

Lab Manual**1. Implement the data link layer framing methods such as character stuffing and bit stuffing****Character stuffing**

THEORY: The framing method gets around the problem of resynchronization after an error by having each frame start with the ASCII character sequence DLE STX and the sequence DLE ETX. If the destination ever loses the track of the frame boundaries all it has to do is look for DLE STX or DLE ETX characters to figure out. The data link layer on the receiving end removes the DLE before the data are given to the network layer. This technique is called character stuffing .

ALGORITHM:

Step 1:Read a character string

Step 2:Read a character for starting & ending delimiter

Step 3:Trace a character in the string

Step 4:If character is same as starting or ending delimiter, then Add the same character next to it i.e. stuff a character

Step 5:Repeat 3 & 4 until end of string

Step 6:Display stuffed string

Source Code:

```
#include<stdio.h>

#include<conio.h>

#include<string.h>

int main()

{

    char a[100],b[100],c[100];

    int i,j=0,n,k=0;

    printf("Enter the data");

    gets(a);

    n=strlen(a);

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

    {

        if((a[i] == 'd' && a[i+1]=='l' && a[i+2]=='e')||
```

```
(a[i] == 'e' && a[i+1] == 's' && a[i+2] == 'c'))
```

```
{
```

```
    c[k++] = 'e';
```

```
    c[k++] = 's';
```

```
    c[k++] = 'c';
```

```
}
```

```
    c[k++] = a[i];
```

```
}
```

```
c[k++] = '\0';
```

```
b[j++] = '\0';
```

```
printf("%s\n",b);
```

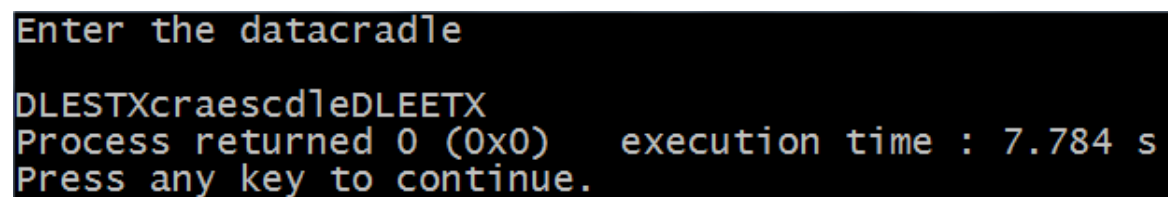
```
printf("DLESTX");
```

```
printf("%s",c);
```

```
printf("DLEETX");
```

```
return 0;
```

```
}
```

OUTPUT:

```
Enter the datacradle
DLESTXcraescdleDLEETX
Process returned 0 (0x0)    execution time : 7.784 s
Press any key to continue.
_
```

Bit Stuffing

THEORY: The technique allows data frames to contain an arbitrary number of bits . Each frame begins and ends with special bit pattern , 01111110, called a flag byte. When ever the sender's data link layer encounters five consecutive one's in the data, it automatically stuffs a 0 bit into the outgoing bit stream.

ALGORITHM:

Step 1:Read a bit string

Step 2:Trace 5 bits in the string

Step 3:Check if 5 bits are 11111 then insert a '0' bit after it

Step 4:Repeat 2 & 3 until end of string

Step 5:Display stuffed bit string

Source Code:

```
#include<stdio.h>

#include<conio.h>

void main()

{

int a[15];

int i,j,k,n,c=0,pos=0;

printf("\n enter the no of bits");

scanf("%d",&n);

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

scanf("%d",&a[i]);

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

{

if(a[i]==1)

{

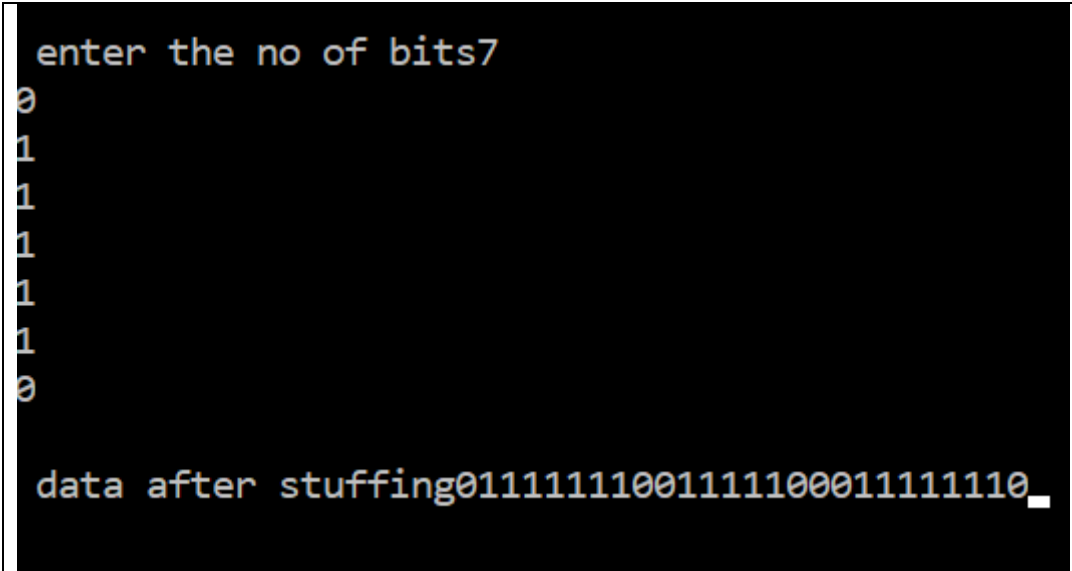
c++;

if(c==5)

{
```

```
pos=i+1;
c=0;
for(j=n;j>=pos;j--)
{
k=j+1;
a[k]=a[j];
}
a[pos]=0;
n=n+1;
}
}
else
c=0;
}

printf("\n data after stuffing");
printf("011111110");
for(i=0;i<n;i++)
{
printf("%d",a[i]) ;
}
printf("011111110");
getch();
}
```

OUTPUT:A screenshot of a terminal window with a black background and white text. The text shows a prompt 'enter the no of bits7' followed by a sequence of bits: 0, 1, 1, 1, 1, 1, 1, 0. Below this, it says 'data after stuffing' followed by the bit sequence '0111111100111110001111110' and a cursor at the end.

```
enter the no of bits7
0
1
1
1
1
1
1
0

data after stuffing0111111100111110001111110_
```

VIVA-VOCE**1.what is bit stuffing? what is the use of bit stuffing?**

A)Bit stuffing is the process of inserting noninformation bits into data to break up bit patterns to affect the synchronous transmission of information. Bit stuffing is commonly used to bring bit streams up to a common transmission rate or to fill frames.

2.What is character stuffing? What is the use of character stuffing?

A)In byte stuffing (or character stuffing), a special byte is added to the data section of the frame when there is a character with the same pattern as the flag. The data section is stuffed with an extra byte. Each frame starts with the ASCII character sequence DLE STX and ends with the sequence DLE ETX.(where DLE is Data Link Escape, STX is Start of TeXt and ETX is End of TeXt.) This method overcomes the drawbacks of the character count method. If the destination ever loses synchronization, it only has to look for DLE STX and DLE ETX characters.

3.By which special bit pattern the frame begins and ends?

A)Each frame begins and ends with a special bit pattern called a flag byte [01111110].

4.What are the functions of data link layer?

A)1)It Provides well-defined service interface to the network layer on source machine to the network layer on destination machine.

2) The source machine sends data in blocks called frames to the destination machine. The starting and ending of each frame should be recognised by the destination machine.

3) The source machine must not send data frames at a rate faster than the destination machine can accept them

5.Name the delimiters for character stuffing?

A)Each frame starts with the ASCII character sequence DLE STX and ends with the sequence DLE ETX

6.Expand DLE STX and DLE ETX?

A)DLE is Data Link Escape, STX is Start of TeXt

DLE is Data Link Escape, ETX is End of TeXt.

2. Implement CRC on a data set of characters using CRC polynomials (CRC 12/ CRC 16/ CRC CCIP).

THEORY: CRC method can detect a single burst of length n , since only one bit per column will be changed, a burst of length $n+1$ will pass undetected, if the first bit is inverted, the last bit is inverted and all other bits are correct. If the block is badly garbled by a long burst or by multiple shorter burst, the probability that any of the n columns will have the correct parity that is 0.5. so the probability of a bad block being expected when it should not be $2^{\text{power}(-n)}$. This scheme some times known as Cyclic Redundancy Code.

