**BIJAY REGMI(210913032)**

**WEEK 5**

DATE - 20/12/2021

1. **HAMMING CODE**

#include<stdio.h>

int powerOf(int base, int e){

int prod = 1;

while(e != 0){

prod = prod\*base;

e--;

}

return prod;

}

int checkPowerOfTwo(int num){

int i = 0, powTwo;

while(i!=100){

powTwo = (int)powerOf(2,i);

if(num == powTwo)

return 1;

else if(num < powTwo)

return 0;

i++;

}

return 0;

}

int calculateRedudentBits(int n){

int r=1,temp;

while(r!=100){

temp = (int)powerOf(2,r);

if(temp>=(n+r+1))

return r;

r++;

}

return 0;

}

int main(){

char s1[100];

int N=0, R, i, j, k, l, parity;

printf("\n\nSENDING SITE");

printf("\n--------------\n\n");

printf("\nEnter data to send : ");

gets(s1);

while(s1[N] != '\0')

N++;

// Calculating Redundent Bits

R = calculateRedudentBits(N);

printf("\n\nRedundent Bits : %d", R);

int data[N], dataWord[N+R];

// Converting string data to int data in REVERSE

i = N-1;

j=0;

while(j != N){

data[i] = s1[j]-'0';

i--; j++;

}

// Creating DataWord in REVERSE

j=0;

for(i=0; i < N+R; i++){

if(checkPowerOfTwo(i+1))

continue;

dataWord[i]=data[j];

j++;

}

printf("\nParity Bits : ");

for(i=0; i < R; i++){ //Looping for Redundent Bits

j = powerOf(2,i);

parity = 0;

for(k=j-1; k<N+R ; k+=(2\*j)){ //Skipping j number of bits

l = k;

while(l<k+j && l<N+R){ //considering even parity of l number of bits

if(l != j-1)

parity = parity^dataWord[l];

l++;

}

}

dataWord[j-1] = parity;

printf("\t Dataword[%d] : %d",j,dataWord[j-1]);

}

// Reversing Dataword to original form.

printf("\nDATAWORD : ");

i=N+R-1;

while(i>=0){

printf("%d",dataWord[i]);

i--;

}

printf("\n\n\nRECIEVING SITE");

printf("\n--------------\n\n");

char s2[100];

int n;

// Recieving Dataword and checking whether the dataword has same length on sending site or not.

while(n != N+R){

printf("\nEnter data Recieved : ");

gets(s2);

n=0;

while(s2[n] != '\0')

n++;

if(n != N+R)

printf("\nERROR !\nDataword Recieved length is not equal to dataword sent. \n");

}

int dataWordRecieved[100],rbits[100],rbitsLength;

printf("\nDataword Recieved : ");

// Converting Recieved string dataWord to int dataWord in REVERSE

i = n-1;

j=0;

while(j != n){

dataWordRecieved[i] = s2[j]-'0';

printf("%d",dataWordRecieved[i]);

i--; j++;

}

// Finding all redundent bits from dataWordRecieved[100] and storing it in rbits[100]

j=0;

for(i=0; i < n; i++){

if(checkPowerOfTwo(i+1))

j++;

}

rbitsLength = j;

// Looping through redundent bits to find error bits

// Checking Parity for every redundent bits and storing parity in rbits[100]

for(i=0; i < rbitsLength; i++){ //Looping for Redundent Bits

j = powerOf(2,i);

parity = 0;

for(k=j-1; k<n ; k+=(2\*j)){ //Skipping j number of bits

l = k;

while(l<k+j && l<n){ //considering even parity of l number of bits

parity = parity^dataWordRecieved[l];

l++;

}

}

rbits[i] = parity;

}

// Checking whether all redundent bits are 0 or not;

// If not 0, the calculating position

printf("\nRedundent bits Recieved : ");

j=rbitsLength-1;

int position = 0;

while(j>=0){

printf("%d",rbits[j]);

if(rbits[j] == 1){

position = position + powerOf(2,j);

}

j--;

}

if(position == 0){

printf("\nThere is no error in recieved Dataword ");

for(i=0; i<n; i++)

printf("%c",s2[i]);

}

else{

printf("\nThere is error in position %d",position);

if(dataWordRecieved[position-1] == 0)

dataWordRecieved[position-1] = 1;

else

dataWordRecieved[position-1] = 0;

printf("\nCORRECTED DATAWORD : ");

i = n-1;

while(i>=0){

printf("%d",dataWordRecieved[i]);

i--;

}

}

// Printing Recieved Data

printf("\n\nRECIEVED DATA : ");

for(i=n-1; i >= 0; i--){

if(checkPowerOfTwo(i+1))

continue;

printf("%d",dataWordRecieved[i]);

