EXPERIMENT-23

DESIGN THE FUNCTIONALITIES AND EXPLORATION OF UDP USING PACKET TRACER

Aim:

To design the functionalities and exploration of UDP (User Datagram Protocol) using Packet Tracer.

Software/Apparatus required:

Packet Tracer, End devices (PCs), Router, Switch, Server, Ethernet cables.

Procedure:

Step 1: Setup the network topology

1. Open Packet Tracer and create a network topology as shown in the diagram.
2. Drag the following devices onto the workspace:
    - Router0 (ISR 331)
    - Switch0 (Switch-PT)
    - Server0 (Server-PT) with IP address 192.168.1.10
    - PC0 (PC-PT) with IP address 192.168.1.1
    - PC1 (PC-PT) with IP address 192.168.1.2
3. Connect the devices as follows:
    - Connect PC0 and PC1 to Switch0 using Ethernet cables.
    - Connect Switch0 to Router0.
    - Connect Server0 to Router0.

Step 2: Configure IP addresses

1. Double-click on each PC and the server to open the configuration window.
2. Navigate to the Desktop tab and click on the IP Configuration icon.
3. Assign IP addresses and subnet masks:
    - PC0: IP address = 192.168.1.1, Subnet mask = 255.255.255.0
    - PC1: IP address = 192.168.1.2, Subnet mask = 255.255.255.0
    - Server0: IP address = 192.168.1.10, Subnet mask = 255.255.255.0

Step 3: Configure the router

1. Double-click on Router0 to open the configuration window.
2. Navigate to the CLI tab and enter the following commands:

```
enable
configure terminal
interface FastEthernet0/0
ip address 192.168.1.254 255.255.255.0
```

no shutdown

exit

exit

This configures the router's interface with the IP address 192.168.1.254 and enables it.

Step 4: Test the connection

1. Open the command prompt on PC0 and ping PC1 by typing:

ping 192.168.1.2

2. Open the command prompt on PC1 and ping Server0 by typing:

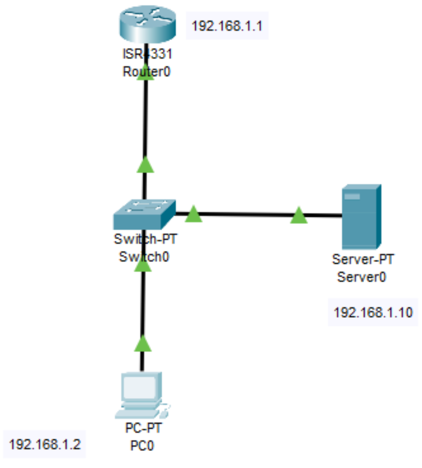
ping 192.168.1.10

3. If the pings are successful, it confirms that the devices are communicating.

Step 5: Explore UDP functionalities

1. Use a UDP-based application or utility (e.g., a simple UDP sender/receiver script or a network tool like Netcat) to simulate UDP communication.
2. On PC0, set up a UDP sender to send data to Server0 on a specific port (e.g., port 5000).
3. On Server0, set up a UDP receiver to listen on the same port (5000).
4. Observe the data transmission. Note that UDP does not guarantee delivery, order, or error-checking, unlike TCP.

Diagram



Output:

Result:

Thus, the functionalities and exploration of UDP using Packet Tracer were designed successfully.

EXPERIMENT-24

DESIGNING TWO DIFFERENT NETWORKS WITH DYNAMIC ROUTING TECHNIQUES (RIP & OSPF) USING PACKET TRACER

Aim:

To design two different networks using dynamic routing protocols (RIP and OSPF) and analyze their functionalities using Packet Tracer.

Software/Apparatus required:

Packet Tracer, Routers (ISR 331, Router-PT), Switches (2560-2XT), PCs, Ethernet cables.

