//BIT STUFFING

#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;

}

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#define PORT 12345

#define BUFFER\_SIZE 1024

int main() {

int sock = 0;

struct sockaddr\_in server\_address;

char buffer[BUFFER\_SIZE] = {0};

// Create socket

sock = socket(AF\_INET, SOCK\_STREAM, 0);

if (sock < 0) {

perror("Socket creation failed");

exit(EXIT\_FAILURE);

}

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

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

// Convert address from text to binary format

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

perror("Invalid address");

exit(EXIT\_FAILURE);

}

// Connect to server

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

perror("Connection failed");

exit(EXIT\_FAILURE);

}

printf("Connected to server. Type 'exit' to quit.\n");

while (1) {

printf("Enter message: ");

fgets(buffer, BUFFER\_SIZE, stdin);

buffer[strcspn(buffer, "\n")] = 0; // Remove newline

if (strcmp(buffer, "exit") == 0)

break;

send(sock, buffer, strlen(buffer), 0);

memset(buffer, 0, BUFFER\_SIZE);

recv(sock, buffer, BUFFER\_SIZE, 0);

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

}

close(sock);

return 0;

}

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#define PORT 12345

#define BUFFER\_SIZE 1024

int main() {

int server\_fd, new\_socket;

struct sockaddr\_in address;

int addrlen = sizeof(address);

char buffer[BUFFER\_SIZE] = {0};

// Create socket

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

if (server\_fd == 0) {

perror("Socket failed");

exit(EXIT\_FAILURE);

}

// Bind to port

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);

// Accept incoming connections

new\_socket = accept(server\_fd, (struct sockaddr\*)&address, (socklen\_t\*)&addrlen);

if (new\_socket < 0) {

perror("Accept failed");

exit(EXIT\_FAILURE);

}

printf("Client connected!\n");

// Communication loop

while (1) {

memset(buffer, 0, BUFFER\_SIZE);

int bytes\_received = recv(new\_socket, buffer, BUFFER\_SIZE, 0);

if (bytes\_received <= 0) {

printf("Client disconnected.\n");

break;

}

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

// Send response

send(new\_socket, "Message received!", strlen("Message received!"), 0);

}

close(new\_socket);

close(server\_fd);

return 0;

}

IMPLEMENTATION OF SERVER – CLIENT USING TCP SOCKET PROGRAMMING

//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);

}