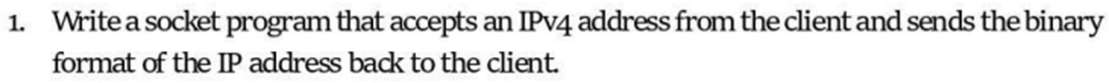
Computer Networks Lab-4

Akhil P S – 24MCS1018

MTech CSE

IP addressing and Classless Addressing



1. Write a socket program that accepts an IPv4 address from the client and sends the binary format of the IP address back to the client.

Server.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <netinet/in.h>

#define PORT 8080
#define BUFFER_SIZE 16

void convert_ip_to_binary(const char *ip, char *binary_ip) {
    struct in_addr addr;
    if (inet_pton(AF_INET, ip, &addr) != 1) {
        strcpy(binary_ip, "Invalid IP");
        return;
    }

    uint32_t ip_num = ntohl(addr.s_addr);
    for (int i = 31; i >= 0; i--) {
        binary_ip[31 - i] = (ip_num & (1 << i)) ? '1' : '0';
    }
    binary_ip[32] = '\0';
}

int main() {
    int server_fd, new_socket;
    struct sockaddr_in address;
    socklen_t addr_len = sizeof(address);
    char buffer[BUFFER_SIZE] = {0};
    char binary_ip[33];

    // Creating socket
    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
        perror("Socket failed");
        exit(EXIT_FAILURE);
    }

    // Configure server address
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons(PORT);

    // Bind the socket
    if (bind(server_fd, (struct sockaddr *)&address, sizeof(address)) < 0) {
        perror("Bind failed");
        exit(EXIT_FAILURE);
    }
```

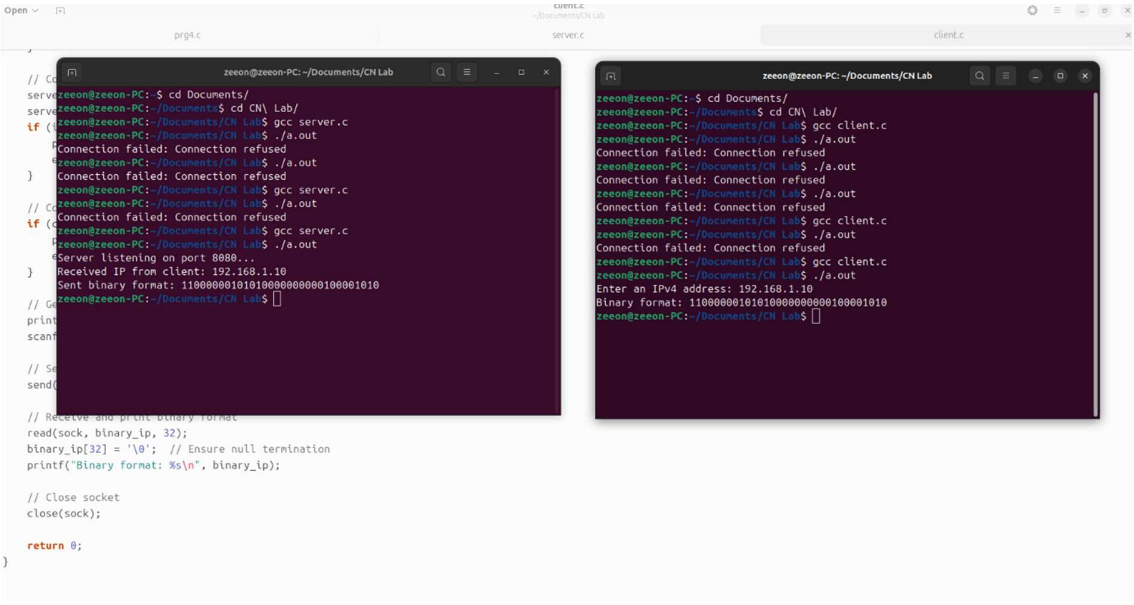
```
// Listen for incoming connections
if (listen(server_fd, 3) < 0) {
    perror("Listen failed");
    exit(EXIT_FAILURE);
}

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

// Accept a client connection
if ((new_socket = accept(server_fd, (struct sockaddr *)&address, &addr_len)) < 0) {
    perror("Accept failed");
    exit(EXIT_FAILURE);
}

// Read IPv4 address from client
read(new_socket, buffer, BUFFER_SIZE);
printf("Received IP from client: %s\n", buffer);

// Convert to binary and send back
convert_ip_to_binary(buffer, binary_ip);
send(new_socket, binary_ip, strlen(binary_ip), 0);
printf("Sent binary format: %s\n", binary_ip);

// Close sockets
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 BUFFER_SIZE 16

int main() {
    int sock;
    struct sockaddr_in server_addr;
    char ip_address[BUFFER_SIZE];
    char binary_ip[33] = {0};

    // Create socket
    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

    // Configure server address
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(PORT);
    if (inet_pton(AF_INET, "127.0.0.1", &server_addr.sin_addr) <= 0) {
        perror("Invalid address");
        exit(EXIT_FAILURE);
    }

    // Connect to server
    if (connect(sock, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0) {
        perror("Connection failed");
        exit(EXIT_FAILURE);
    }

    // Get user input
    printf("Enter an IPv4 address: ");
    scanf("%15s", ip_address);

    // Send IP address to server
    send(sock, ip_address, strlen(ip_address), 0);

    // Receive and print binary format
    read(sock, binary_ip, 32);
    binary_ip[32] = '\0';  // Ensure null termination
    printf("Binary format: %s\n", binary_ip);

    // Close socket
    close(sock);

    return 0;
}
```

Output:



