*UCS1511 - Networks Lab* **Name:** *Krithika Swaminathan*
**AY:** *2022-23* **Roll No.:** *205001057*

## Assignment 6 – Computing Hamming Code for Error Correction

**Date:** 30/09/2022

### Aim:

To implement Hamming Code for Single Error Correction using socket programming in C.

### Algorithm:

Server:

1. Start
2. Read the input from a user (zero's and one's).
3. Perform encoding of the message using Hamming Code:
    a. Calculate the number of redundant bits.
    b. Position the redundant bits.
    c. Calculate the values of each redundant bit.
4. Introduce error (single bit error or no error) in the data and send the data to the receiver.
5. Stop

Client:

1. Start
2. Receive the data from the sender.
3. Check for any errors in the data by performing the following operations:
    a. Calculation of the number of redundant bits.
    b. Positioning the redundant bits.
    c. Parity checking (Counting the number of 1's in the appropriate range of bits).
    d. If there is any error, correct the error in the data.
4. Display the original message.
5. Stop

**Code:**

```
//Functions for implementation of error correction using Hamming code

int countbits(long num)
{
    int r, count = 0;
    while(num > 0)
    {
        num = num / 10;
        count++;
    }
    return count;
}
int binary(int num)
{
    int bin = 0, r;
    int i = 0;
    while(num > 0)
    {
        r = num % 2;
        bin += r * pow(10, i);
        num /= 2;
        i++;
    }
    return bin;
}
int ispresent(int num,int pos)
{
    int rem;
    for(int i = 0; i < pos; i++)
    {
        rem = num % 10;
        num = num / 10;
    }
    if(rem == 1)
        return 1;
    else
        return 0;
}
int isapower2(int n)
{
    if(ceil(log2(n)) == floor(log2(n)))
```

```
        return 1;
    else
        return 0;
}

int decimal(int num)
{
    int rem, i = 0, result;
    while(num > 0)
    {
        rem = num % 10;
        result += pow(2, i) * rem;
        num /= 10;
        i++;
    }
    return result;
}


//Hamming Code Implementation - Server

#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<stdlib.h>
#include<string.h>
#include<unistd.h>
#include<arpa/inet.h>
#include<fcntl.h>
#include<stdbool.h>
#include<math.h>
#define MAXLINE 1024

#include "hamming.h"

int main(int argc, char ** argv)
{
    srand(time(NULL));

    if (argc < 2){
```

```
    fprintf(stderr, "Enter port number as argument!\n");
    exit(EXIT_FAILURE);
}

int PORT = atoi(argv[1]);

int sockfd, newfd, n, arr[30], count = 0, bin;
char buff[MAXLINE], buffer[MAXLINE], data_t[40];
int i, j, r, total, nob, rem, dig, pos;
long data;
struct sockaddr_in servaddr,cliaddr;

if((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
{
    perror("Socket creation failed!");
    exit(1);
}

bzero(&servaddr,sizeof(servaddr));
servaddr.sin_family = AF_INET; // IPv4
servaddr.sin_addr.s_addr = INADDR_ANY;
servaddr.sin_port = htons(PORT);

if(bind(sockfd, (const struct sockaddr *)&servaddr,sizeof(servaddr)) < 0)
{
    perror("Bind failed!");
    exit(1);
}

int len, m;
listen(sockfd, 2);
printf("Enter the data to send: ");
scanf("%lu", &data);

n = countbits(data);
r = log2(n);
r = floor(r);

// Finding number of redundant bits
while(pow(2, r) < n + r + 1)
    r += 1;
```

