LINUX COMMANDS:

1. ifconfig – check ip address

ifconfig -a

2. traceroute <destination host/IP> - print the route packets transferred to network host

traceroute google.com

3. dig (Domain Information Groper) - interrogating DNS name servers.

dig google.com

4. Telnet - connects destination host/port via a telnet protocol.

Telnet google.com 443

5. nslookup - query Internet domain name servers.

Nslookup google.com

6. netstat - a simple way to review each of your network connections and open sockets.

Netstat

7. scp - secure copy files to and from another host in the network.

scp $filename user@targethost:/$path

8. w - prints a summary of the current activity on the system, including what each user is doing, and their processes.

9. Nmap - checks the opened port on the server. (powerful command)

10. #ifup eth0 – Enable network interface

11. #ifdown eth0 - Disable network interface

SOCKET COMMUNICATION API:

1. socket

To do network I/O, the first thing a process must do is to call the ystem call, specifying the type of communication protocol desired.

#include <sys/types.h>

#include <sys/socket.h>

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

The family is one of

AF\_UNIX -- Unix internal protocols

AF\_INET -- Internet protocols

AF\_NS -- Xerox NS Protocols

AF\_IMPLINK -- IMP link layer

The AF\_ prefix stands for "address family." In the first project, we are going to use AF\_INET.

The socket type is one of the following:

SOCK\_STREAM stream socket

SOCK\_DGRAM datagram socket

SOCK\_RAW raw socket

SOCK\_SEQPACKET sequenced packet socket

SOCK\_RDM reliably delivered message socket

2. bind

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

#include <sys/types.h>

#include <sys/socket.h>

int bind(int sockfd, struct sockaddr \*myaddr, int addrlen);

The first argument is the socket descriptor returned from socket system call. The second argument is a pointer to a protocol-specific address and the third argument is the size of this address. There are three uses of bind.

3. connect

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

#include <sys/types.h>

#include <sys/socket.h>

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

The sockfd is a socket descriptor that was returned by the socket system call. The second and third arguments are a pointer to a socket address, and its size, as described earlier.

4. listen

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

#include <sys/types.h>

#include <sys/socket.h>

int listen(int sockfd, int backlog);

It is usually executed after both the socket and bind system calls, and immediately before the accept system call. The backlog argument specifies how many connection requests can be queued by the system while it waits for the server to execute the accept system call. This argument is usually specified as 5, the maximum value currently allowed.

5. accept

After a connection-oriented server executes the listen system call described above, an actual connection from some client process is waited for by having the server execute the accept system call.

#include <sys/types.h>

#include <sys/socket.h>

int accept(int sockfd, struct sockaddr \*peer, int \*addrlen);

accept takes the first connection request on the queue and creates another socket with the same properties as sockfd. If there are no connection requests pending, this call blocks the caller until one arrives.

The peer and addrlen arguments are used to return the address of the connected peer process (the client). addrlen is called a value-result argument: the caller sets its value before the system call, and the system call stores a result in the variable. For this system call the caller sets addrlen to the size of the sockaddr structure whose address is passed as the peer argument.

6. send, sendto, recv and recvfrom

These system calls are similar to the standard read and write system calls, but additional arguments are required.

#include <sys/types.h>

#include <sys/socket.h>

int send(int sockfd, char \*buff, int nbytes, int flags);

int sendto(int sockfd, char \*buff, int nbytes, int flags, struct sockaddr \*to, int addrlen);

int recv(int sockfd, char \*buff, int nbytes, int flags);

int recvfrom(int sockfd, char \*buff, int nbytes, int flags, struct sockaddr \*from, int \*addrlen);

The first three arguments, sockfd, buff, and nbytes, to the four system calls are similar to the first three arguments for read and write. The flags argument can be safely set to zero ignoring the details for it. The to argument for sendto specifies the protocol-specific address of where the data is to be sent. Since this address is protocol-specific, its length must be specified by addrlen. The recvfrom system call fills in the protocol-specific address of who sent the data into from. The length of this address is also returned to the caller in addrlen.

7.close

The normal Unix close system call is also used to close a socket.

int close(int fd);

If the socket being closed is associated with a protocol that promises reliable delivery (e.g., TCP or SPP), the system must assure that any data within the kernel that still has to be transmitted or acknowledged, is sent. Normally, the system returns from the close immediately, but the kernel still tries to send any data already queued.

UDP CLIENT-SERVER CHAT