```
zeeon@zeeon-PC: ~/Documents/CN Lab
zeeon@zeeon-PC:~$ cd Documents/
zeeon@zeeon-PC:~/Documents$ cd CN\ Lab/
zeeon@zeeon-PC:~/Documents/CN Lab$ gcc server.c
zeeon@zeeon-PC:~/Documents/CN Lab$ ./a.out
Connection failed: Connection refused
zeeon@zeeon-PC:~/Documents/CN Lab$ ./a.out
Connection failed: Connection refused
zeeon@zeeon-PC:~/Documents/CN Lab$ gcc server.c
zeeon@zeeon-PC:~/Documents/CN Lab$ ./a.out
Connection failed: Connection refused
zeeon@zeeon-PC:~/Documents/CN Lab$ gcc server.c
zeeon@zeeon-PC:~/Documents/CN Lab$ ./a.out
Server listening on port 8080...
Received IP from client: 192.168.1.10
Sent binary format: 11000000101010000000000100001010
zeeon@zeeon-PC:~/Documents/CN Lab$ 
```

```
zeeon@zeeon-PC: ~/Documents/CN Lab
zeeon@zeeon-PC:~$ cd Documents/
zeeon@zeeon-PC:~/Documents$ cd CN\ Lab/
zeeon@zeeon-PC:~/Documents/CN Lab$ gcc client.c
zeeon@zeeon-PC:~/Documents/CN Lab$ ./a.out
Connection failed: Connection refused
zeeon@zeeon-PC:~/Documents/CN Lab$ ./a.out
Connection failed: Connection refused
zeeon@zeeon-PC:~/Documents/CN Lab$ ./a.out
Connection failed: Connection refused
zeeon@zeeon-PC:~/Documents/CN Lab$ gcc client.c
zeeon@zeeon-PC:~/Documents/CN Lab$ ./a.out
Connection failed: Connection refused
zeeon@zeeon-PC:~/Documents/CN Lab$ gcc client.c
zeeon@zeeon-PC:~/Documents/CN Lab$ ./a.out
Enter an IPv4 address: 192.168.1.10
Binary format: 11000000101010000000000100001010
zeeon@zeeon-PC:~/Documents/CN Lab$ 
```

```
    // Receive and print binary format
    read(sock, binary_ip, 32);
    binary_ip[32] = '\0';  // Ensure null termination
    printf("Binary format: %s\n", binary_ip);

    // Close socket
    close(sock);

    return 0;
}
```

2. Write a socket program where the client sends an IP address and a subnet mask to the server, and the server calculates the network address.
   <u>Steps:</u>

- The client sends an IP address and subnet mask to the server (e.g., 192.168.1.10 and 255.255.255.0).
- The server calculates the network address (192.168.1.0) using bitwise operations.
- The network address is sent back to the client.

<u>**Server.c:**</u>

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <arpa/inet.h>

#include <unistd.h>


#define PORT 8080
```

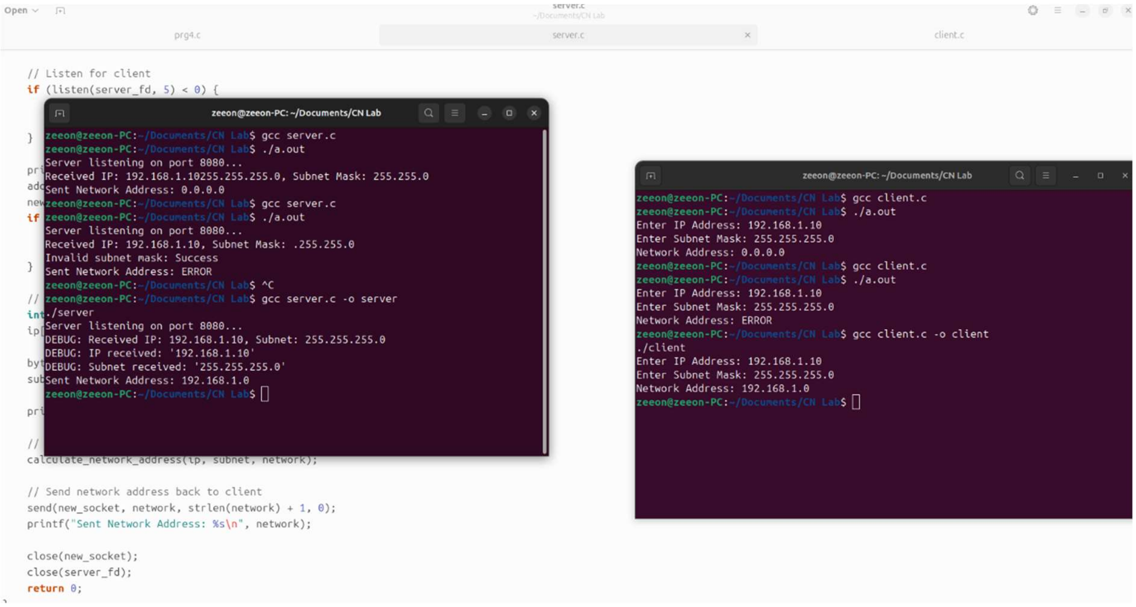
```
void calculate_network_address(char *ip, char *subnet, char *network) {
    struct in_addr ip_addr, subnet_addr, net_addr;

    printf("DEBUG: IP received: '%s'\n", ip);
    printf("DEBUG: Subnet received: '%s'\n", subnet);

    // Convert IP and subnet to binary form
    if (inet_pton(AF_INET, ip, &ip_addr) <= 0) {
        perror("ERROR: Invalid IP address");
        strcpy(network, "ERROR");
        return;
    }

    if (inet_pton(AF_INET, subnet, &subnet_addr) <= 0) {
        perror("ERROR: Invalid subnet mask");
        strcpy(network, "ERROR");
        return;
    }

    // Compute network address (bitwise AND)
    net_addr.s_addr = ip_addr.s_addr & subnet_addr.s_addr;

    // Convert back to string
    if (inet_ntop(AF_INET, &net_addr, network, INET_ADDRSTRLEN) == NULL) {
        perror("ERROR: Failed to convert network address");
        strcpy(network, "ERROR");
    }
}
```

```
int main() {
    int server_fd, new_socket;
    struct sockaddr_in server_addr, client_addr;
    socklen_t addr_size;
    char ip[INET_ADDRSTRLEN] = {0}, subnet[INET_ADDRSTRLEN] = {0},
network[INET_ADDRSTRLEN] = {0};

    // Create socket
    server_fd = socket(AF_INET, SOCK_STREAM, 0);
    if (server_fd == -1) {
        perror("Socket creation 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 socket
    if (bind(server_fd, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0) {
        perror("Bind failed");
        exit(EXIT_FAILURE);
    }

    // Listen for client
    if (listen(server_fd, 5) < 0) {
        perror("Listen failed");
        exit(EXIT_FAILURE);
    }
    printf("Server listening on port %d...\n", PORT);
    addr_size = sizeof(client_addr);
    new_socket = accept(server_fd, (struct sockaddr *)&client_addr, &addr_size);
    if (new_socket < 0) {
        perror("Accept failed");
        exit(EXIT_FAILURE);
    }

    // Receive IP address and subnet mask from client
    int bytes_received = recv(new_socket, ip, sizeof(ip), 0);
    ip[bytes_received] = '\0'; // Ensure null termination

    bytes_received = recv(new_socket, subnet, sizeof(subnet), 0);
    subnet[bytes_received] = '\0'; // Ensure null termination

    printf("DEBUG: Received IP: %s, Subnet: %s\n", ip, subnet);

    // Compute network address
    calculate_network_address(ip, subnet, network);

    // Send network address back to client
    send(new_socket, network, strlen(network) + 1, 0);
    printf("Sent Network Address: %s\n", network);

    close(new_socket);
    close(server_fd);
    return 0;
}
```