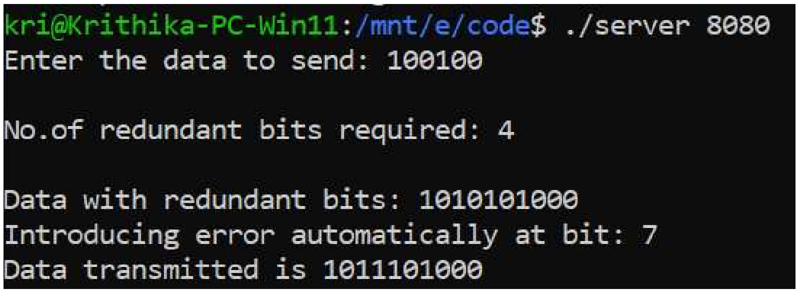
```
printf("\nNo.of redundant bits required: %d\n", r);
total = n + r;
nob = floor(log2(total));
for(i = 1; i <= total; i++)
{
    dig = data % 10;
    if(isapower2(i) == 0)
    {
        arr[total - i] = dig;
        data /= 10;
    }
    else
        arr[total-i]=0;
}
for(i = 0; i < r; i++)
{
    for(j = 1; j <= total; j++)
    {
        if((int)(pow(2, i)) != j)
        {
            bin = binary(j);
            if(ispresent(bin, i + 1))
                count += arr[total - j];
        }
    }
    if(count % 2 == 0)
        arr[total - (int)(pow(2, i))] = 0;
    else
        arr[total - (int)(pow(2, i))] = 1;
    count = 0;
}
printf("\nData with redundant bits: ");
for(i = 0; i < total; i++)
    printf("%d", arr[i]);
// printf("\nEnter error position: ");
// scanf("%d", &pos);

pos = rand() % total + 1;
printf("\nIntroducing error automatically at bit: %d\n", pos);

if(arr[total - pos] == 0)
    arr[total - pos] = 1;
else
```

```
        arr[total - pos] = 0;

    int k = 0;
    long num = 0;
    for(i = total - 1; i >= 0; i--)
    {
        num += pow(10, k) * arr[i];
        k++;
    }
    sprintf(data_t, "%lu", num);
    printf("Data transmitted is %s\n", data_t);

    len = sizeof(cliaddr);
    newfd = accept(sockfd, (struct sockaddr*)&cliaddr, &len);
    m = write(newfd, data_t, sizeof(data_t));
}


//Hamming Code Implementation - Client

#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<string.h>
#include<unistd.h>
#include<arpa/inet.h>
#include<stdlib.h>
#include<math.h>
#define MAXLINE 1024

#include "hamming.h"

int main(int argc, char **argv)
{
    if (argc < 2){
        fprintf(stderr, "Please enter port number as second argument!\n");
        exit(EXIT_FAILURE);
    }

    int PORT = atoi(argv[1]);

    long num;
```

```
int sockfd, total, i, rem, arr[20], count = 0, r = 0, result = 0, bin, j, newarr[20], finalarr[20];
char buffer1[40];
struct sockaddr_in servaddr;
if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
    perror("Socket creation failed!");
    exit(1);
}
bzero(&servaddr,sizeof(servaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_port = htons(PORT);
servaddr.sin_addr.s_addr = inet_addr("127.0.0.1");
int n, len;
connect(sockfd, (struct sockaddr*)&servaddr, sizeof(servaddr));
n = read(sockfd, buffer1, sizeof(buffer1));
num = atol(buffer1);
total = countbits(num);
printf("Received data: %lu\n", num);
i = 1;
while(num > 0)
{
    rem = num % 10;
    arr[total - i] = rem;
    num /= 10;
    i++;
}
for(i = 1; i <= total; i++)
    if(ceil(log2(i)) == floor(log2(i)))
        r++;
int k = 0;
for(i = 0; i < 4; i++)
{
    for(j = 1; j <= total; j++)
    {
        bin = binary(j);
        if(ispresent(bin, i + 1))
            count += arr[total - j];
    }
    if(count % 2 == 0)
        result += pow(10, k) * 0;
    else
        result += pow(10, k) * 1;
    k++;
    count=0;
    }
    int error = decimal(result);
    printf("\nError bit in binary: %d\n", result);
    printf("\nError in bit %d\n", error);
    if(arr[total - error] == 0)
        arr[total - error] = 1;
    else
        arr[total - error] = 0;
    k = 0;
    printf("\nData after error correction: ");
    for(i = total - 1; i >= 0; i--)
    {
        newarr[k] = arr[i];
        k++;
    }
    int x = 0;
    for(i = 0;i < k; i++)
    {
        if(ceil(log2(i + 1)) != floor(log2(i + 1)))
        {
            finalarr[x] = newarr[i];
            x++;
        }
    }
    for(i = x - 1; i >= 0; i--)
        printf("%d", finalarr[i]);
    printf("\n");
    return 0;
}
```

