EXPERIMENT-28

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.

EXPERIMENT: 29

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-30
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.

Hello Server

2. Client-side output:

Hello Client

Result:

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