Procedure:

Network 1: Dynamic Routing using RIP

Step 1: Setup the network topology

1. Open Packet Tracer and create the first network topology as shown in Diagram 1:
    - Router0 (ISR 331)
    - Router1 (ISR 331)
    - Switch0 (2560-2XT) connected to Router0
    - Switch1 (2560-2XT) connected to Router1
    - PC0 and PC1 connected to Switch0
    - PC2 and PC3 connected to Switch1

Step 2: Configure IP addresses

1. Assign IP addresses to the PCs:
    - PC0: 192.168.1.1/24
    - PC1: 192.168.1.2/24
    - PC2: 192.168.2.1/24
    - PC3: 192.168.2.2/24
2. Configure the router interfaces:
    - Router0 (ISR 331):
        - Interface connected to Switch0: 192.168.1.254/24
        - Interface connected to Router1: 10.0.0.1/30
    - Router1 (ISR 331):
        - Interface connected to Switch1: 192.168.2.254/24
        - Interface connected to Router0: 10.0.0.2/30

Step 3: Configure RIP routing

1. On Router0, enter the following commands:

enable

configure terminal

router rip

version 2

network 192.168.1.0

network 10.0.0.0

no auto-summary

exit

2. On Router1, enter the following commands:

enable

configure terminal

router rip

version 2

network 192.168.2.0

network 10.0.0.0

no auto-summary

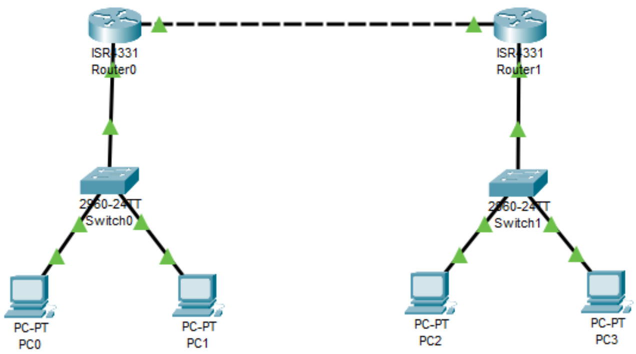
exit

Step 4: Test the connection

1. Use the ping command to test connectivity between PCs across the network.
    - For example, ping PC2 from PC0:

ping 192.168.2.1

Diagram



Output :

Network 2: Dynamic Routing using OSPF

Step 1: Setup the network topology

1. Open Packet Tracer and create the second network topology as shown in Diagram 2:
    - Router3 (Router-PT)
    - Router4 (Router-PT)
    - Router5 (Router-PT)
    - Router6 (Router-PT)
    - Router7 (Router-PT)
    - PC0, PC1, PC2, and PC3 connected to respective routers.

Step 2: Configure IP addresses

1. Assign IP addresses to the PCs:
    - PC0: 192.168.10.1/24
    - PC1: 192.168.20.1/24
    - PC2: 192.168.30.1/24
    - PC3: 192.168.40.1/24
2. Configure the router interfaces:
    - Assign IP addresses to all router interfaces based on the network design.

Step 3: Configure OSPF routing

1. On each router, enable OSPF and advertise the connected networks. For example, on Router3:

```
enable
configure terminal
router ospf 1
network 192.168.10.0 0.0.0.255 area 0
network <connected network> <wildcard mask> area 0
exit
```

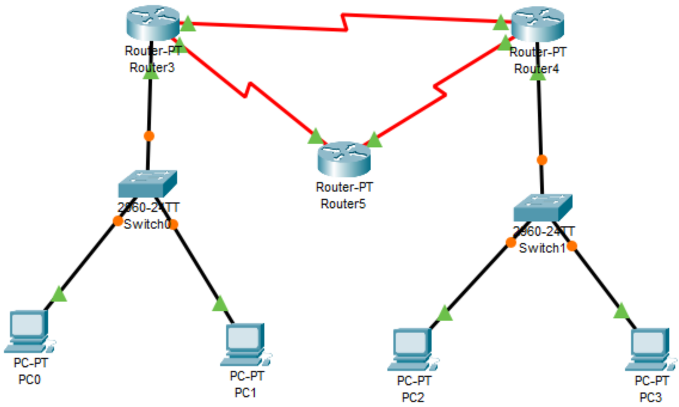
2. Repeat the OSPF configuration on all routers, ensuring all networks are advertised in Area 0.

Step 4: Test the connection

1. Use the ping command to test connectivity between PCs across the network.
    - For example, ping PC3 from PC0:

```
ping 192.168.40.1
```

Diagram:



Output:

Result:

Thus, two different networks using dynamic routing techniques (RIP and OSPF) were designed and analyzed successfully using Packet Tracer.

