**DEC 2018 Set-1**

1. Write and run a TCP client and a server program using C-language in Linux/Unix environment to perform the following tasks : 20

(a) Establish a TCP connection between a TCP client and a TCP server.

(b) The client program sends two strings to the server.

(c) The server program returns 'YES' if both the strings are equal and `NO' if both are not equal.

Answer

// TCP Server Program

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#define PORT 8080

#define BUFFER\_SIZE 1024

int main() {

int server\_fd, client\_fd;

struct sockaddr\_in server\_addr, client\_addr;

socklen\_t addr\_len;

char buffer1[BUFFER\_SIZE], buffer2[BUFFER\_SIZE];

int valread;

// Create socket

if ((server\_fd = socket(AF\_INET, SOCK\_STREAM, 0)) == 0) {

perror("Socket failed");

exit(EXIT\_FAILURE);

}

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_addr.s\_addr = INADDR\_ANY;

server\_addr.sin\_port = htons(PORT);

// Bind the socket to the address and port

if (bind(server\_fd, (struct sockaddr\*)&server\_addr, sizeof(server\_addr)) < 0) {

perror("Bind failed");

exit(EXIT\_FAILURE);

}

// Listen for incoming connections

if (listen(server\_fd, 3) < 0) {

perror("Listen failed");

exit(EXIT\_FAILURE);

}

printf("Waiting for client to connect...\n");

addr\_len = sizeof(client\_addr);

// Accept a client connection

if ((client\_fd = accept(server\_fd, (struct sockaddr\*)&client\_addr, &addr\_len)) < 0) {

perror("Accept failed");

exit(EXIT\_FAILURE);

}

printf("Client connected\n");

// Receive two strings from the client

valread = read(client\_fd, buffer1, BUFFER\_SIZE);

buffer1[valread] = '\0';

valread = read(client\_fd, buffer2, BUFFER\_SIZE);

buffer2[valread] = '\0';

// Compare the two strings and send the result

if (strcmp(buffer1, buffer2) == 0) {

send(client\_fd, "YES", 3, 0);

} else {

send(client\_fd, "NO", 2, 0);

}

close(client\_fd);

close(server\_fd);

return 0;

}

// TCP Client Program

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#define PORT 8080

#define BUFFER\_SIZE 1024

int main() {

int sock;

struct sockaddr\_in server\_addr;

char buffer1[BUFFER\_SIZE], buffer2[BUFFER\_SIZE];

char response[3];

int valwrite, valread;

// Create socket

if ((sock = socket(AF\_INET, SOCK\_STREAM, 0)) < 0) {

perror("Socket creation failed");

exit(EXIT\_FAILURE);

}

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_port = htons(PORT);

// Convert IPv4 and IPv6 addresses from text to binary

if (inet\_pton(AF\_INET, "127.0.0.1", &server\_addr.sin\_addr) <= 0) {

perror("Invalid address");

exit(EXIT\_FAILURE);

}

// Connect to the server

if (connect(sock, (struct sockaddr\*)&server\_addr, sizeof(server\_addr)) < 0) {

perror("Connection failed");

exit(EXIT\_FAILURE);

}

printf("Connected to server\n");

// Input two strings from the user

printf("Enter the first string: ");

fgets(buffer1, sizeof(buffer1), stdin);

buffer1[strcspn(buffer1, "\n")] = '\0'; // Remove trailing newline

printf("Enter the second string: ");

fgets(buffer2, sizeof(buffer2), stdin);

buffer2[strcspn(buffer2, "\n")] = '\0'; // Remove trailing newline

// Send the strings to the server

valwrite = send(sock, buffer1, strlen(buffer1), 0);

if (valwrite == -1) {

perror("Send failed");

exit(EXIT\_FAILURE);

}

valwrite = send(sock, buffer2, strlen(buffer2), 0);

if (valwrite == -1) {

perror("Send failed");

exit(EXIT\_FAILURE);

}

// Receive the response from the server

valread = read(sock, response, sizeof(response));

if (valread == -1) {

perror("Receive failed");

exit(EXIT\_FAILURE);

}

response[valread] = '\0';

printf("Server response: %s\n", response);

close(sock);

return 0;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_----

2. Write the commands to perform the following tasks : 20

(a) To provide information about network interfaces and all assigned IP addresses. (b) To provide a list of all routers your connection cross when reaching out to a remote system.

