EMERTXE TRAINING PROJECT DOCUMENTATION FRAMEWORK

**REQUIREMENTS & DESIGN DOCUMENT**

**Module – Linux Internals**

# TFTP Protocol

EMERTXE

# Contents

# 1 Abstract

Trivial File Transfer Protocol (TFTP) is a simple, File Transfer Protocol which allows a client to get from or put a file onto a remote host. One of its primary uses is in the early stages of nodes booting from a local area network. TFTP has been used for this application because it is very simple to implement.

This protocol doesn’t support advanced features that is there in FTP. User authentication, directory-listing features are not supported by this protocol. This implementation is done with simplicity and user friendliness as main focus.

TFTP was first standardized in 1981 and the current specification for the protocol can be found in RFC 1350

In this protocol three modes of transfer are currently supported: netascii (This is ascii as defined in "USA Standard Code for Information Interchange" with the modifications specified in "Telnet Protocol Specification" .) Note that it is 8 bit ascii. The term "netascii" will be used throughout this document to mean this particular version of ascii.); octet (This replaces the "binary" mode of previous versions of this document.) raw 8 bit bytes; mail, netascii characters sent to a user rather than a file. (The mail mode is obsolete and should not be implemented or used.) Additional modes can be defined by pairs of cooperating hosts.

# 2 Requirements

- Refer RFC 1350
- The server runs infinitely in a loop .The client runs till the user types “bye” in the prompt. Once a new process is forked the new port where the server is waiting should be informed to the client. That is automatically done when the server responds to the client’s request. In this implementation the new port starts from 20000 and for each connection the port number is increased. By doing this the port numbers are unique and the clients will have unique port numbers for contacting the server.

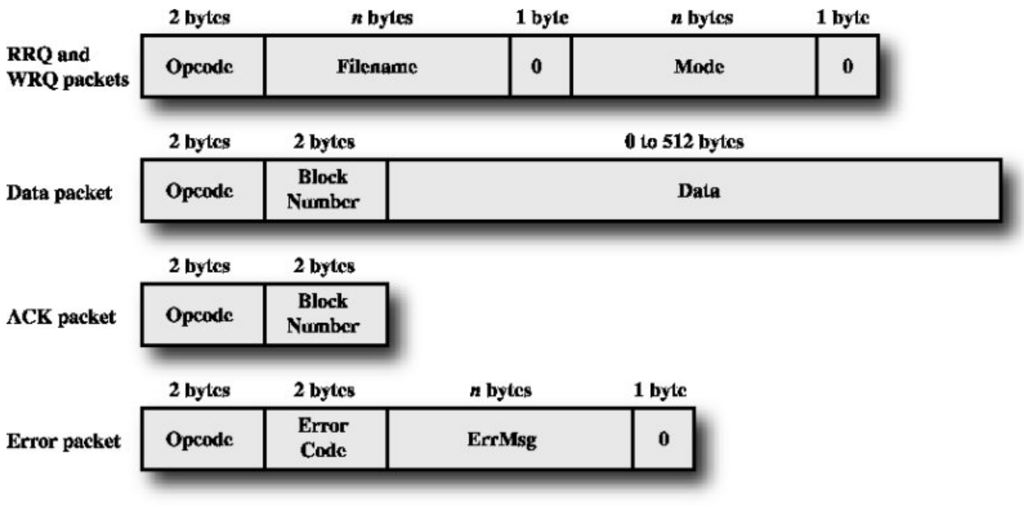
# 3 Prerequisites

- Socket Programming

# 4 Design

**Protocol Description and Header Formats:**

The client initially sends the Request for read (RRQ) or Write (WRQ). Once the request is send the one side sends the data and another side sends the Acknowledgment. So in the implementation also these functions are made common for both client and server for code re-usability. When the data is sent from one entity to another one the data is read from the file in 512 bytes chunks. With Each data packet a unique sequence number is chosen in order to keep track of the transmission. Following diagram shows the different headers used for the TFTP.



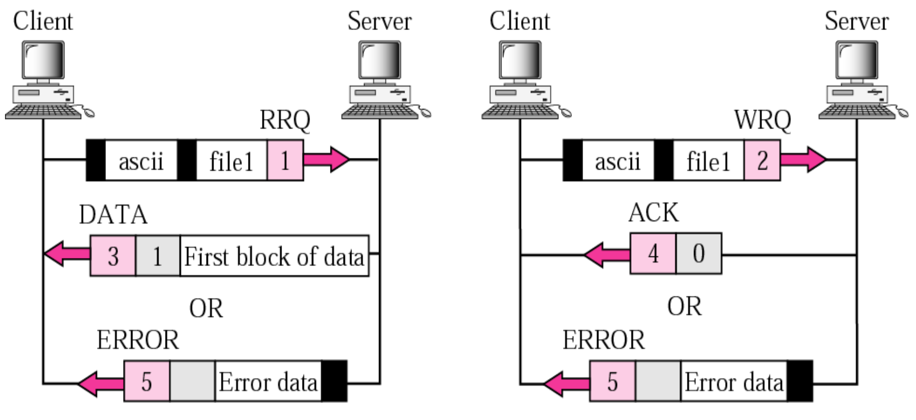
And the opcode for different packets are given below.

