Network Lab Manual

Department of Computer Science and Engineering
College of Engineering Trivandrum
Thiruvananthapuram

Contents

1 Basics of Network configurations files and Networking Commands

Aim :

Getting started with Basics of Network configurations files and Networking Commands in Linux.

Basic Commands :

ifconfig

- ifconfig(interface configurator) command is use to initialize an interface, assign IP Address to interface and enable or disable interface on demand.
- With this command you can view IP Address and Hardware / MAC address assign to interface and also MTU(Maximum transmission unit) size.

ping

- PING(Packet INternet Groper) command is the best way to test con- nectivity between two nodes, whether it is Local Area Network(LAN) or Wide Area Network (WAN).
- Ping use ICMP (Internet Control Message Protocol) to communicate to other devices. You can ping host name of ip address using below command.
- In Linux ping command keep executing until you interrupt. Ping with -c option exit after N number of request.

traceroute

- traceroute is a network troubleshooting utility which shows number of hops taken to reach destination also determine packets traveling path.

netstat

- Netstat(Network Statistic) command displays connection info, routing table information etc.
- To display routing table information use option as -r.

nslookup

- nslookup is a command-line administrative tool for testing and troubleshooting DNS servers (Domain Name Server).
- Most operating systems comes with built-in nslookup feature.

route

- route command shows and manipulates ip routing table.
- It can add and delete routes and default Gateway.

dig

- Dig (Domain Information Groper) query DNS related information like a record, CNAME, MX Record etc. This command mainly use to troubleshoot DNS related query.

arp

- ARP(Address Resolution Protocol) is useful to view or add the contents of the kernel's ARP tables.

host

- Host command to find name to IP or IP to name in IPv4 or IPv6 and also query DNS records.

hostname

- hostname is to identify in a network. Execute hostname command to see the hostname of your box.

ethtool

- ethtool is a replacement of mii-tool. It is to view, setting speed and duplex of your Network Interface Card (NIC).

2 System Calls

Aim :

To familiarize and understand the use and functioning of System Calls used for Operating system and network programming in Linux.