ALGORITHM**Implementation of CRC – 12**

- Step 1: Read values for transmitted data
- Step 2: Compute polynomial for CRC-12
- Step 3: Concatenate twelve 0's at the end of transmitted data
- Step 4: Perform binary division where CRC-12 generating polynomial is the divisor and dividend is the concatenated string.
- Step 5: Repeat the step 4 until length of remainder = 12
- Step 6: If remainder is less than 12, then add required number of zero's to the beginning of the remainder such that length is 12.
- Step 7: Read received data
- Step 8: Append CRC with received data
- Step 9: Perform binary division on this with the divisor as generating polynomial
- Step10: Repeat step 9 until remainder= 0
- Step 11: If remainder= 0 then print message “ data transmitted correctly”
- Step12: Else print message “data transmitted incorrectly”

Implementation of CRC – 16

- Step 1: Read values for transmitted data
- Step 2: Compute polynomial for CRC-16
- Step 3: Concatenate sixteen 0's at the end of transmitted data
- Step 4: Perform binary division where CRC-16 generating polynomial is the divisor and dividend is the concatenated string.
- Step 5: Repeat the step 4 until length of remainder = 16
- Step 6: If remainder is less than 16, then add required number of zero's to the beginning of the remainder such that length is 16.
- Step 7: Read received data
- Step 8: Append CRC with received data
- Step 9: Perform binary division on this with the divisor as generating polynomial
- Step10: Repeat step 9 until remainder= 0
- Step 11: If remainder= 0 then print message “ data transmitted correctly”
- Step12: Else print message “data transmitted incorrectly”

SOURCE CODE:

```
#include <stdio.h>

#include <conio.h>

#include <string.h>

void main() {

    int i,j,keylen,msglen;

    char input[100], key[30],temp[30],quot[100],rem[30],key1[30];

    printf("Enter Data: ");

    gets(input);

    printf("Enter Key: ");

    gets(key);

    keylen=strlen(key);

    msglen=strlen(input);

    strcpy(key1,key);

    for (i=0;i<keylen-1;i++) {

        input[msglen+i]='0';

    }

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

        temp[i]=input[i];

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

        quot[i]=temp[0];

        if(quot[i]=='0')

            for (j=0;j<keylen;j++)

                key[j]='0'; else

            for (j=0;j<keylen;j++)
```



```
        key[j]=key1[j];
        for (j=keylen-1;j>0;j--) {
            if(temp[j]==key[j])
                rem[j-1]='0'; else
                rem[j-1]='1';
        }
        rem[keylen-1]=input[i+keylen];
        strcpy(temp,rem);
    }
    strcpy(rem,temp);
    printf("\nQuotient is ");
    for (i=0;i<msglen;i++)
        printf("%c",quot[i]);
    printf("\nRemainder is ");
    for (i=0;i<keylen-1;i++)
        printf("%c",rem[i]);
    printf("\nFinal data is: ");
    for (i=0;i<msglen;i++)
        printf("%c",input[i]);
    for (i=0;i<keylen-1;i++)
        printf("%c",rem[i]);
    getch();
}
```

OUTPUT:

```

Enter Data: 110001100011
Enter Key: 1101

Quotient is 100110011111
Remainder is 011
Final data is: 110001100011011_

```

VIVA-VOCE**1.What is CRC?**

A cyclic redundancy check (CRC) is an error-detecting code commonly used in digital networks and storage devices to detect accidental changes to raw data. Blocks of data entering these systems get a short check value attached, based on the remainder of a polynomial division of their contents. On retrieval, the calculation is repeated and, in the event the check values do not match, corrective action can be taken against data corruption. CRCs can be used for error correction .

2.What is the use of CRC?

The use of cyclic codes, which encode messages by adding a fixed-length check value, for the purpose of error detection in communication networks.

3.Name the CRC standards for generator polynomial?

CRC-12: $x^{12}+x^{11}+x^3+x^2+x+1$

CRC- 16: $x^{16}+x^{15}+x^2+1$

CRC-CCITT: $x^{16}+x^{12}+x^5+1$

4.How do you convert generator polynomial into binary form?

The generator polynomial is converted into the binary form by considering coefficients of polynomial

EX: $x^{12}+x^{11}+x^3+x^2+x+1 = 1100000001111$

5. Define checksum?

A checksum is a simple type of redundancy check that is used to detect errors in data

6. How do you perform binary division operation in CRC?

```

                                10101100
                                -----
11011 ) 111001010000
        11011
        -----
          01111010000
            11011
            -----
              010110000
                11011
                -----
                  1101000
                    11011
                    -----
                      000100

```

3.Implement Dijkstra's algorithm to compute the Shortest path through a graph.

THEORY: Build a graph of the subnet, with each node representing a router and each arc of the graph a communication link.

The labels on arc could be computed as a function of the delay. Initially, no paths are known, so all nodes are labeled with infinity, as algorithm proceeds and paths are found, labels may change, reflecting better paths. Initially all labels are tentative. When it is discovered, that a label represent the shortest possible path from the source to that node, it is made permanent and never changed thereafter.

