               out.println("Invalid password");
               done=true;
            }


}
if(str.startsWith("PASS")) out.println("230 User "+user+" logged in.");
if(str.startsWith("XPWD")){
          out.println("257 \""+dir+"\" is current directory");
 }
if(str.startsWith("SYS"))  out.println("500 SYS not understood");
if(str.startsWith("PORT")) {
          System.out.println("port");
          out.println("200 PORT command successful");


          lng = str.length() - 1;
          lng2 = str.lastIndexOf(",");
          lng1 = str.lastIndexOf(",",lng2-1);
          for(i=lng1+1;i<lng2;i++){
               a1 = a1 + str.charAt(i);}
          for(i=lng2+1;i<=lng;i++){
               a2 = a2 + str.charAt(i); }
          ip1 = Integer.parseInt(a1);
          ip2 =Integer.parseInt(a2);
          System.out.println(ip1+"   "+ip2);
          ip = ip1 * 16 *16 + ip2;
          System.out.println(ip);


                      }
if (str.startsWith("NLST"))
{
  System.out.println("Inside nlst");
  try
  {
        out.println("150 opening ASCII mode data connection");
```

```
Socket t = new Socket(host,ip);

PrintWriter out2 = new PrintWriter(t.getOutputStream(),true);

System.out.println(str);

str = str.substring(4);

str = str.trim();

str=str.substring(2,str.length());


System.out.println(str);

File f = new File(dir);

FilenameFilter filter=new onlyExt(str);

String d,e;

int i1,j1;

String a[] = f.list(filter);

j1 = a.length;

d = f.getName();

System.out.println(d);

for(i1=0;i1<j1;i1++)

{

        File fil1=new File(a[i1]);

        out2.println(a[i1]);

}

t.close();

out.println("226 Transfer complete");
```

```
  }
  catch(Exception e)
  {
     System.out.println(e);
   }


}
if(str.startsWith("LIST")) {
         System.out.println("list");
          try
          {  out.println("150 opening ASCII mode data connection");
             Socket t = new Socket(host,ip);
             PrintWriter out2
               = new PrintWriter(t.getOutputStream(),true);
             System.out.println(dir);
             File f = new File(dir);
             String[] a = new String[100];
             String d,e;
             int i1,j1;
             a = f.list();
             j1 = a.length;
             d = f.getName();


             System.out.println(d);
```

```
for(i1=0;i1<j1;i1++)
{
 String property="";
                    File file1 = new File(a[i1]);
 if (file1.isDirectory())
     property += "d";
 else
     property += "-";


 if (file1.canWrite())
       property += "w";
 else
     property += "-";


 if (file1.canRead())
     property += "rx";
 else
     property += "-";


 System.out.println(property);
 long len=file1.length() ;
 property +="    " +len+"    ";
```

```
                    out2.println(property+a[i1]);
                     }
                    t.close();
                    out.println("226 Transfer complete");
                    }
                    catch(IOException e){}
                          }


         }
         incoming.close();
        }
        catch (Exception e)
        {
          System.out.println(e);
        }
      }
private Socket incoming;
private int counter;
private static String r;
}


class onlyExt implements FilenameFilter
{
                  String ext;
```

```
            public onlyExt(String ext)

            {

                    this.ext="."+ext;

            }

            public boolean accept(File dir ,String name)

            {

                    return name.endsWith(ext);

            }

}
```

Chapter 7

IMPLEMENTATION ISSUES

The inbuilt data structures are used in finding the required information such as file permissions, file ownership, network address i.e. IP address of the host system, etc. These data structures are the input or output of system calls used. Some of the data structures used are as explained below.

Data Structures

The functions that convert IP address of a system to and from the ``dotted'' addresses as in 137.92.11.1 to 32 bit integer addresses are:

- unsigned long inet_addr (char * ptr)
- char *inet_ntoa (struct in_addr in)

The structure in_addr is used to store the IP address in network byte order. Its description is given below:

```
struct in_addr {
   unsigned long int s_addr;
}
```

The BSD library provides some functions for finding host names.

- char * gethostname (char * name, int size) - finds the ordinary host name.
- struct hostent * gethostbyname (char * name) - returns a pointer to a structure with two important fields: "char * h_name" which is the "official" network name of the host and "char * * h_addr_list" which is a list of TCP/IP addresses.

The structure hostent is used to store details about host information such as host name, list of IP addresses, etc. Its description is given below.

```
struct hostent {
  char   * h_name;       /* official name of host */
  char   * * h_aliases;  /* alias list             */
  int      h_addrtype;   /* host address type      */
  int      h_length;     /* length of address      */
  char   * * h_addr_list; /* list of addresses     */
}
```

To handle byte ordering for non-standard size integers, there are conversion functions

- htonl - host to network, long int
- htons - host to network, short int
- ntohl - network to host, long int
- ntohs - network to host, short int

The address of an IP service given is using a structure