Client.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <arpa/inet.h>
#include <unistd.h>

#define PORT 8080

int main() {
    int sock;
    struct sockaddr_in server_addr;
    char ip[INET_ADDRSTRLEN] = {0}, subnet[INET_ADDRSTRLEN] = {0}, network[INET_ADDRSTRLEN] = {0};

    // Create socket
    sock = socket(AF_INET, SOCK_STREAM, 0);
    if (sock == -1) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(PORT);
    inet_pton(AF_INET, "127.0.0.1", &server_addr.sin_addr);

    // Connect to server
    if (connect(sock, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0) {
        perror("Connection failed");
        exit(EXIT_FAILURE);
    }

    // Get user input
    printf("Enter IP Address: ");
    scanf("%s", ip);
    printf("Enter Subnet Mask: ");
    scanf("%s", subnet);

    // Send data to server (including null terminator)
    send(sock, ip, strlen(ip) + 1, 0);
    send(sock, subnet, strlen(subnet) + 1, 0);

    // Receive and print network address from server
    recv(sock, network, sizeof(network), 0);
    printf("Network Address: %s\n", network);

    close(sock);
    return 0;
}
```

Output:

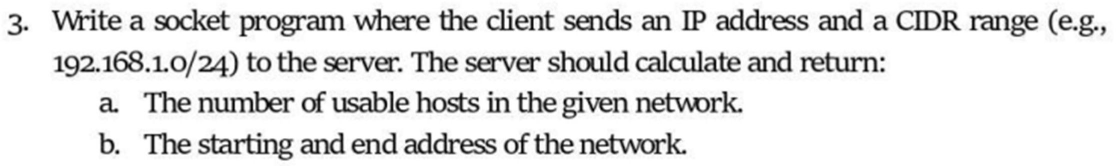

Open
server.c
~/Documents/CN Lab
prg4.c
server.c
client.c
zeeon@zeeon-PC: ~/Documents/CN Lab
zeeon@zeeon-PC: ~/Documents/CN Lab

```
// Listen for client
if (listen(server_fd, 5) < 0) {
```

```
zeeon@zeeon-PC:~/Documents/CN Lab$ gcc server.c
zeeon@zeeon-PC:~/Documents/CN Lab$ ./a.out
Server listening on port 8080...
Received IP: 192.168.1.10255.255.255.0, Subnet Mask: 255.255.0
Sent Network Address: 0.0.0.0
zeeon@zeeon-PC:~/Documents/CN Lab$ gcc server.c
zeeon@zeeon-PC:~/Documents/CN Lab$ ./a.out
Server listening on port 8080...
Received IP: 192.168.1.10, Subnet Mask: .255.255.0
Invalid subnet mask: Success
Sent Network Address: ERROR
zeeon@zeeon-PC:~/Documents/CN Lab$ ^C
zeeon@zeeon-PC:~/Documents/CN Lab$ gcc server.c -o server
./server
Server listening on port 8080...
DEBUG: Received IP: 192.168.1.10, Subnet: 255.255.255.0
DEBUG: IP received: '192.168.1.10'
DEBUG: Subnet received: '255.255.255.0'
Sent Network Address: 192.168.1.0
zeeon@zeeon-PC:~/Documents/CN Lab$ 
```

```
zeeon@zeeon-PC:~/Documents/CN Lab$ gcc client.c
zeeon@zeeon-PC:~/Documents/CN Lab$ ./a.out
Enter IP Address: 192.168.1.10
Enter Subnet Mask: 255.255.255.0
Network Address: 0.0.0.0
zeeon@zeeon-PC:~/Documents/CN Lab$ gcc client.c
zeeon@zeeon-PC:~/Documents/CN Lab$ ./a.out
Enter IP Address: 192.168.1.10
Enter Subnet Mask: 255.255.255.0
Network Address: ERROR
zeeon@zeeon-PC:~/Documents/CN Lab$ gcc client.c -o client
./client
Enter IP Address: 192.168.1.10
Enter Subnet Mask: 255.255.255.0
Network Address: 192.168.1.0
zeeon@zeeon-PC:~/Documents/CN Lab$ 
```