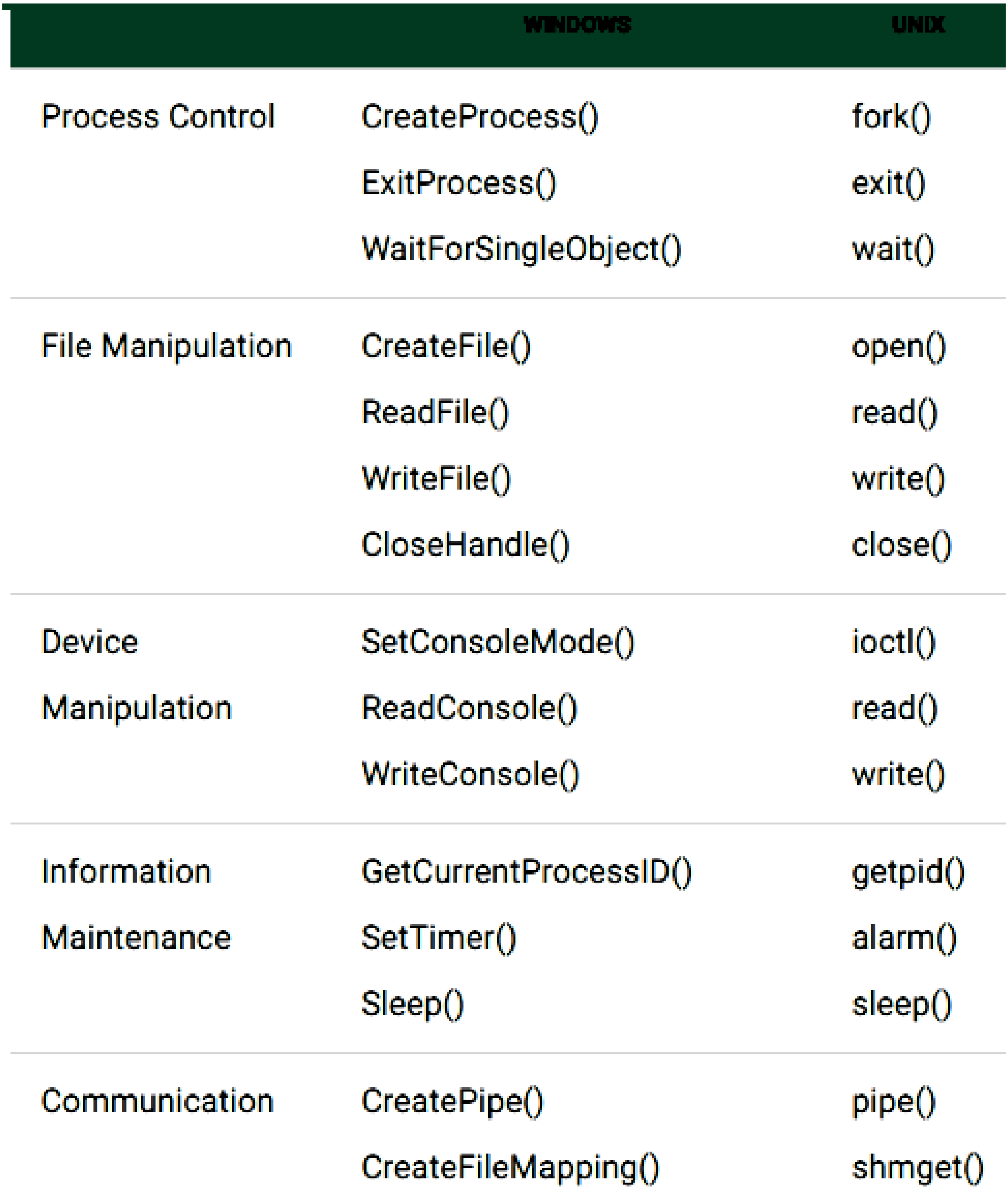
System Call

In computing, a system call is the programmatic way in which a computer program requests a service from the kernel of the operating system it is executed on. ... It provides an interface between a process and operating system to allow user-level processes to request services of the operating system.

Types of System Calls

- Process control
- File management
- Device management
- Information maintenance.
- Communications

Example



	WINDOWS	UNIX
Process Control	CreateProcess()	fork()
	ExitProcess()	exit()
	WaitForSingleObject()	wait()
File Manipulation	CreateFile()	open()
	ReadFile()	read()
	WriteFile()	write()
	CloseHandle()	close()
Device Manipulation	SetConsoleMode()	ioctl()
	ReadConsole()	read()
	WriteConsole()	write()
Information Maintenance	GetCurrentProcessID()	getpid()
	SetTimer()	alarm()
	Sleep()	sleep()
Communication	CreatePipe()	pipe()
	CreateFileMapping()	shmget()

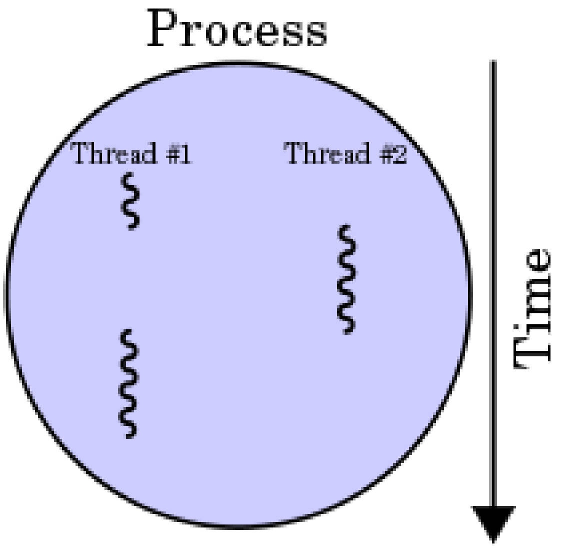
3 Process and thread

Aim :

Familiarization and implementation of programs related to Process and thread.

Thread

A thread is a path of execution within a process. A process can contain multiple threads. A thread is also known as lightweight process. The idea is to achieve parallelism by dividing a process into multiple threads. For example, in a browser, multiple tabs can be different threads. MS Word uses multiple threads: one thread to format the text, another thread to process inputs, etc. The primary difference is that threads within the same process run in a shared memory space, while processes run in separate memory spaces. Threads are not independent of one another like processes are, and as a result threads share with other threads their code section, data section, and OS resources (like open files and signals). But, like process, a thread has its own program counter (PC), register set, and stack space.



4 First Readers-Writers Problem

Aim :

Implement the First Readers-Writers Problem.