**Answer:**

Here are the commands to perform the tasks you described:

**(a) To provide information about network interfaces and all assigned IP addresses:**

1. **Linux (Unix-based systems):**

bash

ifconfig

or

bash

ip a

These commands will display detailed information about all network interfaces, including IP addresses, network masks, and other related data.

1. **Windows:**

cmd

ipconfig

This will show information about all network interfaces and their assigned IP addresses.

**(b) To provide a list of all routers your connection crosses when reaching out to a remote system:**

1. **Linux (Unix-based systems):**

bash

traceroute <destination>

or, if traceroute is not installed:

bash

tracepath <destination>

Replace <destination> with the IP address or hostname of the remote system.

1. **Windows:**

cmd

tracert <destination>

This command will trace the route your data packets take to reach the remote system, showing each router it passes through.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_--

DEC 2018 SET-2

1. Write and run a UDP client and a UDP server using C-language in Unix/Linux for performing the following tasks : 20

(a) Establish a UDP connection between a UDP client and a UDP server.

(b) UDP client will send two variables a and b with values 15 and 20 respectively. (c) The server program with swap the values of these variables i. e., a will have 20 and b will be assigned to 15 and return it to the client.

Answer:

// udp\_server.c

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <arpa/inet.h>

#include <unistd.h>

#define PORT 8080

#define MAXLINE 1024

// Structure to hold the two integers

struct data {

int a;

int b;

};

int main() {

int sockfd;

struct sockaddr\_in servaddr, cliaddr;

socklen\_t len;

struct data values;

// Create socket

sockfd = socket(AF\_INET, SOCK\_DGRAM, 0);

if (sockfd < 0) {

perror("Socket creation failed");

exit(1);

}

memset(&servaddr, 0, sizeof(servaddr));

memset(&cliaddr, 0, sizeof(cliaddr));

// Fill server information

servaddr.sin\_family = AF\_INET;

servaddr.sin\_addr.s\_addr = INADDR\_ANY;

servaddr.sin\_port = htons(PORT);

// Bind the socket with the server address

if (bind(sockfd, (const struct sockaddr \*)&servaddr, sizeof(servaddr)) < 0) {

perror("Bind failed");

close(sockfd);

exit(1);

}

printf("Server is waiting for messages...\n");

len = sizeof(cliaddr);

// Receive the data from the client

recvfrom(sockfd, &values, sizeof(values), 0, (struct sockaddr \*)&cliaddr, &len);

printf("Received values: a = %d, b = %d\n", values.a, values.b);

// Swap the values

int temp = values.a;

values.a = values.b;

values.b = temp;

// Send the swapped values back to the client

sendto(sockfd, &values, sizeof(values), 0, (const struct sockaddr \*)&cliaddr, len);

printf("Swapped values sent back: a = %d, b = %d\n", values.a, values.b);

// Close the socket

close(sockfd);

return 0;

}

// udp\_client.c

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <arpa/inet.h>

#include <unistd.h>

#define PORT 8080

#define MAXLINE 1024

// Structure to hold the two integers

struct data {

int a;

int b;

};

int main() {

int sockfd;

struct sockaddr\_in servaddr;

struct data values;

// Create socket

sockfd = socket(AF\_INET, SOCK\_DGRAM, 0);

if (sockfd < 0) {

perror("Socket creation failed");

exit(1);

}

memset(&servaddr, 0, sizeof(servaddr));

// Fill server information

servaddr.sin\_family = AF\_INET;

servaddr.sin\_port = htons(PORT);

servaddr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

// Initialize the values to be sent

values.a = 15;

values.b = 20;

// Send data to the server

sendto(sockfd, &values, sizeof(values), 0, (const struct sockaddr \*)&servaddr, sizeof(servaddr));

printf("Sent values: a = %d, b = %d\n", values.a, values.b);

// Receive the swapped values from the server

recvfrom(sockfd, &values, sizeof(values), 0, NULL, NULL);

printf("Received swapped values: a = %d, b = %d\n", values.a, values.b);

// Close the socket

close(sockfd);

return 0;

}

-----------------------------------------------------------------------------------------

2. Write the commands to perform the following tasks : 20

(a) Does a DNS query for the user (i. e. find out IP address for a particular system or system name).

(b) To ensure that a network connection can be established with a remote system or test whether a network connection is good or not.

Answer

**(a) DNS Query (to find out the IP address of a particular system or domain name):**

You can use the nslookup or dig command to query DNS and find the IP address of a system or domain.