Date:

EXPERIMENT: 25

TRANSPORT LAYER PROTOCOL HEADER ANALYSIS USING WIRE SHARK-TCP

Aim: To analyze capturing of Transport layer protocol header analysis using Wire shark- TCP

SOFTWARE USED:

Wire shark network analyzer

Procedure:

1. Open wire shark.
2. Click on list the available capture interface.
3. Choose the LAN interface.
4. Click on start button.
5. Active packets will be displayed.
6. Capture the packets & select any IP address from the source.
7. Click on the expression and select IPV4 →IP addr source address in the field name.
8. Select the double equals (==) from the selection and enter the selected IP source address.
9. Click on apply button.
10. All the packets will be filtered using source address.

Result: Hence, the capturing of packets using wire shark network analyzer was analyzed for TCP

Date:

EXPERIMENT: 26

TRANSPORT LAYER PROTOCOL HEADER ANALYSIS USING WIRE SHARK- UDP

Aim: To analyze capturing of Transport layer protocol header analysis using Wire shark- UDP.

SOFTWARE USED:

Wire shark network analyzer

Procedure:

1. Open wire shark.
2. Click on list the available capture interface.
3. Choose the LAN interface.
4. Click on start button.
5. Active packets will be displayed.
6. Capture the packets & select any IP address from the source.
7. Click on the expression and select IPV4 →IP addr source address in the field name.
8. Select the double equals (==) from the selection and enter the selected IP source address.
9. Click on apply button.
10. All the packets will be filtered using source address.

Result: Hence, the capturing of packets using wire shark network analyzer was analyzed for UDP.

Date:

EXPERIMENT-27

NETWORK LAYER PROTOCOL HEADER ANALYSIS USING WIRE SHARK – SMTP

Aim: To analyze capturing of Transport layer protocol header analysis using Wire shark- SMTP

SOFTWARE USED:

Wire shark network analyzer

Procedure:

1. Open wire shark.
2. Click on list the available capture interface.
3. Choose the LAN interface.
4. Click on start button.
5. Active packets will be displayed.
6. Capture the packets & select any IP address from the source.
7. Click on the expression and select IPV4 →IP addr source address in the field name.
8. Select the double equals (==) from the selection and enter the selected IP source address.
9. Click on apply button.
10. All the packets will be filtered using source address.

Result: Hence, the capturing of packets using wire shark network analyzer was analyzed for SMTP

Date:

EXPERIMENT-28

NETWORK LAYER PROTOCOL HEADER ANALYSIS USING WIRE SHARK –ICMP

Aim: To analyze capturing of Transport layer protocol header analysis using Wire shark- ICMP.

SOFTWARE USED:

Wire shark network analyzer

Procedure:

1. Open wire shark.
2. Click on list the available capture interface.
3. Choose the LAN interface.
4. Click on start button.
5. Active packets will be displayed.
6. Capture the packets & select any IP address from the source.
7. Click on the expression and select IPV4 →IP addr source address in the field name.
8. Select the double equals (==) from the selection and enter the selected IP source address.
9. Click on apply button.
10. All the packets will be filtered using source address.

Result: Hence, the capturing of packets using wire shark network analyzer was analyzed for ICMP.

Date:

EXPERIMENT-29

NETWORK LAYER PROTOCOL HEADER ANALYSIS USING WIRE SHARK – ARP

AIM: To analyze capturing of Transport layer protocol header analysis using Wire shark- ARP

SOFTWARE USED:

Wire shark network analyzer

PROCEDURE:

1. Open wire shark.
2. Click on list the available capture interface.
3. Choose the LAN interface.
4. Click on start button.
5. Active packets will be displayed.
6. Capture the packets & select any IP address from the source.
7. Click on the expression and select IPV4 →IP addr source address in the field name.
8. Select the double equals (==) from the selection and enter the selected IP source address.
9. Click on apply button.
10. All the packets will be filtered using source address.

Result: Hence, the capturing of packets using wire shark network analyzer was analyzed for ARP

Date:

EXPERIMENT-30

NETWORK LAYER PROTOCOL HEADER ANALYSIS USING WIRE SHARK –HTTP

AIM: To analyze capturing of Transport layer protocol header analysis using Wire shark- HTTP.