```
calculate_network_address(ip, subnet, network);

// Send network address back to client
send(new_socket, network, strlen(network) + 1, 0);
printf("Sent Network Address: %s\n", network);

close(new_socket);
close(server_fd);
return 0;
```



3. Write a socket program where the client sends an IP address and a CIDR range (e.g., 192.168.1.0/24) to the server. The server should calculate and return:
    a. The number of usable hosts in the given network.
    b. The starting and end address of the network.

<u>Server.c:</u>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <arpa/inet.h>

#include <unistd.h>

#include <math.h>

#define PORT 8080

```
void calculate_network_info(const char *ip, int cidr, char *start_ip, char *end_ip, int
*usable_hosts) {
    struct in_addr addr, netmask, start, end;

    inet_pton(AF_INET, ip, &addr);  // Convert IP string to binary form

    // Calculate netmask from CIDR
    uint32_t mask = (0xFFFFFFFF << (32 - cidr)) & 0xFFFFFFFF;
    netmask.s_addr = htonl(mask);

    // Calculate network start address
    start.s_addr = addr.s_addr & netmask.s_addr;

    // Calculate broadcast address
    end.s_addr = start.s_addr | ~netmask.s_addr;

    // Usable hosts calculation
    *usable_hosts = (cidr == 32 || cidr == 31) ? 0 : ((1 << (32 - cidr)) - 2);

    // Convert network and broadcast addresses back to strings
    inet_ntop(AF_INET, &start, start_ip, INET_ADDRSTRLEN);
    inet_ntop(AF_INET, &end, end_ip, INET_ADDRSTRLEN);
}

int main() {
    int server_fd, new_socket;
    struct sockaddr_in address;
    socklen_t addrlen = sizeof(address);
    char buffer[1024] = {0};
```

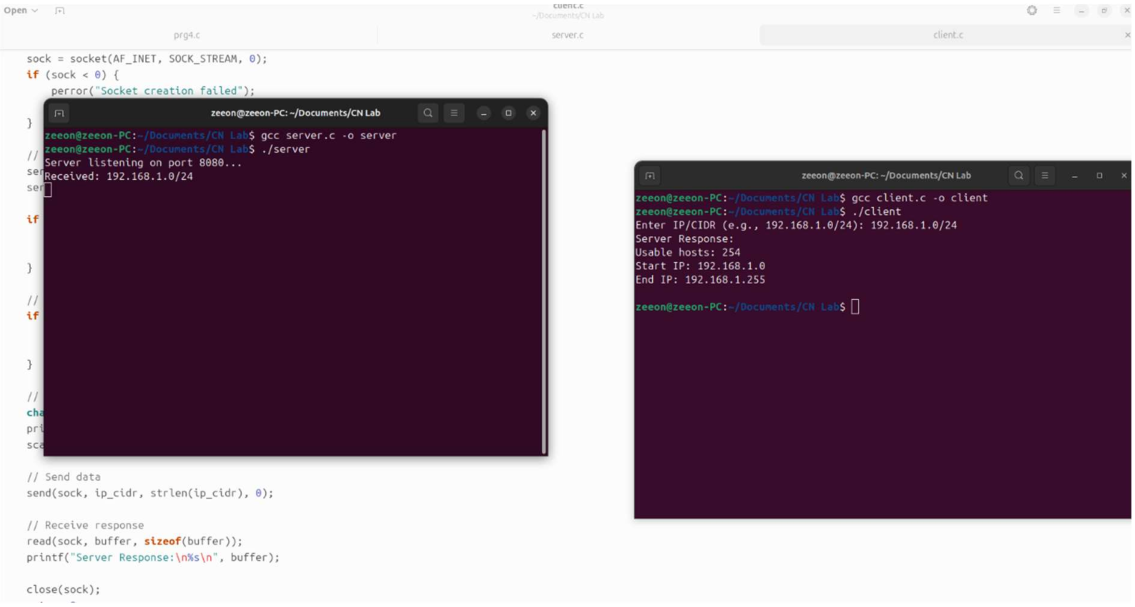
```
// Create socket
server_fd = socket(AF_INET, SOCK_STREAM, 0);
if (server_fd == 0) {
    perror("Socket failed");
    exit(EXIT_FAILURE);
}

// Bind socket
address.sin_family = AF_INET;
address.sin_addr.s_addr = INADDR_ANY;
address.sin_port = htons(PORT);

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);

while (1) {
    // Accept a connection
    new_socket = accept(server_fd, (struct sockaddr *)&address, &addrlen);
```

```
        if (new_socket < 0) {
            perror("Accept failed");
            exit(EXIT_FAILURE);
        }

        read(new_socket, buffer, 1024);
        printf("Received: %s\n", buffer);

        char ip[20];
        int cidr;
        sscanf(buffer, "%[^/]/%d", ip, &cidr);

        char start_ip[INET_ADDRSTRLEN], end_ip[INET_ADDRSTRLEN];
        int usable_hosts;

        calculate_network_info(ip, cidr, start_ip, end_ip, &usable_hosts);

        char response[256];
        snprintf(response, sizeof(response), "Usable hosts: %d\nStart IP: %s\nEnd IP: %s\n",
                 usable_hosts, start_ip, end_ip);

        send(new_socket, response, strlen(response), 0);
        close(new_socket);
    }

    return 0;
}
```

Client.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <arpa/inet.h>
#include <unistd.h>

#define PORT 8080

int main() {
    int sock;
    struct sockaddr_in server_addr;
    char buffer[1024] = {0};

    // Create socket
    sock = socket(AF_INET, SOCK_STREAM, 0);
    if (sock < 0) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

    // Define server address
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(PORT);

    if (inet_pton(AF_INET, "127.0.0.1", &server_addr.sin_addr) <= 0) {
        perror("Invalid address/ Address not supported");
        exit(EXIT_FAILURE);
    }

    // Connect to server
    if (connect(sock, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0) {
        perror("Connection failed");
        exit(EXIT_FAILURE);
    }

    // Get user input
    char ip_cidr[50];
    printf("Enter IP/CIDR (e.g., 192.168.1.0/24): ");
    scanf("%s", ip_cidr);

    // Send data
    send(sock, ip_cidr, strlen(ip_cidr), 0);

    // Receive response
    read(sock, buffer, sizeof(buffer));
    printf("Server Response:\n%s\n", buffer);

    close(sock);
    return 0;
}
```