Each node is labeled with its distance from the source node along the best known path. This distance will be shortest from source to destination.

ALGORITHM:

Step 1: Plot the subnet, assign weights to each of the edges between nodes.

Step 2: Using the metrics as distance in km calculate the shortest path from the given source to destination.

Step 3: Initially mark all the paths from source as infinity.

Step 4: Starting with the source node check the adjacent nodes for shortest path, and mark them as tentative nodes.

Step 5: From these tentative nodes, select one which is having short distance from source, and mark as permanent.

Step 6: Distance from source to that tentative node should be recorded.

Step 7: Now this node is considered as source node and repeat the steps from 3 to 6.

Step 8: Repeat the steps along the path with the distances being added throughout the path to reach the destination.

SOURCE CODE:

```
#include <stdio.h>

#define infinity 9999

#define MAX 20

int minimum(int a,int b)
{
    if(a<=b)
        return a;
    else
        return b;
}

main()
```

```
{  
  
    int i,j,k,n,start,end,adj[MAX][MAX],path[MAX][MAX];  
  
    printf("Enter number of vertices : ");  
  
    scanf("%d",&n);  
  
    printf("Enter weighted matrix :\n");  
  
    for(i=0;i<n;i++)  
        for(j=0;j<n;j++)  
            scanf("%d",&adj[i][j]);  
  
    for(i=0;i<n;i++)  
        for(j=0;j<n;j++)  
            if(adj[i][j]==0)  
                path[i][j]=infinity;  
            else  
                path[i][j]=adj[i][j];  
  
    for(k=0;k<n;k++)  
    {  
        for(i=0;i<n;i++)  
            for(j=0;j<n;j++){  
  
                if(i==j)  
                    path[i][j]=infinity;  
  
                else  
  
                    path[i][j]=minimum(path[i][j],path[i][k]+path[k][j]);  
  
            }  
        }  
  
    printf("Shortest path matrix is :\n");  
  
    for(i=0;i<n;i++)
```

```
        {  
            for(j=0;j<n;j++)  
                printf("%6d",path[i][j]);  
            printf("\n");  
        }  
        printf("Enter start vertex :");  
        scanf("%d",&start);  
        printf("Enter end vertex :");  
        scanf("%d",&end);  
        printf("the min. cost between %d and %d is  
%d",start,end,path[start][end]);  
    }
```

OUTPUT:

```
Enter number of vertices : 4  
Enter weighted matrix :  
0 1 3 0  
1 0 0 4  
3 0 0 4  
0 4 4 0  
Shortest path matrix is :  
9999 1 3 5  
1 9999 4 4  
3 4 9999 4  
5 4 4 9999  
Enter start vertex :1  
Enter end vertex :3  
the min. cost between 1 and 3 is 4  
Process returned 34 (0x22) execution time : 69.346 s  
Press any key to continue.
```

VIVA VOCE:**1. What is Flow based routing algorithm?**

Flow-based routing seeks to find a routing table to minimize the average packet delay through the subnet.

2. What is the Link state routing algorithm?

Link State improves the convergence of Distance Vector by having everybody share their idea of the state of the net with everybody else (more information is available to nodes, so better routing tables can be constructed).

3. In shortest path which metric is considered?

Shortest Path Metric is used in Shortest path Routing Algorithm

4. What is the another name for shortest path algorithm?

Another name for ShortestPath Algorithm is Dijkstra's algorithm

5. What is the disadvantage of Dijkstra's algorithm?

The major disadvantage of the algorithm is the fact that it does a blind search there by consuming a lot of time waste of necessary resources. Another disadvantage is that it cannot handle negative edges. This leads to acyclic graphs and most often cannot obtain the right shortest path.

6. What is the advantage of Dijkstra's algorithm?

- 1) It is used in Google Maps
- 2) It is used in finding Shortest Path.
- 3) It is used in geographical Maps
- 4) To find locations of Map which refers to vertices of graph.
- 5) Distance between the location refers to edges.
- 6) It is used in IP routing to find Open shortest Path First.

4. Take an example subnet graph with weights indicating delay between nodes. Now obtain Routing table at each node using distance vector routing algorithm.

THEORY: This algorithm operates by having each router maintain a table indexed by, and containing one entry for, each router in the subnet. This entry contains two parts: the preferred outgoing line to use for that destination, and an estimate of the time or distance to that destination. The metric used is delay between nodes. These tables are updated by exchanging information with the neighbors for every T seconds for example 30 sec.