* **Using nslookup:**

bash

nslookup <domain\_name\_or\_system\_name>

Example:

bash

nslookup example.com

This will return the IP address of the domain example.com.

* **Using dig:**

bash

dig <domain\_name\_or\_system\_name>

Example:

bash

dig example.com

This will also return detailed DNS information, including the IP address.

**(b) To test if a network connection can be established with a remote system or check if the network connection is good:**

You can use the ping or traceroute (or tracert on Windows) command.

* **Using ping (for checking network connection):**

bash

ping <remote\_system\_ip\_or\_hostname>

Example:

bash

ping example.com

This will send ICMP packets to the remote system to check if it's reachable and respond. If the system is reachable, it will reply with response times.

* **Using traceroute (or tracert on Windows) for detailed path check:**

bash

traceroute <remote\_system\_ip\_or\_hostname>

Example (on Linux/macOS):

bash

traceroute example.com

On Windows, use tracert:

bash

tracert example.com

This will display the path packets take to reach the destination system, helping to diagnose network issues along the way.

In summary:

1. nslookup <domain\_name> or dig <domain\_name> for DNS query.
2. ping <remote\_system> for checking network connectivity.
3. traceroute <remote\_system> or tracert <remote\_system> for path and connectivity details.

----------------------

**DEC 2018 SET-3**

1. Write and run a UDP client and a UDP server program using 'C' language in Unix/Limuc for fed the following tasks : 20

(a) Establish a connection between UDP client and a UDP server.

(b) A UDP client will send a string to a server.

(c) The server program counts the number of vowels in the string and returns the value to the client.

Answer:

// udp\_server.c

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <arpa/inet.h>

#include <unistd.h>

#define PORT 8080

#define MAXLINE 1024

// Function to count vowels in a string

int count\_vowels(const char \*str) {

int count = 0;

while (\*str) {

char ch = \*str;

if (ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u' ||

ch == 'A' || ch == 'E' || ch == 'I' || ch == 'O' || ch == 'U') {

count++;

}

str++;

}

return count;

}

int main() {

int sockfd;

struct sockaddr\_in servaddr, cliaddr;

socklen\_t len;

char buffer[MAXLINE];

int vowel\_count;

// Create socket

sockfd = socket(AF\_INET, SOCK\_DGRAM, 0);

if (sockfd < 0) {

perror("Socket creation failed");

exit(1);

}

memset(&servaddr, 0, sizeof(servaddr));

memset(&cliaddr, 0, sizeof(cliaddr));

// Fill server information

servaddr.sin\_family = AF\_INET;

servaddr.sin\_addr.s\_addr = INADDR\_ANY;

servaddr.sin\_port = htons(PORT);

// Bind the socket with the server address

if (bind(sockfd, (const struct sockaddr \*)&servaddr, sizeof(servaddr)) < 0) {

perror("Bind failed");

close(sockfd);

exit(1);

}

printf("Server is waiting for messages...\n");

len = sizeof(cliaddr);

// Receive the string from the client

recvfrom(sockfd, buffer, MAXLINE, 0, (struct sockaddr \*)&cliaddr, &len);

printf("Received string: %s\n", buffer);

// Count the number of vowels in the string

vowel\_count = count\_vowels(buffer);

// Send the vowel count back to the client

sendto(sockfd, &vowel\_count, sizeof(vowel\_count), 0, (const struct sockaddr \*)&cliaddr, len);

printf("Vowel count sent back: %d\n", vowel\_count);

// Close the socket

close(sockfd);

return 0;

}

// udp\_client.c

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <arpa/inet.h>

#include <unistd.h>

#define PORT 8080

#define MAXLINE 1024

int main() {

int sockfd;

struct sockaddr\_in servaddr;

char buffer[MAXLINE];

int vowel\_count;

// Create socket

sockfd = socket(AF\_INET, SOCK\_DGRAM, 0);

if (sockfd < 0) {

perror("Socket creation failed");

exit(1);

}

memset(&servaddr, 0, sizeof(servaddr));

// Fill server information

servaddr.sin\_family = AF\_INET;

servaddr.sin\_port = htons(PORT);

servaddr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

// Get the string from the user

printf("Enter a string: ");

fgets(buffer, MAXLINE, stdin);

// Send the string to the server

sendto(sockfd, buffer, strlen(buffer), 0, (const struct sockaddr \*)&servaddr, sizeof(servaddr));

printf("Sent string: %s\n", buffer);

// Receive the vowel count from the server

recvfrom(sockfd, &vowel\_count, sizeof(vowel\_count), 0, NULL, NULL);

printf("Number of vowels: %d\n", vowel\_count);

// Close the socket

close(sockfd);

return 0;

}

---------------------------------------------------------------

2. Write the commands to perform the following tasks : 20

(a) Displays all connections including TCP and UDP as well as all multicast connections.