Output:



```
    sock = socket(AF_INET, SOCK_STREAM, 0);
    if (sock < 0) {
        perror("Socket creation failed");

    }

    //
    se
    ser

    if


    }

    //
    if


    }

    //
    cha
    pri
    sca

    // Send data
    send(sock, ip_cidr, strlen(ip_cidr), 0);

    // Receive response
    read(sock, buffer, sizeof(buffer));
    printf("Server Response:\n%s\n", buffer);

    close(sock);
```

zeeon@zeeon-PC: ~/Documents/CN Lab

```
zeeon@zeeon-PC:~/Documents/CN Lab$ gcc server.c -o server
zeeon@zeeon-PC:~/Documents/CN Lab$ ./server
Server listening on port 8080...
Received: 192.168.1.0/24
```

zeeon@zeeon-PC: ~/Documents/CN Lab

```
zeeon@zeeon-PC:~/Documents/CN Lab$ gcc client.c -o client
zeeon@zeeon-PC:~/Documents/CN Lab$ ./client
Enter IP/CIDR (e.g., 192.168.1.0/24): 192.168.1.0/24
Server Response:
Usable hosts: 254
Start IP: 192.168.1.0
End IP: 192.168.1.255

zeeon@zeeon-PC:~/Documents/CN Lab$
```

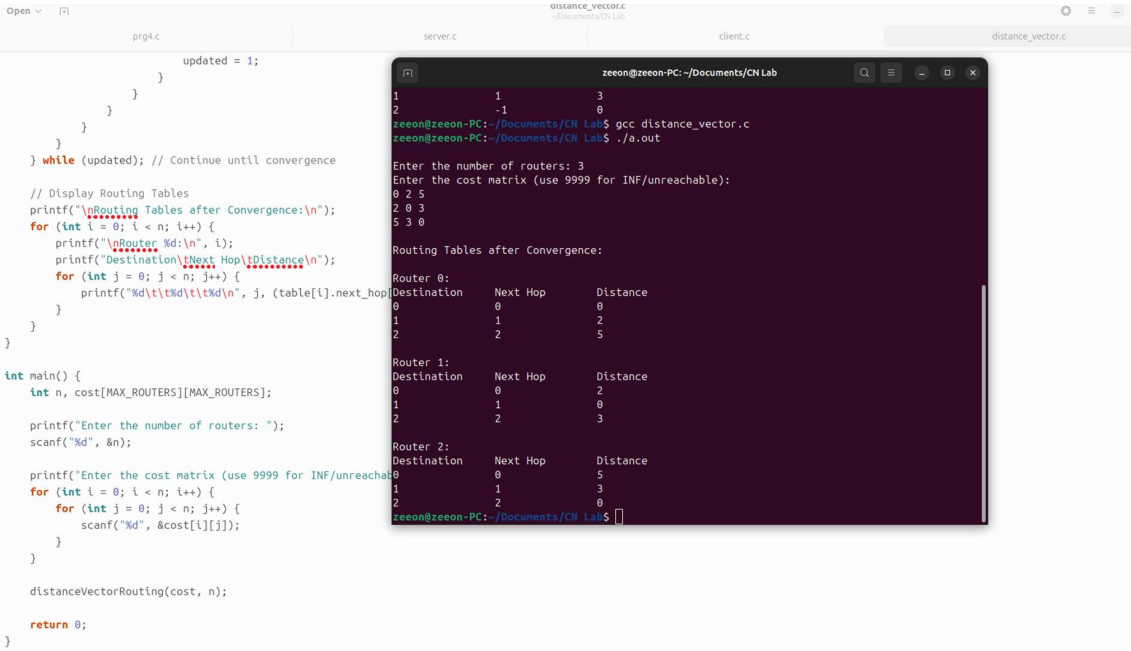
Routing Algorithms

1. Write a C program to simulate the Distance Vector Routing Algorithm. The program should:
    - Accept the number of routers in the network and the cost matrix representing the network topology.
    - Implement the Distance Vector Routing Algorithm to compute the shortest path from each router to every other router.
    - Display the routing table for each router after the algorithm converges.

```
#include <stdio.h>

#include <limits.h>


#define MAX_ROUTERS 10

#define INF 9999  // Represents infinity (unreachable path)


// Structure to store routing table entries
```

```
typedef struct {
    int distance[MAX_ROUTERS];
    int next_hop[MAX_ROUTERS];
} RoutingTable;

void distanceVectorRouting(int cost[MAX_ROUTERS][MAX_ROUTERS], int n) {
    RoutingTable table[MAX_ROUTERS];

    // Initialize routing tables
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            table[i].distance[j] = cost[i][j];
            table[i].next_hop[j] = (cost[i][j] != INF && i != j) ? j : -1;
        }
    }

    int updated;
    do {
        updated = 0;
        for (int i = 0; i < n; i++) { // For each router
            for (int j = 0; j < n; j++) { // For each destination
                for (int k = 0; k < n; k++) { // For each possible next hop
                    if (table[i].distance[k] != INF && cost[k][j] != INF) {
                        int newDist = table[i].distance[k] + cost[k][j];
                        if (newDist < table[i].distance[j]) {
                            table[i].distance[j] = newDist;
                            table[i].next_hop[j] = table[i].next_hop[k];  // Correct next-hop assignment
                            updated = 1;
                        }
                    }
                }
            }
        }
    } while (updated); // Continue until convergence

    // Display Routing Tables
    printf("\nRouting Tables after Convergence:\n");
    for (int i = 0; i < n; i++) {
        printf("\nRouter %d:\n", i);
        printf("Destination\tNext Hop\tDistance\n");
        for (int j = 0; j < n; j++) {
            printf("%d\t\t%d\t\t%d\n", j, (table[i].next_hop[j] == -1 ? i : table[i].next_hop[j]),
table[i].distance[j]);
        }
    }
}

int main() {
    int n, cost[MAX_ROUTERS][MAX_ROUTERS];

    printf("Enter the number of routers: ");
    scanf("%d", &n);

    printf("Enter the cost matrix (use 9999 for INF/unreachable):\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &cost[i][j]);
        }
    }

    distanceVectorRouting(cost, n);

    return 0;
}
```