}

printf("\n\n");

return 0;

}

**INPUT/OUTPUT**

**Text

Description automatically generated**

1. **FIFO Packet Scheduling Mechanism**

#include<stdio.h>

#include<stdlib.h>

# define MAX 3

// Structure of Queue

typedef struct Queue{

int data;

int priority;

struct Queue \*left, \*right;

}queue;

queue \*head = NULL;

queue \*tail = NULL;

queue \*minPriority = NULL;

int queueSize = 0;

// Creates a new Node

queue\* createNode(int data, int priority){

queue \*newNode = (queue\*)malloc(sizeof(queue));

newNode->data = data;

newNode->priority = priority;

newNode->left = NULL;

newNode->right = NULL;

return newNode;

}

// Checks Whether Queue is Empty Or Not

int isQueueEmpty(){

if(head == NULL && tail == NULL){

printf("\nERROR ! Queue is Empty.");

return 1;

}

return 0;

}

// Finds next Min Priority

queue\* findMinPriority(){

if(head != NULL && tail != NULL){

queue \*temp = tail->right, \*minPriority=tail;

while (temp){

if(minPriority->priority > temp->priority)

minPriority = temp;

temp=temp->right;

}

return minPriority;

}

return NULL;

}

// Discard min priority and adds upcoming

void priorityDiscardPolicy(queue \*newNode ){

// Removing Min Priority

if(minPriority->left)

minPriority->left->right = minPriority->right;

if(minPriority->right)

minPriority->right->left = minPriority->left;

printf("\nPacket %d(%d) is removed, Since it has got low priority.",minPriority->data,minPriority->priority);

queue \*temp = minPriority;

free(temp);

minPriority = findMinPriority();

// Adding NewNode to tail

if(minPriority->priority > newNode->priority)

minPriority = newNode;

newNode->right = tail;

tail->left = newNode;

tail = tail->left;

printf("\nPacket %d(%d) is added in Queue.",tail->data,tail->priority);

}

// Enter value in tail

void enqueue(int data, int priority){

queue \*newNode = createNode(data,priority);

if(head == NULL && tail == NULL){

head = newNode;

tail = newNode;

minPriority = newNode;

queueSize ++;

printf("\nPacket %d(%d) is first data in Queue.",tail->data,tail->priority);

return;

}

if(queueSize == MAX){

int ch;

printf("\n\nQUEUE OVERFLOW!!\nPACKET DISCARD POLICY\n1. Priority\n2. Tail Drop\nEnter Choice : ");

scanf("%d",&ch);

if(ch==1){

priorityDiscardPolicy(newNode);

return;

}

else if(ch == 2){

printf("\nPacket %d(%d) has dropped, since its on Tail.",newNode->data,newNode->priority);

free(newNode);

return;

}

}

if(minPriority->priority > newNode->priority)

minPriority = newNode;

newNode->right = tail;

tail->left = newNode;

tail = tail->left;

queueSize ++;

printf("\nPacket %d(%d) is added in Queue.",tail->data,tail->priority);

}

// Delete value from head

void dequeue(){

if(!isQueueEmpty()){

queue \*temp = head;

if(head == tail){

head = NULL;

tail = NULL;

}

else{

head = head->left;

head->right = NULL;

}

if(minPriority == temp )

minPriority = findMinPriority();

printf("\nPacket %d(%d) is removed !",temp->data,temp->priority);

queueSize -- ;

free(temp);

}

}

// Displays queue

void display(){

queue \*temp=tail;

printf("\n\nQUEUE : ");

while (temp){

printf("%d(%d)\t",temp->data,temp->priority);

temp = temp->right;

}

printf("\nQueueSize : %d",queueSize);

}

int main(){

printf("\n\n");

int ch=0, data, priority;

printf("\nQUEUE PACKET SCHEDULING\n\n");

while(ch != 5){

printf("\n\n1. Insert Packet\n2. Remove Packet\n3. Display all Packets in Queue\n4. Exit\n\nEnter Your Choice : ");

scanf("%d",&ch);

if(ch == 1){

printf("\nEnter Packet Number and its Priority : ");

scanf("%d", &data);

scanf("%d", &priority);

enqueue(data, priority);

}

else if(ch==2)

dequeue();

else if(ch ==3)

display();

else if(ch == 4)

break;

else

printf("\n\nERROR ! Invalid Input.");

}

printf("\n\n");

return 0;

}

**INPUT/OUTPUT**

Text

Description automatically generated

Text

Description automatically generatedA picture containing shape

Description automatically generated