(b) To enable authentication. for a single user mode on Linux.

**Answer**

**(a) To display all connections (including TCP, UDP, and multicast):**

You can use the netstat or ss command to list network connections in Linux.

* **Using netstat:**

bash

netstat -tuln

This will display all active TCP (-t) and UDP (-u) connections, showing the ports and the IP addresses they are bound to, along with their state.

If you also want to see multicast connections, you can use:

bash

netstat -g

This shows multicast group memberships.

* + To combine both TCP/UDP and multicast information:

bash

netstat -tuln -g

* **Using ss (Socket Stat):** ss is a more modern and faster tool compared to netstat. To display TCP, UDP, and multicast connections:

bash

ss -tuln

This will show active TCP (-t), UDP (-u), and listening sockets (-l).

For multicast connections:

bash

ss -g

This will show multicast group memberships.

To display all connections (TCP/UDP + Multicast):

bash

ss –tulng

**(b) To enable authentication for a single user in Linux:**

You can enable authentication for a single user using various authentication methods, but the most common one is enabling the **password authentication** for a specific user through configuration in /etc/ssh/sshd\_config or through user account settings. Here's how:

1. **Enable password authentication for a single user in SSH:**
   * Open the SSH configuration file:

bash

sudo nano /etc/ssh/sshd\_config

* + Ensure the following settings are configured correctly:

bash

PasswordAuthentication yes

* + To enforce authentication for a specific user, add a rule like:

bash

Match User <username>

PasswordAuthentication yes

Replace <username> with the actual username.

* + Save and close the file, then restart the SSH service:

bash

sudo systemctl restart sshd

1. **Enable password for a user:** If you want to ensure that a password is set for a user, use:

bash

sudo passwd <username>

Replace <username> with the username of the user for whom you want to enable authentication. This command will prompt you to set a password for that user.

If you wish to disable password authentication for all other users (not just the one), you could update /etc/ssh/sshd\_config to enforce that only the specific user can authenticate with a password, while others may need keys or other forms of authentication.

1. **Enable Authentication for sudo:** If you want to enforce authentication for using sudo for a user, check the sudoers file:

bash

sudo visudo

Ensure that the user’s line looks something like:

bash

<username> ALL=(ALL) ALL

This enforces that the user must enter their password to use sudo.

In summary:

1. **Display all connections (TCP, UDP, multicast):**
   * netstat -tuln -g or ss -tulng
2. **Enable authentication for a single user:**
   * Configure /etc/ssh/sshd\_config for SSH authentication or use passwd to set a password for the user.

=============================================

**DEC 2018 SET 4**

1. Write and run a TCP client and a TCP server program in C-language in

Linux/Unix environment to perform the following tasks : 20

(a) Establish a TCP connection between a TCP client and a TCP server.

(b) A TCP client program will send the following numbers :

5, 10, 2, 4, 7, 3

to the server program.

(c) A TCP server will sort the numbers in descending order.

Answer

// server.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#define PORT 8080

#define MAX\_NUMBERS 6

// Function to compare two integers for sorting in descending order

int compare(const void \*a, const void \*b) {

return (\*(int\*)b - \*(int\*)a); // Descending order

}

int main() {

int server\_fd, new\_socket, valread;

struct sockaddr\_in address;

char buffer[1024] = {0};

int numbers[MAX\_NUMBERS];

// Create server socket

if ((server\_fd = socket(AF\_INET, SOCK\_STREAM, 0)) == 0) {

perror("Socket failed");

exit(EXIT\_FAILURE);

}

// Set address structure

address.sin\_family = AF\_INET;

address.sin\_addr.s\_addr = INADDR\_ANY;

address.sin\_port = htons(PORT);

// Bind socket to address

if (bind(server\_fd, (struct sockaddr \*)&address, sizeof(address)) < 0) {

perror("Bind failed");

exit(EXIT\_FAILURE);

}

// Listen for connections

if (listen(server\_fd, 3) < 0) {

perror("Listen failed");

exit(EXIT\_FAILURE);

}

printf("Server listening on port %d\n", PORT);