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

**Protocol Handshake**

When the request is sent from the client to the server, the handshake starts. When the server gets the request from the client it checks for errors. The errors are classified into read and write errors. The protocol, which happens initially, is called as Initial Connection Protocol. After that protocol is completed the data transmission starts between two entities.

The following diagram shows the initial connections for reading and writing.



a. Connection for reading    b. Connection for writing

Following are the common error messages, which is supported in the TFTP implementation.

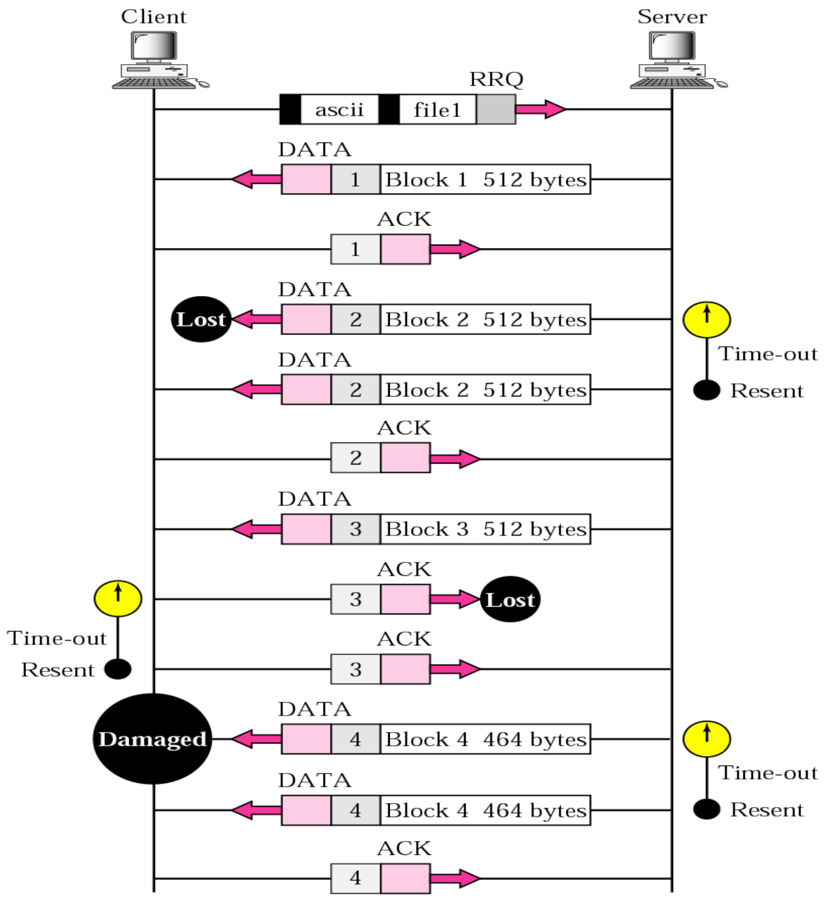
**Error Codes:**

| **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 | Illegal TFTP operation |
| 6 | Unknown transfer ID. |
| 7 | File already exists. |

