EXERCISE 10

ERROR CORRECTION

AIM:

To implement hamming code to detect and correct error in message transmission from client to server using socket programming in C.

CODE:

Server:

```
#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<sys/types.h>

#include<sys/socket.h>

#include<netinet/in.h>

#include<string.h>

#include<math.h>


int main(int argc,char **argv){

    int len;

    int sockfd, newfd, n;

    struct sockaddr_in servaddr, cliaddr;

    char buff[1024];


    sockfd = socket(AF_INET, SOCK_STREAM, 0);

    if(sockfd < 0) {

        perror("cannot create socket");

        exit(1);

    }


    bzero(&servaddr, sizeof(servaddr));
```

```
servaddr.sin_family = AF_INET;

servaddr.sin_addr.s_addr = htonl(INADDR_ANY);

servaddr.sin_port = htons(7228);


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

    perror("Bind error");

    exit(1);

}


if(listen(sockfd, 2)) {

    perror("listen");

    exit(1);

}


len = sizeof(cliaddr);

newfd = accept(sockfd, (struct sockaddr*)&cliaddr, &len);

if(newfd < 0) {

    perror("accept error");

    exit(1);

}


// Receiving the message

if((n = recv(newfd, buff, sizeof(buff), 0)) < 0) {

    perror("read error");

    exit(1);

}


printf("\nReceived Message is: %s\n", buff);


int num = strlen(buff);

printf("num = %d\n", num);


int r = 0;
```

```
while ((1 << r) < num + 1) {
    r++;
}
printf("r = %d\n", r);

int errorPos = 0;

// Error detection using parity check
for (int i = 0; i < r; i++) {
    int x = 1 << i;  // Calculate 2^i
    int sum = 0;

    // Perform parity check
    for (int j = 1; j <= num; j++) {
        if (j & x) {
            sum ^= (buff[j - 1] - '0');  // Convert char to int for XOR
        }
    }

    // Calculate the error position
    errorPos += sum * x;
}

if (errorPos) {
    printf("Error detected at position: %d\n", errorPos);

    // Correct the error (flip the bit at errorPos)
    buff[errorPos - 1] = (buff[errorPos - 1] == '0') ? '1' : '0';
    printf("Corrected Message is: %s\n", buff);
} else {
    printf("No error detected in received data.\n");
}
```

```
    close(sockfd);
    close(newfd);
    return 0;
}
```

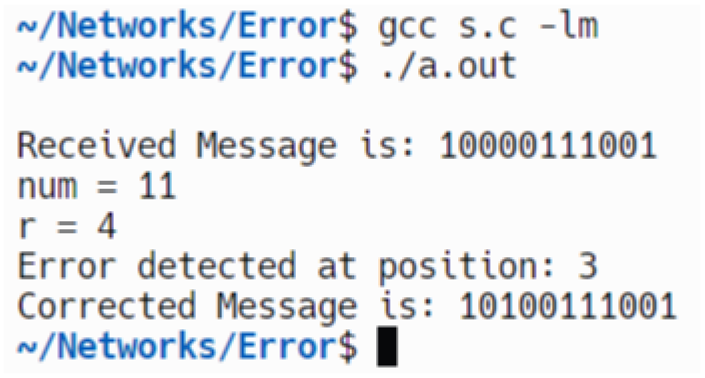
Client:

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

int main(int argc,char **argv){
    int len;
    int sockfd,n;
    struct sockaddr_in servaddr,cliaddr;
    char str[1000];
    char buff[1024];

    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if(sockfd < 0) {
        perror("cannot create socket");
        exit(1);
    }

    bzero(&servaddr,sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = inet_addr(argv[1]);
    servaddr.sin_port = htons(7228);
    if(connect(sockfd, (struct sockaddr*)&servaddr, sizeof(servaddr)) < 0)
{
        perror("connect error");
        exit(1);
    }

    // Getting input message
    printf("Enter the message: ");
    scanf("%s", buff);
    int m = strlen(buff);
    printf("m = %d\n", m);

    int r = 0;
    while ((1 << r) < (m + r + 1)) {
        r++;
    }
    printf("r = %d\n", r);

    // Constructing the Hamming code
    int totalBits = m + r;
    int hammingCode[totalBits];
    memset(hammingCode, 0, sizeof(hammingCode));