Introduction

It is possible to protect the shared data behind a mutual exclusion mutex, in which case no two threads can access the data at the same time. However, this solution is suboptimal, because it is possible that a reader R1 might have the lock, and then another reader R2 requests access. It would be foolish for R2 to wait until R1 was done before starting its own read operation; instead, R2. This is the motivation for the first readers-writers problem, in which the constraint is added that no reader shall be kept waiting if the share is currently opened for reading. This is also called readers-preference

Algorithm 1 First Reader Writers Problem

```
semaphore resource=1;
semaphore rmutex=1;
readcount=0;
procedure WRITER
    resource.P();                    ▷ resource.P() is equivalent to wait(resource)
    <CRITICAL Section>
    <EXIT Section>
    resource.V();
end procedure
procedure READER
    rmutex.P();                      ▷ rmutex.P() is equivalent to wait(rmutex)
    <CRITICAL Section>
    readcount++;
    if readcount == 1 then
        resource.P();
    end if
    <EXIT CRITICAL Section>
    rmutex.V();                      ▷ resource.V() is equivalent to signal(resource)
    rmutex.P();
    <CRITICAL Section>
    readcount–;
    if readcount == 0) then
        resource.V();
    end if
    <EXIT CRITICAL Section>
    rmutex.V();                      ▷ rmutex.V() is equivalent to signal(rmutex)
end procedure
```

5 Second Readers-Writers Problem

Aim :

Implement the Second Readers-Writers problem.

Introduction

The first solution is suboptimal, because it is possible that a reader R1 might have the lock, a writer W be waiting for the lock, and then a reader R2 requests access. It would be unfair for R2 to jump in immediately, ahead of W; if that happened often enough, W would starve. Instead, W should start as soon as possible. This is the motivation for the second readers-writers problem, in which the constraint is added that no writer, once added to the queue, shall be kept waiting longer than absolutely necessary. This is also called writers-preference.

Algorithm 2 Second Reader Writers Problem

```
semaphore resource=1;
semaphore rmutex=1;
semaphore wmutex=1;
semaphore resource=1;
readcount=0;
writecount = 0;
procedure READER
    <ENTRY Section>
    readTry.P()
    rmutex.P()
    readcount++
    if readcount == 1 then
        resource.P()
    end if
    rmutex.V()
    readTry.V()
    <CRITICAL Section>
    <EXIT Section>
    rmutex.P()
    readcount–
    if readcount == 0 then
        resource.V()
    end if
    rmutex.V()
end procedure
```

```
procedure WRITER
    <ENTRY Section>
    w.mutex.P();
    writecount++
    if writecount == 1 then
        readtry.P()
    end if
    wmutex.V()
    <CRITICAL Section>
    resource.P()
    resource.V()
    <EXIT Section>
    writecount–;
    if writecount == 1 then
        readTry.V();
    end if
    wmutex.V();
end procedure
```

6 PIPE, Message Queue and Shared Memory

Aim :

Implement programs for Inter Process Communication using PIPE, Message Queue and Shared Memory.

Theory :

Pipes

A pipe acts as a conduit allowing two processes to communicate. Pipes were one of the first IPC mechanisms in early UNIX systems. They typically provide one of the simpler ways for processes to communicate with one another, although they also have some limitations. Two common types of pipes used on both UNIX and Windows systems: ordinary pipes and named pipes.

Ordinary Pipes : Ordinary pipes allow two processes to communicate in standard producer consumer fashion: the producer writes to one end of the pipe (the write-end) and the consumer reads from the other end (the read-end). Ordinary pipes are unidirectional. If two-way communication is required, two pipes must be used, with each pipe sending data in a different direction.
On UNIX systems, ordinary pipes are constructed using the function
pipe(int fd[ ])
This function creates a pipe that is accessed through the int fd[] file descriptors: fd[0] is the read-end of the pipe, and fd[1] is the write-end. UNIX treats a pipe as a special type of file.

Named Pipes : Named pipes provide a much more powerful communication tool. Communication can be bidirectional, and no parent-child relationship is required. Once a named pipe is established, several processes can use it for communication. In fact, in a typical scenario, a named pipe has several writers. Additionally, named pipes continue to exist after communicating processes have finished.

