SSN College of Engineering, Kalavakkam
Department of Computer Science and Engineering
V Semester - CSE 'B'
UCS1511 NETWORKS LAB

Date : 15/10/2020 **Name :** Rahul Ram M
Exercise : 07 **Reg No :** 185001121

HAMMING CODE ERROR DETECTION AND CORRECTION

Learning Objective:

To develop a socket program to establish a client server communication in which the server and client uses hamming code for error detection and correction.

Algorithm for Server:

1. Read the input from the user(0's and 1's).
2. Fill the message bits in the new array in reverse order leaving appropriate places for redundant bits.
3. Find the number of redundant bits needed by using the equation $2^r \geq m + r + 1$ where, r = redundant bit, m = data bit.
4. Calculate the value of the redundant bit based on the even priority and insert the reduntant bits in the position 2^i where i range from 0 to r.
5. Again reverse the bits in the final array and convert this integer array to a string array.
6. Creating a socket using the function socket(domain, type, protocol) which the returns an integer as the status of the socket creation. Here the domain is AF_INET(iPv4 protocol), type is SOCK_STEAM and protocol as 0.
7. Using bzero(&server_addr, sizeof(server_addr)) function setting values of all the socket structures to null.
8. Using bind() to binf the socket to the address and port number specified in addr(custom data structure). Here, we bind the server to the localhost, hence we use INADDR_ANY to specify the IP address.
9. listen() function is used to set the server socket in the passive mode, where it waits for the client to approach the server to make a connection, with maximum number of connection in this case is 2.
10. accept() creates a new connected socket and returns a new file descriptor referring to the socket. After this the connection between server and client is established.
11. Copy the string array to the buffer.
12. write(new_socket, buffer, sizeof(buffer)) - is used to write the message in the buffer to be sent to client.
13. close() function shuts down the socket associated with socket descriptor's, and frees resources allocated to the socket.

Algorithm for Client:

1. Creating a socket using the function socket(domain, type, protocol) which the returns an integer as the status of the socket creation. Here the domain is AF_INET(iPv4 protocol), type is SOCK_STEAM and protocol as 0.

2. Using bzero(&server_addr, sizeof(server_addr)) function setting values of all the socket structures to null.
3. The above two steps are same as the server.
4. The connect() system call connects the socket referred to by the file descriptor socket_fd to the address specified by server_addr. Server's address and port is specified in server_addr.
5. read(new_socket, buffer, sizeof(buffer)) - reads the message sent by the client in the buffer specified in the parameter along with its size preceded by the new socket descriptor.
6. The string in the buffer contains the message by the server along with the redundant bits by the hamming codes.
7. The string from the server may contains error in any bit.
8. First the string is converted into an integer array and this array is reversed.
9. Now the redundant bits are calculated by ham_calc() function for positions 2^i where i is from 0 to r.
10. Now the decimal value for the obtained redundant bits is calculated.
11. If the decimal value is greater than length of the string then there is no error.
12. Else there is error at the index specified by the decimal value.
13. Flip the bit in that index and extract the message bit leaving all the redundant bits at position 2^i.
14. Printing the corrected message from the server using printf().
15. close() function shuts down the socket associated with socket descriptor's, and frees resources allocated to the socket.

Program for Server:

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

#define max_r 20

int ham_calc(int position,int len, int data[])
{
        int count=0, i, j;
        i = position - 1;
        while(i < len)
        {
                for(j = i; j < i+position; j++)
                {
                        if(data[j] == 1)
                                count++;
                }
                i = i + 2*position;
        }
        if(count%2 == 0)
                return 0;
        else
                return 1;
}

char* convertToString(int n_len, int e_data[])
```

```
{
	char *code = malloc(n_len + 1);
	for(int i = 0; i < n_len; i++)
	{
		code[i] = e_data[i] + '0';
	}
	return code;
}

void printBinary(int len, int data[])
{
	for(int i = 0; i < len; i++)
	{
		printf("%d", data[i]);
	}
	printf("\n");
}

int main()
{
	int data[100], n_data[200];
	int n_len, r, i, count, position, e_index = 4;
	char tmp_data[100], code[200];

	int len, server_fd, new_socket, n;
	struct sockaddr_in server_addr, client_addr;
	char buffer[1024];

	printf("Input data: ");
	scanf("%s", tmp_data);

	// convert string to integer array.
	for(i = 0; i < strlen(tmp_data); i++)
	{
		data[i] = tmp_data[i] - 48;
	}