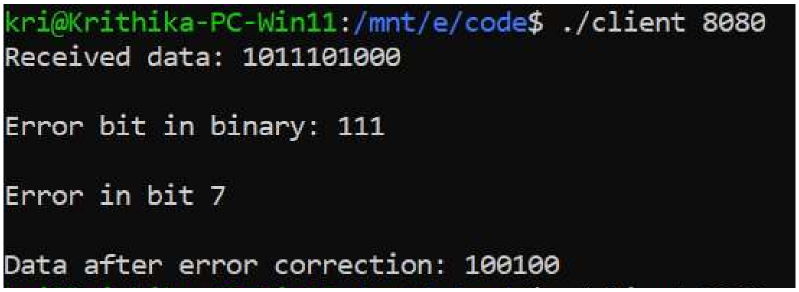
**Output:**

Server side:



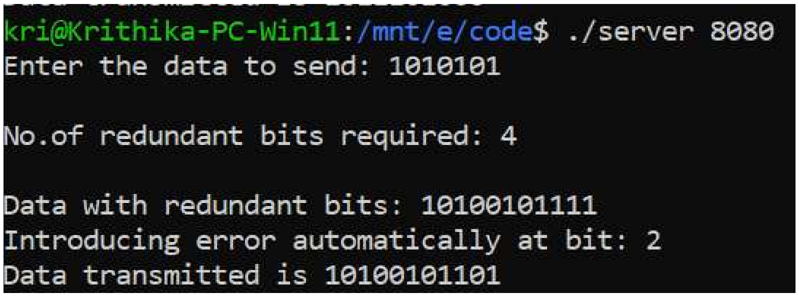
```
kri@Krithika-PC-Win11:/mnt/e/code$ ./server 8080
Enter the data to send: 100100

No.of redundant bits required: 4

Data with redundant bits: 1010101000
Introducing error automatically at bit: 7
Data transmitted is 1011101000
```

Client side:



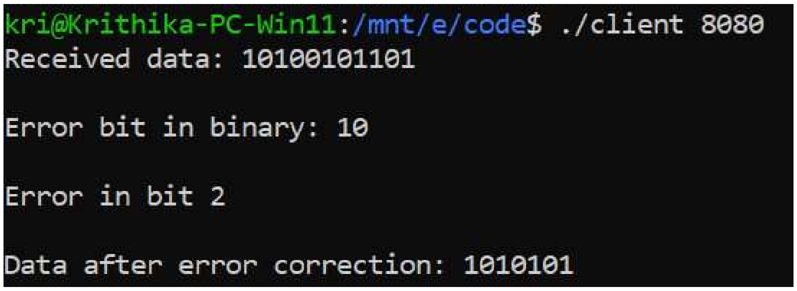
```
kri@Krithika-PC-Win11:/mnt/e/code$ ./client 8080
Received data: 1011101000

Error bit in binary: 111

Error in bit 7

Data after error correction: 100100
```

Server side:



```
kri@Krithika-PC-Win11:/mnt/e/code$ ./server 8080
Enter the data to send: 1010101

No.of redundant bits required: 4

Data with redundant bits: 10100101111
Introducing error automatically at bit: 2
Data transmitted is 10100101101
```

Client side:



```
kri@Krithika-PC-Win11:/mnt/e/code$ ./client 8080
Received data: 10100101101

Error bit in binary: 10

Error in bit 2

Data after error correction: 1010101
```

**Learning outcomes:**

- The procedure for encoding and decoding using Hamming Code was understood.
- Error detection and correction using Hamming Code was understood and implemented.
- Error correction during communication between a client and server to safely transmit the given data was understood and implemented.