Message Queues

The POSIX standard (IEEE Std 1003.1-2001) defines an IPC mechanism based on message queues, which is usually known as POSIX message queues. POSIX message queues sport a number of advantages over the older queues:

- A much simpler file-based interface to the applications
- Native support for message priorities

- Native support for asynchronous notification of message arrivals, either by means of signals or thread creation
- Timeouts for blocking send and receive operations

However, there are some fundamental differences between pipes and message queues:

- Message queues have internal structure. With a named pipe, a writer is just pumping bits. For a reader, there's no distinction between different calls to write() from different writers.
- Message queues are priority-driven.
- The programmer has control over the geometry of a message queue. When a message queue is created, the programmer may set ceilings on the number of messages that can be on the queue, and the size of each message.
- A process can determine the status of a message queue. One major drawback of pipes is that their state is unknown.

Shared Memory

Interprocess communication using shared memory requires communicating processes to establish a region of shared memory. Typically, a shared-memory region resides in the address space of the process creating the shared-memory segment. Other processes that wish to communicate using this shared-memory segment must attach it to their address space. Recall that, normally, the operating system tries to prevent one process from accessing another processs memory. Shared memory requires that two or more processes agree to remove this restriction. They can then exchange information by reading and writing data in the shared areas. The form of the data and the location are determined by these processes and are not under the operating systems control. The processes are also responsible for ensuring that they are not writing to the same location simultaneously.

7 Socket Programming : TCP

Aim :

Implement Client-Server communication using Socket Programming and TCP as transport layer protocol.*

TCP

TCP (Transmission Control Protocol) is a standard that defines how to establish and maintain a network conversation via which application programs can exchange data. TCP works with the Internet Protocol (IP), which defines how computers send packets of data to each other. Together, TCP and IP are the basic rules defining the Internet. TCP is defined by the Internet Engineering Task Force (IETF) in the Request for Comment (RFC) standards document number 793.

Client, Server and Socket

- Server-A server is a software that waits for client requests and serves or processes them accordingly.
- Client- a client is requester of this service. A client program request for some resources to the server and server responds to that request.
- Socket- Socket is the endpoint of a bidirectional communications channel between server and client. Sockets may communicate within a process, between processes on the same machine, or between processes on different machines. For any communication with a remote program, we have to connect through a socket port.

8 Socket Programming : UDP

Aim :

Implement Client-Server communication using Socket Programming and UDP as transport layer protocol.*

UDP

UDP (User Datagram Protocol) is an alternative communications protocol to Transmission Control Protocol (TCP) used primarily for establishing low-latency and loss-tolerating connections between applications on the internet. It is a process to process communication. It is unreliable.

Client, Server and Socket

- Server-A server is a software that waits for client requests and serves or processes them accordingly.
- Client- a client is requester of this service. A client program request for some resources to the server and server responds to that request.
- Socket- Socket is the endpoint of a bidirectional communications channel between server and client. Sockets may communicate within a process, between processes on the same machine, or between processes on different machines. For any communication with a remote program, we have to connect through a socket port.

9 Multi user chat server using TCP

Aim :

Implement a multi user chat server using TCP as transport layer protocol.

Theory :

Sockets can be thought of as endpoints in a communication channel. Here, we set up a socket on each end and allow a client to interact with other clients via the server. The socket on the server side associates itself with some hardware port on the server side. Any client that has a socket associated with the same port can communicate with the server socket.

Multi-Threading

A thread is sub process that runs a set of commands individually of any other thread. So, every time a user connects to the server, a separate thread is created for that user and communication from server to client takes place along individual threads based on socket objects created for the sake of identity of each client. We will require two scripts to establish the chat room. One to keep the serving running, and another that every client should run in order to connect to the server.

Server Side Script

The server side script will attempt to establish a socket and bind it to an IP address and port specified by the user. The script will then stay open and receive connection requests, and will append respective socket objects to a list to keep track of active connections. Every time a user connects, a separate thread will be created for that user. In each thread, the server awaits a message, and sends that message to other users currently on the chat. If the server encounters an error while trying to receive a message from a particular thread, it will exit that thread.

Client Side Script

The client side script will simply attempt to access the server socket created at the specified IP address and port. Once it connects, it will continuously check as to whether the input comes from the server or from the client, and accordingly redirects output. If the input is from the server, it displays the message on the terminal. If the input is from the user, it sends the message that the users enters to the server for it to be broadcasted to other users.This is the client side script, that each user must use in order to connect to the server.