	// find the number of reduntant bits.
	for(i = 1; i < max_r; i++)
	{
		if((int)pow(2, i) >= (strlen(tmp_data) + i + 1))
		{
			r = i;
			break;
		}
	}

	printf("Number of redundant bits needed is: %d\n", r);

	// length of new data array.
	n_len = strlen(tmp_data) + r;

	// filling the reduntant bits as -1.
	for(i = 0; i < r; i++)
```

```
{
        n_data[(int)pow(2, i) - 1] = -1;
}

// filling the message bits in the reverse order.
count = 0;
for(i = n_len-1; i >= 0; i--)
{
        if(n_data[i] != -1)
        {
                n_data[i] = data[count++];
        }
}

// filling the reduntant bits with correct parity.
for(i = 0; i < r; i++)
{
        position = (int)pow(2, i);
        n_data[position - 1] = ham_calc(position, n_len, n_data);
}

// reversing the data bits.
for(i = 0; i < n_len/2; i++)
{
        int temp = n_data[i];
        n_data[i] = n_data[(n_len-1) - i];
        n_data[(n_len-1) - i] = temp;
}

strcpy(code, convertToString(n_len, n_data));

printf("Data with redundant bits: %s\n", code);

// introducing error in the code.
if(n_data[e_index] == 1)
        n_data[e_index] = 0;
else
        n_data[e_index] = 1;

strcpy(code, convertToString(n_len, n_data));

printf("Introduce error in data: %s\n", code);

//---------------------------------------------------------------------
if((server_fd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
{
        perror("Socker error");
}

bzero(&server_addr,sizeof(server_addr));

server_addr.sin_family = AF_INET;
server_addr.sin_addr.s_addr = INADDR_ANY;
server_addr.sin_port = htons(8080);
```

```
	if(bind(server_fd, (struct sockaddr*)&server_addr, sizeof(server_addr)) < 0)
	{
		perror("Bind error");
	}

	if(listen(server_fd,2) < 0)
	{
		perror("Listen error");
	}

	len = sizeof(client_addr);

	if((new_socket = accept(server_fd, (struct sockaddr*)&client_addr, &len)) < 0)
	{
		perror("Accept error");
	}

	strcpy(buffer, code);

	write(new_socket, buffer, sizeof(buffer));

	close(server_fd);
	close(new_socket);
	//---------------------------------------------------------------------

	return 0;
}
```

Program for Client:

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


int ham_calc(int position,int len, int data[])
{
	int count=0, i, j;
	i = position - 1;
	while(i < len)
	{
		for(j = i; j < i+position && j < len; j++)
		{
			if(data[j] == 1)
			{
				count++;
			}
		}
```

```
                i = i + 2*position;
        }
        if(count%2 == 0)
        {
                return 0;
        }
        else
        {
                return 1;
        }
}

void printBinary(int len, int data[])
{
        for(int i = 0; i < len; i++)
        {
                printf("%d", data[i]);
        }
        printf("\n");
}

int main()
{
        int data[200], c_data[100], rbits[20];
        int len, r, i, count, position, e_index;
        char code[200];

        //--------------------------------------------------------------------------------
        int socket_fd;
        struct sockaddr_in server_addr;
        char buffer[1024];

        if((socket_fd=socket(AF_INET, SOCK_STREAM, 0)) < 0)
        {
                perror("Socket error");
        }

        bzero(&server_addr,sizeof(server_addr));

        server_addr.sin_family = AF_INET;
        server_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
        server_addr.sin_port = htons(8080);

        if(connect(socket_fd,(struct sockaddr*)&server_addr, sizeof(server_addr)) < 0)
        {
                perror("Connect error");
        }

        read(socket_fd, buffer, sizeof(buffer));


        close(socket_fd);
        //--------------------------------------------------------------------------------
```

```
strcpy(code, buffer);

len = strlen(code);

// convert string to integer array.
for(i = 0; i < len; i++)
{
	data[i] = code[i] - 48;
}

printf("Data received: ");
printBinary(len, data);

// reversing the data array.
for(i = 0; i < len/2; i++)
{
	int temp = data[i];
	data[i] = data[(len-1) - i];
	data[(len-1) - i] = temp;
}

// find the number of redundant bits.
i = 0;
while((int)pow(2, i) < (len + i + 1))
{
	i++;
}
r = i;

// extracting the redundant bits and placing it in separate array.
for(i = 0; i < r; i++)
{
	position = (int)pow(2, i);
	rbits[i] = ham_calc(position, len, data);
}

// calculating the value of the rbits array.
// binary to decimal conversion.
e_index = 0;
for(i = 0; i < r; i++)
{
	if(rbits[i] == 1)
	{
		e_index += (int)pow(2, i);
	}
}

// reversing the rbits array.
for(i = 0; i < r/2; i++)
{
	int temp = rbits[i];
	rbits[i] = rbits[(r-1) - i];
	rbits[(r-1) - i] = temp;
}
```

```
	printf("Calculated redundant bits: ");
	printBinary(r, rbits);

	// correcting the data.
	e_index--;
	if(e_index >= 0)
	{
		if(data[e_index] == 1)
			data[e_index] = 0;
		else
			data[e_index] = 1;
	}

	// assigning the rbits as -1.
	for(i = 0; i < r; i++)
	{
		position = (int)pow(2, i);
		data[position-1] = -1;
	}

	count = 0;
	for(i = 0; i < len; i++)
	{
		if(data[i] != -1)
		{
			c_data[count++] = data[i];
		}
	}

	printf("Corrected data: ");
	printBinary(len - r, c_data);

	return 0;
}
```

