**CRC :**

#include <iostream>

#include <cstring> // For strcpy, strcat, strlen

using namespace std;

int crc(char \*ip, char \*op, char \*poly, int mode) {

strcpy(op, ip); // Copy input to output

if (mode) {

// Append (length of polynomial - 1) zeros to the input

for (int i = 1; i < strlen(poly); i++)

strcat(op, "0");

}

int n = strlen(poly); // Length of the polynomial

for (int i = 0; i <= strlen(op) - n; i++) {

if (op[i] == '1') { // Perform XOR only when the bit is 1

for (int j = 0; j < n; j++) {

// XOR operation

op[i + j] = (op[i + j] == poly[j]) ? '0' : '1';

}

}

}

// Check for errors (remaining part after CRC division)

for (int i = strlen(op) - strlen(poly) + 1; i < strlen(op); i++) {

if (op[i] == '1') return 0; // Error detected

}

return 1; // No error

}

int main() {

char ip[50], op[100], recv[50];

char poly[] = "10111"; // CRC polynomial

cout << "Enter the input message in binary:" << endl;

cin >> ip;

// Generate CRC and transmitted message

crc(ip, op, poly, 1);

cout << "The transmitted message is: " << ip << string(op + strlen(ip)) << endl;

cout << "Enter the received message in binary:" << endl;

cin >> recv;

// Check for errors in the received message

if (crc(recv, op, poly, 0))

cout << "No error in data" << endl;

else

cout << "Error in data transmission has occurred" << endl;

return 0;

}



**LEAKAGE BUCKET ALGORITHM :**

#include <bits/stdc++.h>

using namespace std;

int main()

{

int no\_of\_queries, storage, output\_pkt\_size;

int input\_pkt\_size, bucket\_size, size\_left;

// initial packets in the bucket

storage = 0;

// total no. of times bucket content is checked

no\_of\_queries = 4;

// total no. of packets that can

// be accommodated in the bucket

bucket\_size = 10;

// no. of packets that enters the bucket at a time

input\_pkt\_size = 4;

// no. of packets that exits the bucket at a time

output\_pkt\_size = 1;

for (int i = 0; i < no\_of\_queries; i++) // space left

{

size\_left = bucket\_size - storage;

if (input\_pkt\_size <= size\_left) {

// update storage

storage += input\_pkt\_size;

}

else {

printf("Packet loss = %d\n", input\_pkt\_size);

}

printf("Buffer size= %d out of bucket size= %d\n",

storage, bucket\_size);

storage -= output\_pkt\_size;

}

return 0;

}