10 Concurrent Time Server application using UDP

Aim :

Implement Concurrent Time Server application using UDP to execute the program at remoteserver. Client sends a time request to the server, server sends its system time back to the client. Client displays the result.*

Client, Server and Socket

- Server-A server is a software that waits for client requests and serves or processes them accordingly.
- Client- a client is requester of this service. A client program request for some resources to the server and server responds to that request.
- Socket- Socket is the endpoint of a bidirectional communications channel between server and client. Sockets may communicate within a process, between processes on the same machine, or between processes on different machines. For any communication with a remote program, we have to connect through a socket port.

Time Server

Implement Concurrent Time Server application using UDP to execute the program at remote server. Client sends a time request to the server, server sends its system time back to the client. Client displays the result. We have a UDP based application which sends back current system time to the server indicating that some operation has been performed at the server.

11 Distance vector routing protocol

Aim :

Implement and simulate algorithm for Distance vector routing protocol.

Theory :

A distance-vector routing (DVR) protocol requires that a router inform its neighbors of topology changes periodically. Each router maintains a Distance Vector table containing the distance between itself and all possible destination nodes. Distances,based on a chosen metric, are computed using information from the neighbors' distance vectors. DVR uses Bellman-Ford's Algorithm.

Algorithm :

- Input a matrix, cost of size N X N.
- Initialize a matrix, d of size N X N with values of cost.
- Iterate through each node i.
  - Iterate through each node j.
    - Iterate through each node k.
      - d[i][j]=min(d[i][j],cost[i][k]+d[k][j])
- Display d[i][1..N] for all node i.

12 Link state routing protocol

Aim :

Implement and simulate algorithm for Link state routing protocol.

Theory :

Link-State Routing protocol is a main class of routing protocols. It is performed by every switching node/router in the network. The basic concept of link-state routing is that every node constructs a map of the connectivity to the network, in the form of a Graph, showing which nodes are connected to which other nodes. Each node then independently calculates the next best logical path from it to every possible destination in the network. The collection of best paths will then form the node's routing table.

Algorithm :

Link State Routing Algorithm for source node u.

- Add u to vector, N'.
- Input cost matrix, c.
- for all node v
    - if v is a neighbour of u
        - D[v]=c[u][v]
    - else
        - D[v]= ∞
- Iterate till size of N' becomes N (no of nodes in network)
    - Find a node w not in N' such that D(w) is minimum
    - Add w to N'
    - Update D[v] for each neighbour v of w and not in N'
        - D[v]=min(D[v],D[w]+c[w][v])
- print d[v] for all node v

13 Simple Mail Transfer Protocol

Aim :

To implement a subset of Simple Mail transfer Protocol using UDP.

Theory :

Within the Internet, email is delivered by having the sending computer establish a TCP connection to port 25 of the receiving computer. Listening to this port is a mail server that speaks SMTP (Simple Mail Transfer Protocol). This server accepts incoming connections, subject to some security checks, and accepts messages for delivery. If a message cannot be delivered, an error report con- taining the first part of the undeliverable message is returned to the sender.

SMTP is a simple ASCII protocol. This is not a weakness but a feature. Using ASCII text makes protocols easy to develop, test, and debug. They can be tested by sending commands manually, and records of the messages are easy to read. Most application-level Internet protocols now work this way (e.g., HTTP).

After establishing the TCP connection to port 25, the sending machine, operating as the client, waits for the receiving machine, operating as the server, to talk first. The server starts by sending a line of text giving its identity and telling whether it is prepared to receive mail. If it is not, the client releases the connection and tries again later. If the server is willing to accept email, the client announces whom the email is coming from and whom it is going to. If such a recipient exists at the destination, the server gives the client the go-ahead to send the message. Then the client sends the message and the server acknowledges it. No checksums are needed because TCP provides a reliable byte stream. If there is more email, that is now sent. When all the email has been exchanged in both directions, the connection is released.

The basic SMTP works well, but it is limited in several respects. It does not include authentication. This means that the FROM command in the example could give any sender address that it pleases. This is quite useful for sending spam. An- other limitation is that SMTP transfers ASCII messages, not binary data. This is why the base64 MIME content transfer encoding was needed. However, with that encoding the mail transmission uses bandwidth inefficiently, which is an issue for large messages. A third limitation is that SMTP sends messages in the clear. It has no encryption to provide a measure of privacy against prying eyes.