```
struct sockaddr_in {
 short   sin_family;
 u_short sin_port;
 struct  in_addr sin_addr;
 char    sin_zero[8];}
```

Creating sockets and binding it to the application using it establish the communication between systems. There are system calls for creating a socket, binding it to a particular physical port, making it to listen to a port, accepting connection request from remote systems and connecting it to other remote systems. The list of system calls for the above-mentioned services are as follows:

- int socket (int family, int type, int protocol);
- int bind (int s, struct sockaddr * name, int namelen);
- int listen (int s, int backlog);
- int accept (int sockfd, struct sockaddr * name, int * namelen);
- int connect (int s, struct sockaddr * name, int namelen);

Two additional data transfer library calls, namely send() and recv(), are available if the sockets are connected. They correspond very closely to the read() and write() functions used for I/O on ordinary file descriptors.

- int send (int sd, char * buf, int len, int flags);
- int recv (int sd, char * buf, int len, int flags);

In both cases, sd is the socket descriptor.

Chapter 8

TESTING TECHNIQUES

Good software testing ensures the reliability of the software. The testing techniques that were followed when testing this software are as follows:

Software was tested from two different perspectives. They were:

- Internal program logic was exercised using "white-box" test case design techniques.
- Software requirements were exercised using "black-box" test case design techniques.

When the testing phase of the project began, the point of view on the software had been changed to "break" the software design test cases in a disciplined fashion and review the test cases that were created for thoroughness. White-box and black box testing was done to test the software.

8.1 White-Box Testing

Basis path testing, a widely used white-box testing, was followed as the testing technique. The steps that include when white box testing was done were as follows:

- All independent paths within a module were exercised once. Searching a user's login details in password file, checking the authentication matches for logging in, sending messages or data across network, displaying the passed message or data in user readable form, servicing more number of clients at a time by server, etc. were some of the independent paths present within the system.
- All logical decisions on their true and false sides were exercised.

- All loops present were exercised at their boundary conditions and also within their operational bounds.
- All internal data structures were exercised to ensure their validity.

8.2 Black-Box Testing

The black box testing was focused on functional requirements of the software. The steps that include when black box testing was done were as follows:

- Validity of all functions used was tested.
- System behavior and its performance were tested.
- Classes of input that make good test cases were determined.
- Boundaries of input data to functions were determined.
- Sensitive input values for the functions involved were determined.

The boundary of input data to the system calls like *"send"*, *"recv"* in functions like Net_send, Net_recv etc (socket-descriptor) must lie in positive number range. It is usually greater than 0. Providing a negative value for socket-descriptor to the above function tested this and the result was examined. The system produced error message indicating negative value for socket descriptor. Like this, the valid input range for every function was determined.

Chapter 9

TESTING STRATEGY

The testing strategy includes performing unit testing followed by system testing. The system testing includes security testing and stress testing.

9.1 Unit Testing

In unit testing, the control flow is mainly exercised and verified that all statements have been executed at least once. Providing various test cases has tested the interfaces between the functions present in different modules.

9.2 System Testing

9.2.1 Security testing

This testing attempts to verify the protection mechanisms built into a system will protect it from improper penetration. The user logging in is tested with protected password. This password file is checked for any possible information leak when opening in text editors. The possibilities of hacking the password is ruled out as there is no option like forget password. And trying the possible combinations of six characters can only do it. Only the server can delete any user accounts, send reminding mails to all its users etc are checked. The security of mails of one user is checked by trying to open it through any other means from client through the network.

9.2.2 Stress Testing

Stress testing executes a system in a manner that demands resource in abnormal quantity, frequency, or volume. The stress testing is performed on this system by making the software to demand more memory space. Space required to build message files is specified as a system environment variable. Around twenty users accounts were created. And an average of twenty-five messages for each user is send and tested. The outcome of this testing is that there can't be more than twenty-seven users for a password file of length sixty-one. This provides more stress on the POP server and thus limits the system from requesting users as the password file is filled. This is used as the upper limit for number of uses and can be increased by enlarging the size of password file. Increasing the key range of hashing function does this.

The server responds with OACK (option acknowledgement) of the format

Opcode	Opt 1	Val 1	....	Opt N	Val N

Thus the facilities or various features can be pre-determined and set accordingly before any block of data is being transferred between the two communicating entity.

Otherwise an error message is returned to restart the initiating process.

MULTICAST OPTION

Often when similar computers are booting remotely they will each download the same image file. By adding multicast into the TFTP option set, two or more computers can download a file concurrently, thus increasing network efficiency. This multicast can be implemented by giving a read request as :

Opcode	Filename	Null	Mode	Null	Multicast	Null

Thus multicast option is being specified in the read request.and server responds by OACK (Option Acknowledgement)

Opcode	Multicast	Null	Addr,port,mc	Null

Opc

The opcode field contains the number 6, for Option Acknowledgment.

Multicast

Acknowledges the multicast option. This is a NULL-terminated field.

Addr

The addr field contains the multicast IP address. This field is terminated with a comma.

CONCLUSION:

Thus the Trivial file Transfer protocol was designed and implemented. using the principles of Software Engineering. Although the protocol specification specifies only five commands ie get, put, send, recv, error we have enhanced our project to include another thirteen more commands to make the project more interactive and a useful package for future uses. The program transfers files successfully between systems on a network enabled with an Ethernet card and all the commands were executed successfully. To provide more user interactive ness the output in the Dos prompt has been brought into the Awt screen.

REFERENCES

- Sollins, K., "The TFTP Protocol (Revision 2)", STD 33, RFC 1350, MIT, July 1992.
- Richard Stevens, Unix Network Programming .
- USA Standard Code for Information Interchange, USASI X3.4-1968
- Postel, J., "User Datagram Protocol," RFC 768, USC/Information Sciences Institute, 28 August 1980
- Postel, J., "Telnet Protocol Specification," RFC 764, USC/Information Sciences Institute, June, 1980.