Output:



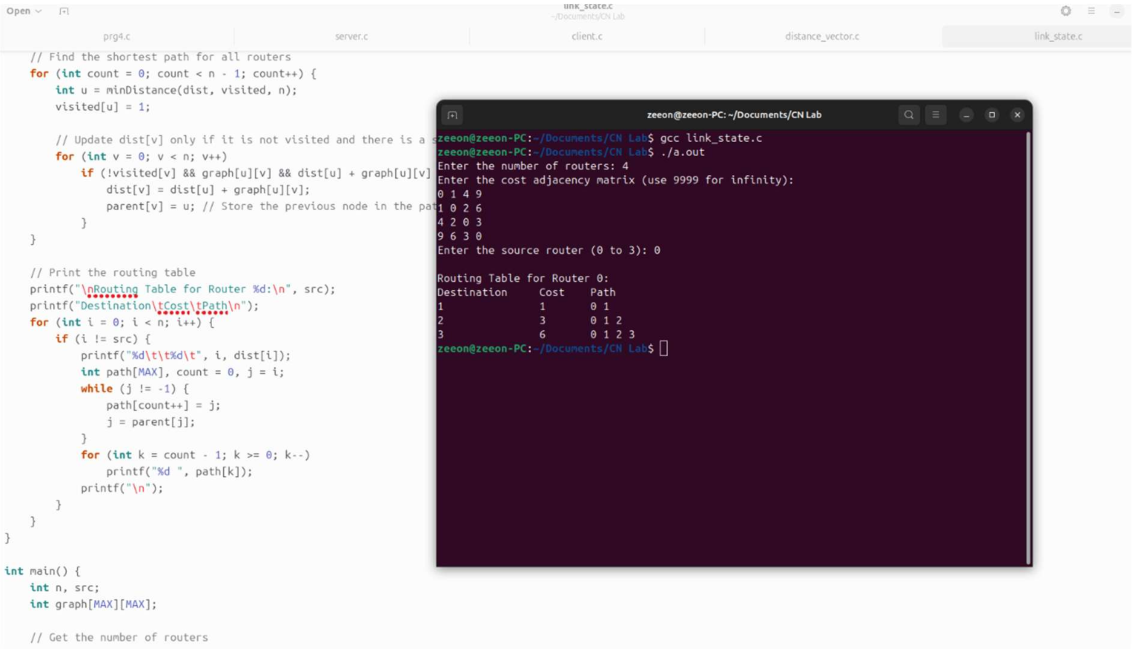
```
                        updated = 1;
                    }
                }
            }
        }
    } while (updated); // Continue until convergence

    // Display Routing Tables
    printf("\nRouting Tables after Convergence:\n");
    for (int i = 0; i < n; i++) {
        printf("\nRouter %d:\n", i);
        printf("Destination\tNext Hop\tDistance\n");
        for (int j = 0; j < n; j++) {
            printf("%d\t\t%d\t\t%d\n", j, (table[i].next_hop[
        }
    }
}

int main() {
    int n, cost[MAX_ROUTERS][MAX_ROUTERS];

    printf("Enter the number of routers: ");
    scanf("%d", &n);

    printf("Enter the cost matrix (use 9999 for INF/unreachab
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &cost[i][j]);
        }
    }

    distanceVectorRouting(cost, n);

    return 0;
}
```

```
1               1               3
2               -1              0
zeeon@zeeon-PC:~/Documents/CN Lab$ gcc distance_vector.c
zeeon@zeeon-PC:~/Documents/CN Lab$ ./a.out

Enter the number of routers: 3
Enter the cost matrix (use 9999 for INF/unreachable):
0 2 5
2 0 3
5 3 0

Routing Tables after Convergence:

Router 0:
Destination     Next Hop        Distance
0               0               0
1               1               2
2               2               5

Router 1:
Destination     Next Hop        Distance
0               0               2
1               1               0
2               2               3

Router 2:
Destination     Next Hop        Distance
0               0               5
1               1               3
2               2               0
zeeon@zeeon-PC:~/Documents/CN Lab$ 
```