ALGORITHM:

- Step 1: Plot the subnet showing the delay between the nodes.
- Step 2: Construct the routing table for each node consisting of delay to reach other nodes in the subnet and the line to be used.
- Step 3: Each router will calculate the delay to reach its adjacent nodes. And this will be recorded in its routing table.
- Step 4: Routing table will be exchanged among the routers for every T seconds.
- Step 5: Each router will wait until it receives an updated table from its neighbors. For example, router A receives routing table from its neighbor X, with X_i as the delay to reach I from X.
- Step 6: With this information, A calculates the new routing table with a delay of $X_i + m$ to reach router I. Where m is a required time for A to reach X.
- Step 7: Router A will now can reach router I via X.
- Step 8: This steps will be repeated for every source to every other destination.
- Step 9: Display routing table of each router.

SOURCE CODE:

```
#include<stdio.h>

struct node
{
    unsigned dist[20];
    unsigned from[20];
}

rt[10];

int main()
{
    int dmat[20][20];
    int n,i,j,k,count=0;
```



```
printf("\nEnter the number of nodes : ");

scanf("%d",&n);

printf("\nEnter the cost matrix :\n");

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

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

{

    scanf("%d",&dmat[i][j]);

    dmat[i][i]=0;

    rt[i].dist[j]=dmat[i][j];

    rt[i].from[j]=j;

}

do

{

    count=0;

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

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

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

    if(rt[i].dist[j]>dmat[i][k]+rt[k].dist[j])

    {

        rt[i].dist[j]=rt[i].dist[k]+rt[k].dist[j];

        rt[i].from[j]=k;

        count++;

    }

}while(count!=0);

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

{

    printf("\n\nState value for router %d is \n",i+1);
```

```
for(j=0;j<n;j++)  
{  
printf("\t\nnode %d via %d Distance%d",j+1,rt[i].from[j]+1,rt[i].dist[j]);  
} }  
printf("\n\n");  
}
```

OUTPUT:

```
Enter the number of nodes : 4  
Enter the cost matrix :  
0 3 5 99  
3 0 99 1  
5 4 0 2  
99 1 2 0  
  
State value for router 1 is  
node 1 via 1 Distance0  
node 2 via 2 Distance3  
node 3 via 3 Distance5  
node 4 via 2 Distance4  
  
State value for router 2 is  
node 1 via 1 Distance3  
node 2 via 2 Distance0  
node 3 via 4 Distance3  
node 4 via 4 Distance1  
  
State value for router 3 is  
node 1 via 1 Distance5  
node 2 via 4 Distance3  
node 3 via 3 Distance0  
node 4 via 4 Distance2  
  
State value for router 4 is  
node 1 via 2 Distance4  
node 2 via 2 Distance1  
node 3 via 3 Distance2  
node 4 via 4 Distance0
```

VIVA –VOCE:**1. What is routing?**

Routing is the process of selecting a path for traffic in a network, or between or across multiple networks.

2. List different routing Algorithms?

Distance Vector (**distance-vector routing**)

link state (**link state routing**)

Path vector (**path-vector routing**).

3. What is routing table? Explain the use of it?

A routing table is a set of rules, often viewed in table format, that is used to determine where data packets traveling over an Internet Protocol (IP) network will be directed. All IP-enabled devices, including routers and switches, use routing tables.

4. What is distance vector routing algorithm?

A routing table contains the information necessary to forward a packet along the best path toward its destination. Each packet contains information about its origin and destination. When a packet is received, a network device examines the packet and matches it to the routing table entry providing the best match for its destination. The table then provides the device with instructions for sending the packet to the next hop on its route across the network.

5. What are the advantages of Distance vector routing?

1.Easy routing table reduction: Distance vector algorithm makes it easy to reduce the routing table as it makes use of less information compared to Link state Algorithm.

2.Simpler to use: easy to implement

6. What are the disadvantages of Distance vector routing?

1.It is slower to converge than Link State.

2.It is at risk from the count-to-infinity problem.

3. It creates more traffic than Link State since a hop count change must be propagated to all routers and processed on each router.