SOFTWARE USED:

Wire shark network analyzer

PROCEDURE:

1. Open wire shark.
2. Click on list the available capture interface.
3. Choose the LAN interface.
4. Click on start button.
5. Active packets will be displayed.
6. Capture the packets & select any IP address from the source.
7. Click on the expression and select IPV4 →IP addr source address in the field name.
8. Select the double equals (==) from the selection and enter the selected IP source address.
9. Click on apply button.
10. All the packets will be filtered using source address.

Result: Hence, the capturing of packets using wire shark network analyzer was analyzed for HTTP.

Date:

EXPERIMENT: 31

IMPLEMENTATION OF SERVER – CLIENT USING TCP SOCKET PROGRAMMING

Aim:

To implement a server-client communication model using TCP socket programming in C.

Software/Apparatus Required:

- C Compiler (GCC or any compatible compiler)
- Linux-based OS (or any OS supporting POSIX sockets)
- Text editor (e.g., Vim, Nano, or any IDE)

Procedure:

Step 1: Write the Server-Side Code

1. Open a text editor and write the server-side C program as provided.
2. Save the file as server.c.

Step 2: Write the Client-Side Code

1. Open a text editor and write the client-side C program as provided.
2. Save the file as client.c.

Step 3: Compile the Programs

1. Open the terminal and navigate to the directory containing the server.c and client.c files.
2. Compile the server program using the following command:

gcc server.c -o server

3. Compile the client program using the following command:

gcc client.c -o client

Step 4: Run the Server

1. Execute the server program using the following command:

./server

2. The server will start listening on port 8080.

Step 5: Run the Client

1. Open another terminal window and navigate to the same directory.
2. Execute the client program using the following command:

./client

3. The client will connect to the server running on 127.0.0.1 (localhost) and port 8080.

Step 6: Test the Communication

1. On the client side, type a message and press Enter. The message will be sent to the server.
2. The server will receive the message, display it, and prompt for a response.
3. The server's response will be sent back to the client and displayed on the client's terminal.
4. To end the communication, type "exit" on either the client or server side.

Code:

//SERVER SIDE

```
#include <stdio.h>
#include <netdb.h>
#include <netinet/in.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h> // read(), write(), close()
#define MAX 80
#define PORT 8080
#define SA struct sockaddr

// Function designed for chat between client and server.
void func(int connfd)
{
    char buff[MAX];
    int n;
    // infinite loop for chat
    for (;;) {
        bzero(buff, MAX);

        // read the message from client and copy it in buffer
        read(connfd, buff, sizeof(buff));
        // print buffer which contains the client contents
```

```
        printf("From client: %s\t To client : ", buff);
        bzero(buff, MAX);
        n = 0;
        // copy server message in the buffer
        while ((buff[n++] = getchar()) != '\n')
            ;

        // and send that buffer to client
        write(connfd, buff, sizeof(buff));

        // if msg contains "Exit" then server exit and chat ended.
        if (strncmp("exit", buff, 4) == 0) {
            printf("Server Exit...\n");
            break;
        }
    }
}

// Driver function
int main()
{
    int sockfd, connfd, len;
    struct sockaddr_in servaddr, cli;

    // socket create and verification
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd == -1) {
        printf("socket creation failed...\n");
        exit(0);
    }
    else
        printf("Socket successfully created..\n");
    bzero(&servaddr, sizeof(servaddr));
```

```
// assign IP, PORT
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
servaddr.sin_port = htons(PORT);

// Binding newly created socket to given IP and verification
if ((bind(sockfd, (SA*)&servaddr, sizeof(servaddr))) != 0) {
    printf("socket bind failed...\n");
    exit(0);
}
else
    printf("Socket successfully binded..\n");

// Now server is ready to listen and verification
if ((listen(sockfd, 5)) != 0) {
    printf("Listen failed...\n");
    exit(0);
}
else
    printf("Server listening..\n");
len = sizeof(cli);

// Accept the data packet from client and verification
connfd = accept(sockfd, (SA*)&cli, &len);
if (connfd < 0) {
    printf("server accept failed...\n");
    exit(0);
}
else
    printf("server accept the client...\n");

// Function for chatting between client and server
func(connfd);
```

```
    // After chatting close the socket
    close(sockfd);
}
```