    // Placing data bits into their positions (excluding parity bit
positions)
    int j = 0;
    for (int i = 0; i < totalBits; i++) {
        if ((i + 1) == (1 << j)) {
            j++;  // Skip parity bit positions
        } else {
            hammingCode[i] = buff[m - 1] - '0';  // Place data bits
            m--;
        }
```

```
}

// Calculate parity bits and place them in the correct positions
for (int i = 0; i < r; i++) {
    int parityPos = 1 << i;
    int sum = 0;
    for (int j = 0; j < totalBits; j++) {
        if ((j + 1) & parityPos) {
            sum ^= hammingCode[j];  // XOR for parity calculation
        }
    }
    hammingCode[parityPos - 1] = sum;  // Set the calculated parity bit
}

// Display the generated Hamming code
printf("\nCorrect Message: ");
for (int i = 0; i < totalBits; i++) {
    printf("%d", hammingCode[i]);
}
printf("\n");

//Introduce error
int ch;
printf ("\nEnter 1 to introduce error, 0 to send correctly: ");
scanf ("%d", &ch);
if (ch==1)
{
    printf ("\nEnter the position of error: ");
    scanf ("%d", &ch);
    if (hammingCode[ch-1]==0)
        hammingCode[ch-1]=1;
    else
        hammingCode[ch-1]=0;
    }

    // Display the generated Hamming code
    printf("\nMessage sent: ");
    for (int i = 0; i < totalBits; i++) {
        printf("%d", hammingCode[i]);
    }
    printf("\n");

    // Convert to string to send via socket
    char newbuff[totalBits + 1];
    for (int i = 0; i < totalBits; i++) {
        newbuff[i] = hammingCode[i] + '0';  // Convert back to characters
    }
    newbuff[totalBits] = '\0';

    // Send Hamming code to the server
    if((n = send(sockfd, newbuff, sizeof(newbuff), 0)) < 0) {
        perror("write error");
        exit(1);
    }

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

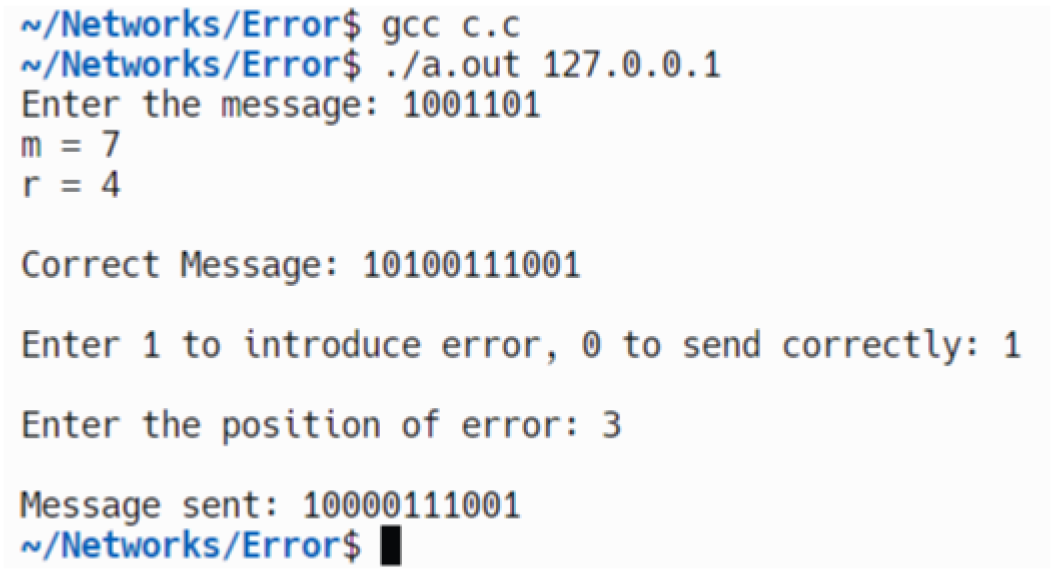
OUTPUT:

Server:



```
~/Networks/Error$ gcc s.c -lm
~/Networks/Error$ ./a.out

Received Message is: 10000111001
num = 11
r = 4
Error detected at position: 3
Corrected Message is: 10100111001
~/Networks/Error$ 
```

Client:



```
~/Networks/Error$ gcc c.c
~/Networks/Error$ ./a.out 127.0.0.1
Enter the message: 1001101
m = 7
r = 4

Correct Message: 10100111001

Enter 1 to introduce error, 0 to send correctly: 1

Enter the position of error: 3

Message sent: 10000111001
~/Networks/Error$ 
```