5. Simulate ARP /RARP protocols.

THEORY: In Address Resolution Protocol (ARP), Receiver's MAC address is fetched. Through ARP, (32-bit) IP address mapped into (48-bit) MAC address. Whereas, In Reverse Address Resolution Protocol (RARP), IP address is fetched through server. Through RARP, (48-bit) MAC address of 48 bits mapped into (32-bit) IP address

ALGORITHM:

Step 1: Initialize the string of ip addresses

Step 2: For every ip address, assign an ethernet address Step 3: To Perform ARP, enter the ip address

Step 4: The equivalent ethernet address is displayedStep

Step 5: If the ethernet address is not found, then 'not found' message is displayed Step 6: To Perform RARP, enter the ethernet address

Step 7: The equivalent ip address is displayed

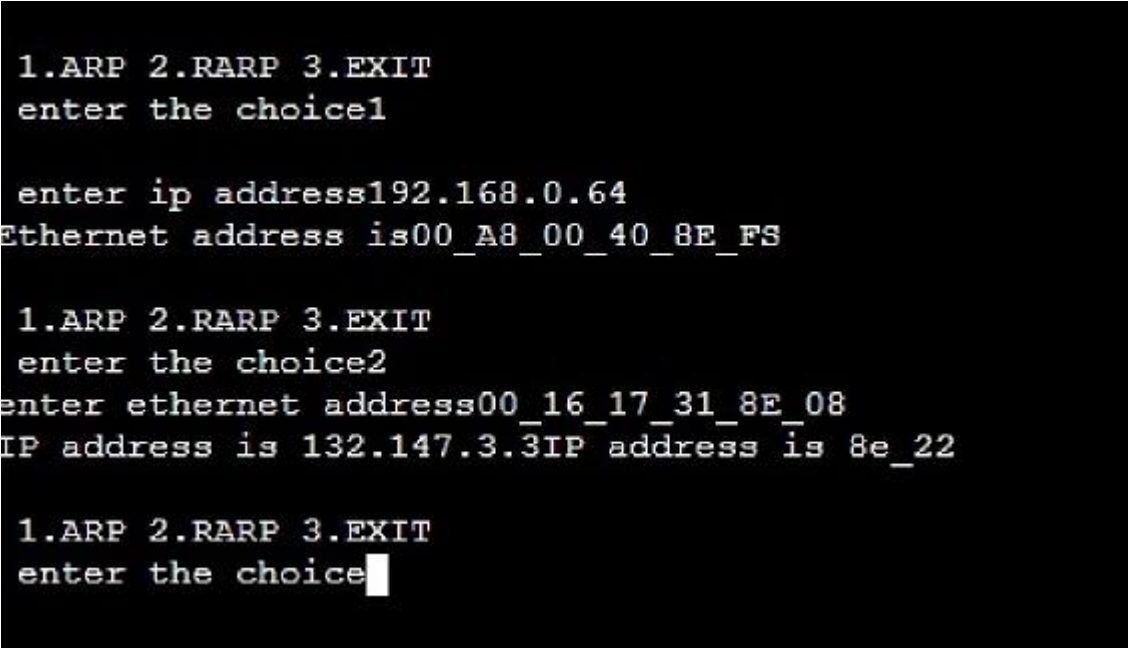
Step 8: If the ip address is not found, then 'not found' message is displayed Step 9: Provide option to perform both ARP and RARP

Step 10: Choose Exit option to terminate the program

SOURCE CODE:

```
#include<stdio.h>
#include<string.h>
#include<conio.h>
main()
{
char ip[10][20]={"192.168.0.64","192.168.0.60","192.168.0.68","132.147.3.3"};
char
et[10][20]={"00_A8_00_40_8E_FS","00_16_17_31_8e_22","00_16_17_31_8E_F7","00_16_17_31_8E_08"};
char ipaddr[20],etaddr[20];
int i,op;
int x=0,y=0;
clrscr();
while(1)
{
printf("\n\n 1.ARP 2.RARP 3.EXIT");
printf("\n enter the choice");
scanf("%d",&op);
switch(op)
{
case 1:
printf("\n enter ip address");
scanf("%s",ipaddr);
for(i=0;i<=20;i++)
{
if(strcmp(ipaddr,ip[i])==0)
{
printf("Ethernet address is%s",et[i]);
x=1;
}
```

```
} }  
if(x==0)  
printf("invalid ip address");  
x=0;  
break;  
case 2:  
printf("enter ethernet address");  
scanf("%s",etaddr);  
for(i=0;i<=20;i++)  
{  
if(strcmp(etaddr,et[i])==0)  
{  
printf("IP address is %s",ip[i]);  
y=1;  
} }  
if(y==0)  
printf("Invalid ethernet address");  
y=0;  
break;  
case 3:  
exit(0);  
}  
}  
}
```

OUTPUT:

```
1.ARP 2.RARP 3.EXIT  
enter the choice1  
  
enter ip address192.168.0.64  
Ethernet address is00_A8_00_40_8E_FS  
  
1.ARP 2.RARP 3.EXIT  
enter the choice2  
enter ethernet address00_16_17_31_8E_08  
IP address is 132.147.3.3IP address is 8e_22  
  
1.ARP 2.RARP 3.EXIT  
enter the choice
```

VIVA - VOCE:**1. What is MAC address?**

A media access control address is a unique identifier assigned to a network interface controller for use as a network address in communications within a network segment. This use is common in most IEEE 802 networking technologies, including Ethernet, Wi-Fi, and Bluetooth.