2. Write a C program to simulate the Link State Routing Algorithm . The program should:
    - Accept the number of routers and the network topology as input.
    - Use Dijkstra's Algorithm to compute the shortest path from a source router to all other routers.
    - Display the routing table for each router, showing the shortest path and the corresponding cost to reach every other router.

```
#include <stdio.h>
#include <limits.h>

#define MAX 10
#define INF 9999

// Function to find the router with the minimum distance
int minDistance(int dist[], int visited[], int n) {
    int min = INF, min_index;

    for (int v = 0; v < n; v++)
        if (!visited[v] && dist[v] <= min) {
            min = dist[v];
            min_index = v;
        }

    return min_index;
}

// Dijkstra's Algorithm
void dijkstra(int graph[MAX][MAX], int n, int src) {
    int dist[MAX];      // Stores the shortest distance from the source to each node
    int visited[MAX];  // Boolean array to track visited nodes
    int parent[MAX];   // To store the shortest path tree

    // Initialize all distances as infinite and visited[] as false
    for (int i = 0; i < n; i++) {
        dist[i] = INF;
        visited[i] = 0;
        parent[i] = -1;
    }

    // Distance from source to itself is always 0
    dist[src] = 0;

    // Find the shortest path for all routers
    for (int count = 0; count < n - 1; count++) {
        int u = minDistance(dist, visited, n);
        visited[u] = 1;

        // Update dist[v] only if it is not visited and there is a shorter path
        for (int v = 0; v < n; v++)
            if (!visited[v] && graph[u][v] && dist[u] + graph[u][v] < dist[v]) {
                dist[v] = dist[u] + graph[u][v];
                parent[v] = u; // Store the previous node in the path
            }
    }

    // Print the routing table
    printf("\nRouting Table for Router %d:\n", src);
    printf("Destination\tCost\tPath\n");
    for (int i = 0; i < n; i++) {
        if (i != src) {
            printf("%d\t\t%d\t", i, dist[i]);
            int path[MAX], count = 0, j = i;
            while (j != -1) {
                path[count++] = j;
                j = parent[j];
            }
            for (int k = count - 1; k >= 0; k--)
                printf("%d ", path[k]);
            printf("\n");
        }
    }
}

int main() {
    int n, src;
    int graph[MAX][MAX];

    // Get the number of routers
    printf("Enter the number of routers: ");
    scanf("%d", &n);

    // Get the cost adjacency matrix
    printf("Enter the cost adjacency matrix (use 9999 for infinity):\n");
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            scanf("%d", &graph[i][j]);

    // Get the source router
    printf("Enter the source router (0 to %d): ", n - 1);
    scanf("%d", &src);

    // Run Dijkstra's algorithm
    dijkstra(graph, n, src);


    return 0;
}
```

Output:



```
    // Find the shortest path for all routers
    for (int count = 0; count < n - 1; count++) {
        int u = minDistance(dist, visited, n);
        visited[u] = 1;

        // Update dist[v] only if it is not visited and there is a s
        for (int v = 0; v < n; v++)
            if (!visited[v] && graph[u][v] && dist[u] + graph[u][v]
                dist[v] = dist[u] + graph[u][v];
                parent[v] = u; // Store the previous node in the pat
            }
    }

    // Print the routing table
    printf("\nRouting Table for Router %d:\n", src);
    printf("Destination\tCost\tPath\n");
    for (int i = 0; i < n; i++) {
        if (i != src) {
            printf("%d\t\t%d\t", i, dist[i]);
            int path[MAX], count = 0, j = i;
            while (j != -1) {
                path[count++] = j;
                j = parent[j];
            }
            for (int k = count - 1; k >= 0; k--)
                printf("%d ", path[k]);
            printf("\n");
        }
    }
}

int main() {
    int n, src;
    int graph[MAX][MAX];

    // Get the number of routers
```

```
zeeon@zeeon-PC:~/Documents/CN Lab$ gcc link_state.c
zeeon@zeeon-PC:~/Documents/CN Lab$ ./a.out
Enter the number of routers: 4
Enter the cost adjacency matrix (use 9999 for infinity):
0 1 4 9
1 0 2 6
4 2 0 3
9 6 3 0
Enter the source router (0 to 3): 0

Routing Table for Router 0:
Destination     Cost    Path
1               1       0 1
2               3       0 1 2
3               6       0 1 2 3
zeeon@zeeon-PC:~/Documents/CN Lab$ 
```

Flow Control

Sender.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
```

```
#include <arpa/inet.h>

#define SERVER_IP "127.0.0.1"
#define PORT 8080
#define TOTAL_FRAMES 10  // Number of frames to send

int main() {
    int sock = 0;
    struct sockaddr_in serv_addr;
    int frame = 1;
    int window_size = 1;

    // Create socket
    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        perror("Socket creation error");
        exit(EXIT_FAILURE);
    }

    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(PORT);

    // Convert IPv4 addresses from text to binary form
    if (inet_pton(AF_INET, SERVER_IP, &serv_addr.sin_addr) <= 0) {
        perror("Invalid address / Address not supported");
        exit(EXIT_FAILURE);
    }

    // Connect to server
    if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0) {
```

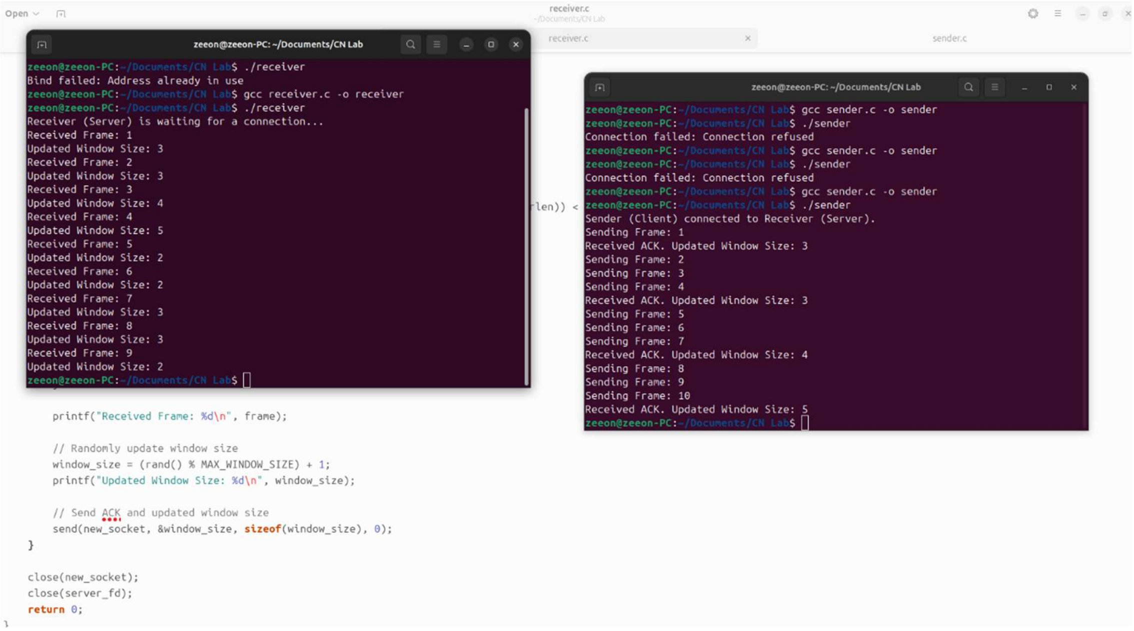
```
    perror("Connection failed");
    exit(EXIT_FAILURE);
}