**File transfer**

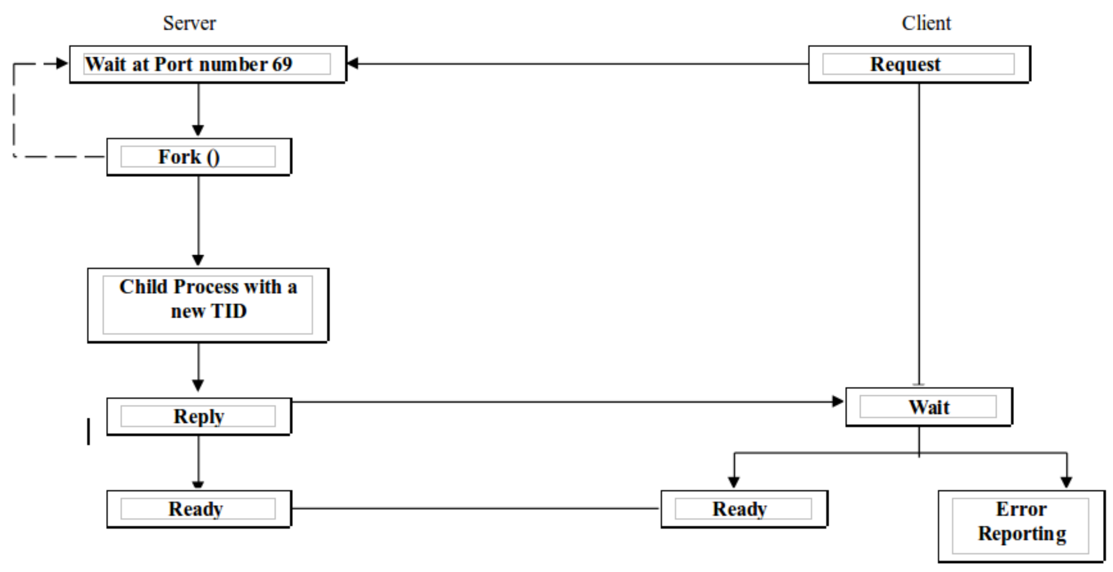
Once the error checking is done and no errors are found the client is notified about that using appropriate Headers. If a packet gets lost in the network, the intended recipient will timeout and may re-transmit his last packet (which may be data or an acknowledgment), thus causing the sender of the lost packet to re-transmit that lost packet. The sender has to keep just one packet on hand for re-transmission, 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.

Finally when the data transmission ends the connection is closed gracefully. Following diagram shows the handshake that occurs between the client and the server (For both read and write operations).



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.

## Flow diagrams



# 5 Sample Output

Server must run in background without any user interface. Client should provide a prompt to user (like default tftp client). User can use commands to connect to server, get/put file and exit.

Following are the commands:

1. connect → To send connect request to server (ip-address)
2. get → To receive a file from server (file name)
3. put → To send a file to server (file name)
4. mode → To set mode of transfer (octet, netascii or mail).
5. bye / quit → close the connections and exit from client.

```
user@emertxe] ./tftp_server &
user@emertxe]
```

*Fig 5 1: Server running in background*

```
user@emertxe] ./tftp_client
Type help to get supported features
tftp] help
connet <server-ip>      : Connect to server
get <file-name>         : Receive a file from server
put <file-name>         : Send a file to the server
mode                    : Transfer mode(s) (octet, netascii or mail)
bye / quit              : Exit the application
tftp] quit
```

*Fig 5 2: Application usage. Help menu*

```
user@emertxe] ls
tftp_client.c   tftp_client.h   tftp_client     README
user@emertxe] ./tftp_client
Type help to get supported features
tftp] connect 127.0.0.1
INFO: Connected to server
tftp] put README
INFO: File: README
INFO: Transfer Mode: netascii
#####
INFO: File transfer successful
tftp] quit
user@emertxe]
```

*Fig 5 3: Establishing the connection with server. File transfer to Server using put command*

```
user@emertxe] ls
tftp_client.c   tftp_client.h   tftp_client     README
user@emertxe] dmesg > /var/log/tftpboot/syslog
user@emertxe] ls /var/lib/tftpboot/
syslog
user@emertxe] ./tftp_client 127.0.0.1
Type help to get supported features
INFO: Connected to server
tftp] get syslog
INFO: File: syslog
INFO: Transfer Mode: netascii
#####
INFO: File receive successful
tftp] bye
user@emertxe] ls
syslog  tftp_client.c   tftp_client.h   tftp_client     README
user@emertxe]
```

*Fig 5 4: Creating a dummy file in server. Establishing the connection with server via command line argument. Receiving the file from the server*

```
user@emertxe] ./tftp_client
Type help to get supported features
tftp] get syslog
ERROR: Not connected to server
tftp] connect 127.0.0.1
INFO: Connected to server
tftp] put README
INFO: File: README
INFO: Transfer Mode: netascii
ERROR: File already exist
tftp] get log
INFO: File: syslog
INFO: Transfer Mode: netascii
ERROR: File not found
tftp] tata
Invalid command
tftp] bye
user@emertxe]
```

*Fig 5 5: Some possible Error Handling*

# 6 Artifacts

## Skeleton Code

- www.emertxe.com/content/linux-internals/code/TFTP_src.zip

## References

- RFC 1350
- http://techgenix.com/understanding-tftp-protocol/
- http://www.tcpipguide.com/free/t_TrivialFileTransferProtocolTFTP.htm
- https://www.rfc-editor.org/rfc/pdfrfc/rfc1350.txt.pdf
- http://citeseerx.ist.psu.edu/viewdoc/downloadjsessionid=B0AA73C24AAA4308A818E331E6213D0C?doi=10.1.1.300.2&rep=rep1&type=pdf