2. What is IP address?

An Internet Protocol address is a numerical label such as 192.0.2.1 that is connected to a computer network that uses the Internet Protocol for communication. An IP address serves two main functions: host or network interface identification and location addressing.

3. What is Address Resolution Protocol?

The Address Resolution Protocol is a communication protocol used for discovering the link layer address, such as a MAC address, associated with a given internet layer address, typically an IPv4 address. This mapping is a critical function in the Internet protocol suite

4. What is Reverse Address Resolution Protocol?

The Reverse Address Resolution Protocol is an obsolete computer communication protocol used by a client computer to request its Internet Protocol address from a computer network, when all it has available is its link layer or hardware address, such as a MAC address

5. Difference between ARP and RARP?

Through ARP, (32-bit) IP address mapped into (48-bit) MAC address. Whereas through RARP, (48-bit) MAC address of 48 bits mapped into (32-bit) IP address.

6.Implement a Hierarchical routing algorithm

THEORY: Hierarchical Routing is the method of routing in networks that is based on hierarchical addressing. Most transmission control protocol, Internet protocol (DCPIP). It addresses the growth of routing tables. Routers are further divided into regions and they know the route of their own regions only.

ALGORITHM:

Step 1: Start
 Step 2: Read data from user Step 3: For I = 0 To Nodes Step 4: if(I==neigh[n])
 Step 5: add1=topology[src][I];
 add add1 to the distance vector of I, otherwise go to 3
 Step 6: Compare the values in distance vector of the source to that of I .
 Step 7: If the value in distance vector of I is less than the original, then store the new value
 Step 8: Display the distance vector of src
 Step 9: Stop

SOURCE CODE:

```
#include<stdio.h>
#define INFINITY 9999
#define MAX 20
int main()
{
int i,j,k,n,u[10],s[10],d[10][10],adj[20][20],origin,destin,total=0,st;
printf("Enter no.of Regions:");
scanf("%d",&n);
printf("Enter region names:");
for(i=0;i<n;i++)
scanf("%d",&u[i]);
printf("Enter region sizes:");
for(i=0;i<n;i++){
scanf("%d",&s[i]);
total+=s[i];
}
for(i=0;i<n;i++){
printf("Enter node names for region%d:",u[i]);
for(j=0;j<s[i];j++)
scanf("%d",&d[i][j]);
}
for(i=0;i<total;i++)
for(j=0;j<total;j++)
adj[i][j]=0;
printf("Enter edge in source-destination format (0 0 to quit): ");
while(1){
scanf("%d %d", &origin, &destin);
if ((origin == 0) && (destin == 0))
```

```

        break;
    if (origin > total || destin > total || (origin <= 0 & destin <= 0))
        printf("Invalid edge!\n");
    else
        adj[origin][destin] = 1;
    }
    printf("Region\tNode Size\tNode Names\n");
    i=0;
    while(i<n){
        printf("%d\t%d\t\t",u[i],s[i]);
        j=0;
        while(j<s[i]){
            printf("%d\t\t",d[i][j]);
            j++;
        }
        printf("\n");
        i++;
    }
    for(i=0;i<total;i++){
        for(j=0;j<total;j++){
            if(adj[i][j]==1)
                adj[j][i]=1;
        }
    }
    printf("Adjacency Matrix\n");
    for(i=0;i<total;i++){
        for(j=0;j<total;j++){
            printf("%4d",adj[i][j]);
        }
        printf("\n");
    }
    printf("\nEnter the starting node:");
    scanf("%d",&st);
    hierarchical(adj,total,st);
}

```

```

void hierarchical(int G[MAX][MAX],int n,int startnode)
{

```

```

    int cost[MAX][MAX],distance[MAX],pred[MAX];
    int visited[MAX],count,mindistance,nextnode,i,j;
    for(i=0;i<n;i++)
        for(j=0;j<n;j++)
            if(G[i][j]==0)
                cost[i][j]=INFINITY;
    else