To allow these and many other problems related to message processing to be addressed, SMTP was revised to have an extension mechanism. This mechanism is a mandatory part of the RFC 5321 standard. The use of SMTP with extensions is called ESMTP (Extended SMTP).

Protocol Overview

An SMTP Session consits of commands originated by an SMTP client and corresponding responses from the SMTP server so that the session is opened and session parameters are exchanged. A session may include zero or more SMTP transactions. A typical SMTP transaction consists of three command/reply sequences.

- **MAIL** command, to establish the return address, also called return-path, reverse-path, bounce address, mfrom, or envelope sender.
- **RCPT** command, to establish a recipient of the message. This command can be issued multiple times, one for each recipient. These addresses are also part of the envelope.
- **DATA** to signal the beginning of the message text; the content of the message, as opposed to its envelope. It consists of a message header and a message body separated by an empty line. DATA is actually a group of commands, and the server replies twice: once to the DATA command itself, to acknowledge that it is ready to receive the text, and the second time after the end-of-data sequence, to either accept or reject the entire message.

Besides the intermediate reply for DATA, each server's reply can be either positive (2xx reply codes) or negative. Negative replies can be permanent (5xx codes) or transient (4xx codes). A reject is a permanent failure and the client should send a bounce message to the server it received it from. A drop is a positive response followed by message discard rather than delivery.

14 Concurrent file server

Aim :

Develop concurrent file server which will provide the file requested by client if it exists. If not server sends appropriate message to the client. Server should also send its process ID (PID) to clients for display along with file or the message.*

Client, Server and Socket

- Server-A server is a software that waits for client requests and serves or processes them accordingly.
- Client- a client is requester of this service. A client program request for some resources to the server and server responds to that request.
- Socket- Socket is the endpoint of a bidirectional communications channel between server and client. Sockets may communicate within a process, between processes on the same machine, or between processes on different machines. For any communication with a remote program, we have to connect through a socket port.

FTP

The File Transfer Protocol (FTP) is a standard network protocol used for the transfer of computer files between a client and server on a computer network. FTP is built on a client-server model architecture and uses separate controland data connections between the client and the server.[1] FTP users may authenticate themselves with a clear-text sign-in protocol, normally in the form of a username and password, but can connect anonymously if the server is configured to allow it.

15 Wireshark : UDP

Aim :

Using Wireshark observe data transferred in client server communication using UDP and identify the UDP datagram.

Wireshark

Wireshark, a network analysis tool formerly known as Ethereal, captures packets in real time and display them in human-readable format. Wireshark includes filters, color coding, and other features that let you dig deep into network traffic and inspect individual packets.

Getting Wireshark

You can download Wireshark for Windows or macOS from its official website. If you're using Linux or another UNIX-like system, you'll probably find Wireshark in its package repositories. For example, if you're using Ubuntu, you'll find Wireshark in the Ubuntu Software Center.

Capturing Packets

After downloading and installing Wireshark, you can launch it and double-click the name of a network interface under Capture to start capturing packets on that interface. For example, if you want to capture traffic on your wireless network, click your wireless interface.

Filtering Packets

The most basic way to apply a filter is by typing it into the filter box at the top of the window and clicking Apply (or pressing Enter). For example, type "dns" and you'll see only DNS packets. When you start typing, Wireshark will help you autocomplete your filter.

16 Wireshark : Three Way handhaking of TCP

Aim :

Using Wireshark observe Three Way Handshaking Connection Establishment, Data Transfer and Three Way Handshaking Connection Termination in client server communication using TCP.

Refer 15

17 Packet capturing and filtering application

Aim :

Develop a packet capturing and filtering application using raw sockets.

Dependencies

- This manual gives sneak peek of how to capture network packages with pcap and Java.
- For this to work you will need to install libpcap for your operation system. For Windows download and install WinPcap.
- Ubuntu users can install libpcap running this command:

```
sudo apt-get install libpcap-dev
```

- We will use jNetPcap as Java wrapper.

18 Network with multiple subnets with wired and wireless LANs

Aim :

Design and configure a network with multiple subnets with wired and wireless LANs using required network devices. Configure the following services in the network- TELNET, SSH, FTP server, Web server, File server, DHCP server and DNS server.*

TELNET

Telnet is a protocol used on the Internet or local area network to provide a bidirectional interactive text-oriented communication facility using a virtual terminal connection.