// Accept incoming connection

if ((new\_socket = accept(server\_fd, (struct sockaddr \*)&address, (socklen\_t\*)&address)) < 0) {

perror("Accept failed");

exit(EXIT\_FAILURE);

}

// Read numbers sent by the client

valread = read(new\_socket, numbers, sizeof(numbers));

if (valread < 0) {

perror("Read failed");

exit(EXIT\_FAILURE);

}

// Sort numbers in descending order

qsort(numbers, MAX\_NUMBERS, sizeof(int), compare);

// Send sorted numbers back to client

send(new\_socket, numbers, sizeof(numbers), 0);

printf("Sorted numbers sent to client\n");

// Close the socket

close(new\_socket);

close(server\_fd);

return 0;

}

// client.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#define PORT 8080

#define MAX\_NUMBERS 6

int main() {

int sock = 0;

struct sockaddr\_in serv\_addr;

int numbers[MAX\_NUMBERS] = {5, 10, 2, 4, 7, 3};

int sorted\_numbers[MAX\_NUMBERS];

// Create client socket

if ((sock = socket(AF\_INET, SOCK\_STREAM, 0)) < 0) {

perror("Socket failed");

exit(EXIT\_FAILURE);

}

serv\_addr.sin\_family = AF\_INET;

serv\_addr.sin\_port = htons(PORT);

// Convert IP address to binary format

if (inet\_pton(AF\_INET, "127.0.0.1", &serv\_addr.sin\_addr) <= 0) {

perror("Invalid address");

exit(EXIT\_FAILURE);

}

// Connect to the server

if (connect(sock, (struct sockaddr \*)&serv\_addr, sizeof(serv\_addr)) < 0) {

perror("Connection failed");

exit(EXIT\_FAILURE);

}

// Send numbers to the server

send(sock, numbers, sizeof(numbers), 0);

printf("Numbers sent to server: ");

for (int i = 0; i < MAX\_NUMBERS; i++) {

printf("%d ", numbers[i]);

}

printf("\n");

// Receive sorted numbers from the server

read(sock, sorted\_numbers, sizeof(sorted\_numbers));

// Display sorted numbers

printf("Sorted numbers received from server: ");

for (int i = 0; i < MAX\_NUMBERS; i++) {

printf("%d ", sorted\_numbers[i]);

}

printf("\n");

// Close the socket

close(sock);

return 0;

}

=================================================

2. Write the command(s) to perform the following tasks : 20

(a) Checking the status of destination host and communication with

another host name.

(b) Finding host/domain name and IP address.

Answer

**(a) Checking the status of a destination host and communication with another host name**

1. **Checking the status of the destination host:** You can use the ping command to check the status of a destination host. It sends ICMP echo requests to the specified host to check if it's reachable.

bash

ping <hostname\_or\_IP>

Example:

bash

ping google.com

This command sends ICMP packets to google.com and reports on the communication status, including whether the destination is reachable or not.

1. **Communication with another host name:** To check the communication between your system and another host, you can also use ping or traceroute to trace the route packets take to the destination. The ping command is simpler, but if you need more detailed information about the route, use traceroute.

bash

traceroute <hostname\_or\_IP>

Example:

bash

traceroute google.com

This will show the route packets take to reach google.com, including any intermediate hosts.

**(b) Finding host/domain name and IP address**

1. **Finding the host/domain name from an IP address:** To find the domain name associated with an IP address, use the nslookup or host command.
   * **Using nslookup:**

bash

nslookup <IP\_address>

Example:

bash

nslookup 8.8.8.8

This will return the domain name corresponding to the IP address 8.8.8.8.

* + **Using host:**

bash

host <IP\_address>

Example:

bash

host 8.8.8.8

This command also returns the domain name associated with the IP address.

1. **Finding the IP address of a domain/host name:** To find the IP address associated with a domain name, use the nslookup or host command:
   * **Using nslookup:**

bash

nslookup <hostname>

Example:

bash

nslookup google.com

This will return the IP addresses associated with google.com.

* + **Using host:**

bash

host <hostname>

Example:

bash

host google.com

This command also returns the IP address of google.com.

**Summary of Commands:**

* **Checking the status of a host:**

bash

ping <hostname\_or\_IP>

traceroute <hostname\_or\_IP>

* **Finding host/domain name from an IP address:**

bash

nslookup <IP\_address>

host <IP\_address>

* **Finding IP address from a domain/host name:**

bash

nslookup <hostname>

host <hostname>