Screenshot for Server:

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

#define max_r 20

int ham_calc(int position,int len, int data[])
{
        int count=0, i, j;
        i = position - 1;
        while(i < len)
        {
                for(j = i; j < i+position; j++)
                {
                        if(data[j] == 1)
                                count++;
                }
                i = i + 2*position;
        }
        if(count%2 == 0)
                return 0;
        else
                return 1;
}

char* convertToString(int n_len, int e_data[])
{
        char *code = malloc(n_len + 1);
        for(int i = 0; i < n_len; i++)
        {
                code[i] = e_data[i] + '0';
        }
        return code;
}

void printBinary(int len, int data[])
{
        for(int i = 0; i < len; i++)
        {
                printf("%d", data[i]);
        }
        printf("\n");
}

int main()
{
        int data[100], n_data[200];
        int n_len, r, i, count, position, e_index = 4;
```

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

#define max_r 20

int ham_calc(int position,int len, int data[])
{
        int count=0, i, j;
        i = position - 1;
        while(i < len)
        {
                for(j = i; j < i+position; j++)
                {
                        if(data[j] == 1)
                                count++;
                }
                i = i + 2*position;
        }
        if(count%2 == 0)
                return 0;
        else
                return 1;
}

char* convertToString(int n_len, int e_data[])
{
        char *code = malloc(n_len + 1);
        for(int i = 0; i < n_len; i++)
        {
                code[i] = e_data[i] + '0';
        }
        return code;
}

void printBinary(int len, int data[])
{
        for(int i = 0; i < len; i++)
        {
                printf("%d", data[i]);
        }
        printf("\n");
}

int main()
{
        int data[100], n_data[200];
        int n_len, r, i, count, position, e_index = 4;
```

```
                n_data[(n_len-1) - i] = temp;
        }

        strcpy(code, convertToString(n_len, n_data));

        printf("Data with redundant bits: %s\n", code);

        // introducing error in the code.
        if(n_data[e_index] == 1)
                n_data[e_index] = 0;
        else
                n_data[e_index] = 1;

        strcpy(code, convertToString(n_len, n_data));

        printf("Introduce error in data: %s\n", code);

        //-------------------------------------------------------------------------
        if((server_fd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
        {
                perror("Socker error");
        }

        bzero(&server_addr,sizeof(server_addr));

        server_addr.sin_family = AF_INET;
        server_addr.sin_addr.s_addr = INADDR_ANY;
        server_addr.sin_port = htons(8080);

        if(bind(server_fd, (struct sockaddr*)&server_addr, sizeof(server_addr)) < 0)
        {
                perror("Bind error");
        }

        if(listen(server_fd,2) < 0)
        {
                perror("Listen error");
        }

        len = sizeof(client_addr);

        if((new_socket = accept(server_fd, (struct sockaddr*)&client_addr, &len)) < 0)
        {
                perror("Accept error");
        }

        strcpy(buffer, code);

        write(new_socket, buffer, sizeof(buffer));

        close(server_fd);
        close(new_socket);
```

Screenshot for Client:

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


int ham_calc(int position,int len, int data[])
{
        int count=0, i, j;
        i = position - 1;
        while(i < len)
        {
                for(j = i; j < i+position && j < len; j++)
                {
                        if(data[j] == 1)
                        {
                                count++;
                        }
                }
                i = i + 2*position;
        }
        if(count%2 == 0)
        {
                return 0;
        }
        else
        {
                return 1;
        }
}

void printBinary(int len, int data[])
{
        for(int i = 0; i < len; i++)
        {
                printf("%d", data[i]);
        }
        printf("\n");
}

int main()
{
        int data[200], c_data[100], rbits[20];
        int len, r, i, count, position, e_index;
        char code[200];

        //--------------------------------------------------------------------------------
        int socket_fd;
```

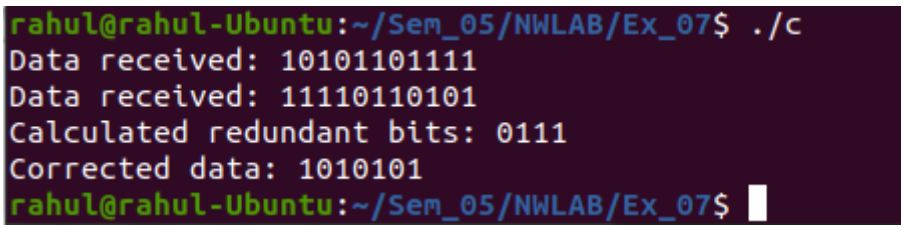
```
        char buffer[1024];

        if((socket_fd=socket(AF_INET, SOCK_STREAM, 0)) < 0)
        {
                perror("Socket error");
        }

        bzero(&server_addr,sizeof(server_addr));

        server_addr.sin_family = AF_INET;
        server_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
        server_addr.sin_port = htons(8080);

        if(connect(socket_fd,(struct sockaddr*)&server_addr, sizeof(server_addr)) < 0)
        {
                perror("Connect error");
        }

        read(socket_fd, buffer, sizeof(buffer));


        close(socket_fd);
        //--------------------------------------------------------------------------------

        strcpy(code, buffer);

        len = strlen(code);

        // convert string to integer array.
        for(i = 0; i < len; i++)
        {
                data[i] = code[i] - 48;
        }

        printf("Data received: ");
        printBinary(len, data);

        // reversing the data array.
        for(i = 0; i < len/2; i++)
        {
                int temp = data[i];
                data[i] = data[(len-1) - i];
                data[(len-1) - i] = temp;
        }

        // find the number of redundant bits.
        i = 0;
        while((int)pow(2, i) < (len + i + 1))
        {
                i++;
        }
        r = i;
```

```
        // binary to decimal conversion.
        e_index = 0;
        for(i = 0; i < r; i++)
        {
                if(rbits[i] == 1)
                {
                        e_index += (int)pow(2, i);
                }
        }

        // reversing the rbits array.
        for(i = 0; i < r/2; i++)
        {
                int temp = rbits[i];
                rbits[i] = rbits[(r-1) - i];
                rbits[(r-1) - i] = temp;
        }

        printf("Calculated redundant bits: ");
        printBinary(r, rbits);

        // correcting the data.
        e_index--;
        if(e_index >= 0)
        {
                if(data[e_index] == 1)
                        data[e_index] = 0;
                else
                        data[e_index] = 1;
        }

        // assigning the rbits as -1.
        for(i = 0; i < r; i++)
        {
                position = (int)pow(2, i);
                data[position-1] = -1;
        }

        count = 0;
        for(i = 0; i < len; i++)
        {
                if(data[i] != -1)
                {
                        c_data[count++] = data[i];
                }
        }

        printf("Corrected data: ");
        printBinary(len - r, c_data);

        return 0;
}
```

Server Output:

Client Output:

```
rahul@rahul-Ubuntu:~/Sem_05/NWLAB/Ex_07$ ./s
Input data: 1010101
Number of redundant bits needed is: 4
Data with redundant bits: 10100101111
Introduce error in data: 10101101111
rahul@rahul-Ubuntu:~/Sem_05/NWLAB/Ex_07$
```



Learning Outcomes:

This assignment helped me to

1. Write program for server and client with socket programming.
2. Understand various functions involved in creating, establishing, maintaining, Sending, receiving and terminating the connection between the server and client.
3. Write code to make server and client communicate with each other using read() and write() functions.
4. Understand the concepts of hamming codes and how it is used for error detection and correction.
5. Implement hamming codes in c along with Socket programming.