SSH

Secure Shell is a cryptographic network protocol for operating network services securely over an unsecured network. Typical applications include remote command-line login and remote command execution, but any network service can be secured with SSH.

FTP Server

An FTP server is a computer which has a file transfer protocol (FTP) address and is dedicated to receiving an FTP connection. An FTP server needs a TCP/IP network for functioning and is dependent on usage of dedicated servers with one or more FTP clients

Web Server

Image result for Web serverwww.fastwebhost.in A Web server is a program that uses HTTP (Hypertext Transfer Protocol) to serve the files that form Web pages to users, in response to their requests, which are forwarded by their computers' HTTP clients.

File Server

In computing, a file server is a computer attached to a network that provides a location for shared disk access, i.e. shared storage of computer files that can be accessed by the workstations that are able to reach the computer that shares the access through a computer network.

DHCP Server

A DHCP Server is a network server that automatically provides and assigns IP addresses, default gateways and other network parameters to client devices. It relies on the standard protocol known as Dynamic Host Configuration Protocol or DHCP to respond to broadcast queries by clients.

DNS Server

A DNS server is a computer server that contains a database of public IP addresses and their associated hostnames, and in most cases, serves to resolve, or translate, those common names to IP addresses as requested.

19 Network simulator NS-2

Aim :

Install network simulator NS-2 in any of the Linux operating system and simulate wired and wireless scenarios.

About NS2

NS (from network simulator) is a name for a series of discrete event network simulators, specifically ns-1, and ns-2. All of them are discrete-event computer network simulators, primarily used in research and teaching.Network simulators are tools used to simulate discrete events in a network and which helps to predict the behaviours of a computer network. Generally the simulated networks have entities like links, switches, hubs, applications, etc. Once the simulation model is complete, it is executed to analyse the performance. Administrators can then customize the simulator to suit their needs. Network simulators typically come with support for the most popular protocols and networks in use today, such as WLAN,UDP,TCP,IP, WAN, etc.

Installation

- Download all in package for ns2 from below link

  https://sourceforge.net/projects/nsnam/files/latest/download

- All the files will be extracted into a folder called "ns-allinone-2.35".
- Ns2 requires a few packages to be pre installed. It also requires the GCC version 4.3 to work correctly
- **Building the dependencies :**

  ```
  sudo apt-get install build-essential autoconf automake libxmu-dev
  ```

  One of the dependencies mentioned is the compiler GCC-4.3, which is no longer available, and thus we have to install GCC-4.4 version. The version 4.4 is the oldest we can get. To do that, use the follwoing command:

  ```
  sudo apt-get install gcc-4.4
  ```

  Now open the file named "ls.h" and scroll to the 137th line. In that change the word "error" to "this->error". The image below shows the line 137 (highlighted in the image below) after making the changes to the ls.h file. We have to tell the ns which version of GCC will be used. In the Makefile.in file, change Change CC= @CC@ to CC=gcc-4.4.

- **Installation**

sudo su cd /ns-allinone-2.35/./install

- **Setting the Environment Path**

  The final step is to tell the system, where the files for ns2 are installed or present. To do that, we have to set the environment path using the ".bashrc" file. In that file, we need to add a few lines at the bottom. The things to be added are given below. But for the path indicated below, many of those lines have "/home/rohit/ns-allinone-2.35/...." , but that is where I have my extracted folder. Make sure you replace them with your path. For example, if you have installed it in a folder "/home/abc", then replace "/home/rohit/ns-allinone-2.35/otcl-1.14" with "/home/abc/ns-allinone-2.35/otcl-1.14"

- **Running NS2**

  Once the system has restarted, open a terminal and start ns2 by using the command:ns

TCL

Tcl (pronounced "tickle" or tee cee ell, /ti si l/) is a high-level, general-purpose, interpreted, dynamic programming language. It was designed with the goal of being very simple but powerful.Tcl casts everything into the mold of a command, even programming constructs like variable assignment and procedure definition. Tcl supports multiple programming paradigms, including object-oriented, imperative and functional programming or procedural styles.

C++

NS2 uses OTcl to create and configure a network, and uses C++ to run simulation. All C++ codes need to be compiled and linked to create an executable file. Since the body of NS2 is fairly large, the compilation time is not negligible. A typical Pentium 4 computer requires few seconds (long enough to annoy