printf("Sender (Client) connected to Receiver (Server).\n");

while (frame <= TOTAL_FRAMES) {
    for (int i = 0; i < window_size && frame <= TOTAL_FRAMES; i++) {
        printf("Sending Frame: %d\n", frame);
        send(sock, &frame, sizeof(frame), 0);
        frame++;
    }

    // Wait for ACK and updated window size
    if (recv(sock, &window_size, sizeof(window_size), 0) <= 0) {
        break;
    }

    printf("Received ACK. Updated Window Size: %d\n", window_size);

    // Pause if window size is 0
    while (window_size == 0) {
        printf("Window size is 0. Waiting...\n");
        recv(sock, &window_size, sizeof(window_size), 0);
        printf("Resuming with new Window Size: %d\n", window_size);
    }
}

close(sock);
```

```
    return 0;
}
```

Receiver.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <time.h>

#define PORT 8080
#define MAX_WINDOW_SIZE 5  // Maximum window size

int main() {
    int server_fd, new_socket;
    struct sockaddr_in address;
    int addrlen = sizeof(address);
    int frame;
    int window_size;

    // Create socket
    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
        perror("Socket failed");
        exit(EXIT_FAILURE);
    }
    address.sin_family = AF_INET;

    address.sin_addr.s_addr = INADDR_ANY;

    address.sin_port = htons(PORT);


    // Bind socket

    if (bind(server_fd, (struct sockaddr *)&address, sizeof(address)) < 0) {

        perror("Bind failed");

        exit(EXIT_FAILURE);

    }


    // Listen

    if (listen(server_fd, 3) < 0) {

        perror("Listen failed");

        exit(EXIT_FAILURE);

    }


    printf("Receiver (Server) is waiting for a connection...\n");


    // Accept connection

    if ((new_socket = accept(server_fd, (struct sockaddr *)&address, (socklen_t*)&addrlen)) < 0) {

        perror("Accept failed");

        exit(EXIT_FAILURE);

    }


    srand(time(0));  // Seed random window size


    while (1) {

        // Receive frame
        if (recv(new_socket, &frame, sizeof(frame), 0) <= 0) {
            break; // If connection closed, exit
        }
        printf("Received Frame: %d\n", frame);

        // Randomly update window size
        window_size = (rand() % MAX_WINDOW_SIZE) + 1;
        printf("Updated Window Size: %d\n", window_size);

        // Send ACK and updated window size
        send(new_socket, &window_size, sizeof(window_size), 0);
    }
    close(new_socket);
    close(server_fd);
    return 0;
}
```

Output:



Open
receiver.c
~/Documents/CN Lab
receiver.c
sender.c
zeeon@zeeon-PC: ~/Documents/CN Lab
zeeon@zeeon-PC: ~/Documents/CN Lab
len)) <

```
zeeon@zeeon-PC:~/Documents/CN Lab$ ./receiver
Bind failed: Address already in use
zeeon@zeeon-PC:~/Documents/CN Lab$ gcc receiver.c -o receiver
zeeon@zeeon-PC:~/Documents/CN Lab$ ./receiver
Receiver (Server) is waiting for a connection...
Received Frame: 1
Updated Window Size: 3
Received Frame: 2
Updated Window Size: 3
Received Frame: 3
Updated Window Size: 4
Received Frame: 4
Updated Window Size: 5
Received Frame: 5
Updated Window Size: 2
Received Frame: 6
Updated Window Size: 2
Received Frame: 7
Updated Window Size: 3
Received Frame: 8
Updated Window Size: 3
Received Frame: 9
Updated Window Size: 2
zeeon@zeeon-PC:~/Documents/CN Lab$
```

```
zeeon@zeeon-PC:~/Documents/CN Lab$ gcc sender.c -o sender
zeeon@zeeon-PC:~/Documents/CN Lab$ ./sender
Connection failed: Connection refused
zeeon@zeeon-PC:~/Documents/CN Lab$ gcc sender.c -o sender
zeeon@zeeon-PC:~/Documents/CN Lab$ ./sender
Connection failed: Connection refused
zeeon@zeeon-PC:~/Documents/CN Lab$ gcc sender.c -o sender
zeeon@zeeon-PC:~/Documents/CN Lab$ ./sender
Sender (Client) connected to Receiver (Server).
Sending Frame: 1
Received ACK. Updated Window Size: 3
Sending Frame: 2
Sending Frame: 3
Sending Frame: 4
Received ACK. Updated Window Size: 3
Sending Frame: 5
Sending Frame: 6
Sending Frame: 7
Received ACK. Updated Window Size: 4
Sending Frame: 8
Sending Frame: 9
Sending Frame: 10
Received ACK. Updated Window Size: 5
zeeon@zeeon-PC:~/Documents/CN Lab$
```

```
        printf("Received Frame: %d\n", frame);

        // Randomly update window size
        window_size = (rand() % MAX_WINDOW_SIZE) + 1;
        printf("Updated Window Size: %d\n", window_size);

        // Send ACK and updated window size
        send(new_socket, &window_size, sizeof(window_size), 0);
    }

    close(new_socket);
    close(server_fd);
    return 0;
}
```