//CLIENT SIDE

```
// Online C compiler to run C program online
#include <arpa/inet.h> // inet_addr()
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <strings.h> // bzero()
#include <sys/socket.h>
#include <unistd.h> // read(), write(), close()
#define MAX 80
#define PORT 8080
#define SA struct sockaddr
void func(int sockfd)
{
    char buff[MAX];
    int n;
    for (;;) {
        bzero(buff, sizeof(buff));
        printf("Enter the string : ");
        n = 0;
        while ((buff[n++] = getchar()) != '\n')
            ;
        write(sockfd, buff, sizeof(buff));
        bzero(buff, sizeof(buff));
        read(sockfd, buff, sizeof(buff));
        printf("From Server : %s", buff);
        if ((strncmp(buff, "exit", 4)) == 0) {
            printf("Client Exit...\n");
```

```
            break;
        }
    }
}

int main()
{
    int sockfd, connfd;
    struct sockaddr_in servaddr, cli;

    // socket create and verification
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd == -1) {
        printf("socket creation failed...\n");
        exit(0);
    }
    else
        printf("Socket successfully created..\n");
    bzero(&servaddr, sizeof(servaddr));

    // assign IP, PORT
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = inet_addr("127.0.0.1");
    servaddr.sin_port = htons(PORT);

    // connect the client socket to server socket
    if (connect(sockfd, (SA*)&servaddr, sizeof(servaddr))
        != 0) {
        printf("connection with the server failed...\n");
        exit(0);
    }
    else
        printf("connected to the server..\n");
```

```
    // function for chat
    func(sockfd);

    // close the socket
    close(sockfd);
}
```

Output:

1. Server-side output:

Copy

Socket successfully created..

Socket successfully binded..

Server listening..

server accept the client...

From client: <Client Message> To client: <Server Response>

2. Client-side output:

Copy

Socket successfully created..

connected to the server..

Enter the string: <Client Message>

From Server: <Server Response>

Result:

Thus, the server-client communication using TCP socket programming was implemented successfully.

EXPERIMENT-32

IMPLEMENTATION OF SERVER – CLIENT USING UDP SOCKET PROGRAMMING

Aim:

To implement a server-client communication model using UDP socket programming in C.

Software/Apparatus Required:

- C Compiler (GCC or any compatible compiler)
- Linux-based OS (or any OS supporting POSIX sockets)
- Text editor (e.g., Vim, Nano, or any IDE)

Procedure:

Step 1: Write the Server-Side Code

1. Open a text editor and write the server-side C program as provided.
2. Save the file as udp_server.c.

Step 2: Write the Client-Side Code

1. Open a text editor and write the client-side C program as provided.
2. Save the file as udp_client.c.

Step 3: Compile the Programs

1. Open the terminal and navigate to the directory containing the udp_server.c and udp_client.c files.
2. Compile the server program using the following command:

gcc udp_server.c -o udp_server

3. Compile the client program using the following command:

gcc udp_client.c -o udp_client

Step 4: Run the Server

1. Execute the server program using the following command:

./udp_server

2. The server will start listening on port 5000.

Step 5: Run the Client

1. Open another terminal window and navigate to the same directory.
2. Execute the client program using the following command:

./udp_client

3. The client will send a message to the server running on 127.0.0.1 (localhost) and port 5000.

Step 6: Test the Communication

1. The client sends a message ("Hello Server") to the server.
2. The server receives the message, prints it, and sends a response ("Hello Client") back to the client.
3. The client receives the server's response and prints it.

Code:

Implementation of server – client using UDP socket programming

```
// server program for udp connection
#include <stdio.h>
#include <strings.h>
#include <sys/types.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include<netinet/in.h>
#define PORT 5000
#define MAXLINE 1000

// Driver code
int main()
{
    char buffer[100];
    char *message = "Hello Client";
    int listenfd, len;
    struct sockaddr_in servaddr, cliaddr;
    bzero(&servaddr, sizeof(servaddr));

    // Create a UDP Socket
    listenfd = socket(AF_INET, SOCK_DGRAM, 0);
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servaddr.sin_port = htons(PORT);
    servaddr.sin_family = AF_INET;

    // bind server address to socket descriptor
```

```
    bind(listenfd, (struct sockaddr*)&servaddr, sizeof(servaddr));

    //receive the datagram
    len = sizeof(cliaddr);
    int n = recvfrom(listenfd, buffer, sizeof(buffer),
            0, (struct sockaddr*)&cliaddr,&len); //receive message from server
    buffer[n] = '\0';
    puts(buffer);

    // send the response
    sendto(listenfd, message, MAXLINE, 0,
          (struct sockaddr*)&cliaddr, sizeof(cliaddr));
}



// udp client driver program
#include <stdio.h>
#include <strings.h>
#include <sys/types.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include<netinet/in.h>
#include<unistd.h>
#include<stdlib.h>

#define PORT 5000
#define MAXLINE 1000

// Driver code
int main()
{
    char buffer[100];
    char *message = "Hello Server";
    int sockfd, n;
```

```
    struct sockaddr_in servaddr;

    // clear servaddr
    bzero(&servaddr, sizeof(servaddr));
    servaddr.sin_addr.s_addr = inet_addr("127.0.0.1");
    servaddr.sin_port = htons(PORT);
    servaddr.sin_family = AF_INET;

    // create datagram socket
    sockfd = socket(AF_INET, SOCK_DGRAM, 0);

    // connect to server
    if(connect(sockfd, (struct sockaddr *)&servaddr, sizeof(servaddr)) < 0)
    {
        printf("\n Error : Connect Failed \n");
        exit(0);
    }

    // request to send datagram
    // no need to specify server address in sendto
    // connect stores the peers IP and port
    sendto(sockfd, message, MAXLINE, 0, (struct sockaddr*)NULL, sizeof(servaddr));

    // waiting for response
    recvfrom(sockfd, buffer, sizeof(buffer), 0, (struct sockaddr*)NULL, NULL);
    puts(buffer);

    // close the descriptor
    close(sockfd);
}
```

Output:

1. Server-side output:

Hello Server

2. Client-side output:

Hello Client

Result:

Thus, the server-client communication using UDP socket programming was implemented successfully.

EXPERIMENT-33
IMPLEMENTATION OF BIT STUFFING MECHANISM USING C

Aim:
To implement the bit stuffing mechanism using the C programming language.
Software/Apparatus required:
C compiler (e.g., GCC), Code editor (e.g., VS Code, Dev C++).

Procedure:
Step 1: Understand the Bit Stuffing Mechanism

1. Bit stuffing is a technique used in data communication to ensure that a specific pattern (e.g., five consecutive 1s) is not mistaken for a control signal.
2. If five consecutive 1s are detected, a 0 is stuffed (inserted) after them to differentiate the data from control signals.

Step 2: Write the C Program

1. Open a code editor and write the following C program to implement bit stuffing:

Step 3: Compile and Run the Program

1. Save the program with a .c extension (e.g., bit_stuffing.c).
2. Compile the program using a C compiler.
3. Run the compiled program
4. **Step 4: Analyze the Output**

The program will output the stuffed bit sequence.
For the input array {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1}, the output will be:
111110111110
Here, a 0 is stuffed after every five consecutive 1s.

Program
```
#include <stdio.h>
#include <string.h>

// Function for bit stuffing
void bitStuffing(int N, int arr[])
{
    // Stores the stuffed array
    int brr[30];

    // Variables to traverse arrays
    int i, j, k;
    i = 0;
    j = 0;

    // Loop to traverse in the range [0, N)
    while (i < N) {

        // If the current bit is a set bit
        if (arr[i] == 1) {

            // Stores the count of consecutive ones
            int count = 1;

            // Insert into array brr[]
            brr[j] = arr[i];

            // Loop to check for
            // next 5 bits
```

```
        for (k = i + 1;
             arr[k] == 1 && k < N && count < 5; k++) {
            j++;
            brr[j] = arr[k];
            count++;

            // If 5 consecutive set bits
            // are found insert a 0 bit
            if (count == 5) {
                j++;
                brr[j] = 0;
            }
            i = k;
        }
    }

    // Otherwise insert arr[i] into
    // the array brr[]
    else {
        brr[j] = arr[i];
    }
    i++;
    j++;
  }

  // Print Answer
  for (i = 0; i < j; i++)
      printf("%d", brr[i]);
}
 // Driver Code
int main()
{
  int N = 12;
  int arr[] = { 1, 1, 1, 1, 1, 1,1,1,1,1,1,1 };

  bitStuffing(N, arr);

  return 0;
}
```

Output:
111110111110

Result:
Thus, the bit stuffing mechanism was successfully implemented using the C programming language.