```



```
        cost[i][j]=G[i][j];
        for(i=0;i<n;i++)
    {
        distance[i]=cost[startnode][i];
        pred[i]=startnode;
        visited[i]=0;
    }

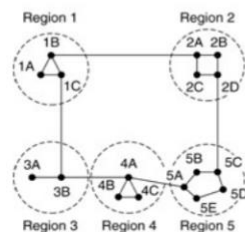
    distance[startnode]=0;
    visited[startnode]=1;
    count=1;

    while(count<n-1)
    {
        mindistance=INFINITY;
        for(i=0;i<n;i++)
            if(distance[i]<mindistance&&!visited[i])
            {
                mindistance=distance[i];
                nextnode=i;
            }
        visited[nextnode]=1;
        for(i=0;i<n;i++)
            if(!visited[i])
                if(mindistance+cost[nextnode][i]<distance[i])
                {
                    distance[i]=mindistance+cost[nextnode][i];
                    pred[i]=nextnode;
                }
        count++;
    }
    for(i=0;i<n;i++)
        if(i!=startnode)
        {
            printf("\nNo.of hops to node%d=%d",i,distance[i]);
            printf("\nPath=%d",i);

            j=i;
            do
            {
                j=pred[j];
                printf("<-%d",j);
            }while(j!=startnode);
        }
    }
```

OUTPUT:

Hierarchical Routing



(a)

Full table for 1A

Dest.	Line	Hops
1A	—	—
1B	1B	1
1C	1C	1
2A	1B	2
2B	1B	3
2C	1B	3
2D	1B	4
3A	1C	3
3B	1C	2
4A	1C	3
4B	1C	4
4C	1C	4
5A	1C	4
5B	1C	5
5C	1B	5
5D	1C	6
5E	1C	5

(b)

Hierarchical table for 1A

Dest.	Line	Hops
1A	—	—
1B	1B	1
1C	1C	1
2	1B	2
3	1C	2
4	1C	3
5	1C	4

(c)

Enter no.of Regions:5

Enter region names:1

2

3

4

5

Enter region sizes:3

4

2

3

5

Enter node names for region1:1

2

3

Enter node names for region2:4

5

6

7

Enter node names for region3:8

9

Enter node names for region4:10

11

12

Enter node names for region5:13

14

15

16

17

Enter edge in source-destination format (0 0 to quit): 1 2

1 3

2 3

2 4

4 5

4 6

5 7

6 7

3 9

8 9

9 10

10 11

10 12

11 12

10 13

13 14

14 15

15 16

13 17

16 17

0 0

Region Node Size Node Names

1 3 1 2 3

2 4 4 5 6 7

3 2 8 9

4 3 10 11 12

5 5 13 14 15 16 17

Adjacency Matrix

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0

0 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0

0 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0

0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0

0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0

0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0

0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0

0 0 0 1 0 0 0 0 1 0 1 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 1 0 1 1 1 0 0 0

```

0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0

```

Enter the starting node:1

No.of hops to node0=9999

Path=0<-1

No.of hops to node2=1

Path=2<-1

No.of hops to node3=1

Path=3<-1

No.of hops to node4=2

Path=4<-2<-1

No.of hops to node5=3

Path=5<-4<-2<-1

No.of hops to node6=3

Path=6<-4<-2<-1

No.of hops to node7=4

Path=7<-5<-4<-2<-1

No.of hops to node8=3

Path=8<-9<-3<-1

No.of hops to node9=2

Path=9<-3<-1

No.of hops to node10=3

Path=10<-9<-3<-1

No.of hops to node11=4

Path=11<-10<-9<-3<-1

No.of hops to node12=4

Path=12<-10<-9<-3<-1

No.of hops to node13=4

Path=13<-10<-9<-3<-1

No.of hops to node14=5

Path=14<-13<-10<-9<-3<-1

No.of hops to node15=6

Path=15<-14<-13<-10<-9<-3<-1

No.of hops to node16=7

Path=16<-15<-14<-13<-10<-9<-3<-1

VIVA - VOCE:**1. What are the benefits of routing?**

The primary function of routing is to share a single network connection among different machines having a router is surely a first step towards securing a network connection. Dynamic Routing, Packet Filtering, Backup Plan, NAT Usage, Integrations and Speed.

2. List different routing algorithms?

Static or Dynamic Routing, Distributed or Centralized, Single path or Multi-path, Flat or Hierarchical, IntraDomain or Inter-Domain and link State or Distance Vector.

3. What is Hierarchical routing algorithm?

Hierarchical Routing is the method of routing in networks that is based on hierarchical addressing. Most transmission control protocol, Internet protocol (DCPIP). Routing is based on two level of hierarchical routing in which IP address is divided into a network, person and a host person

4. What are the advantages of Hierarchical routing?

Hierarchical Routing decreases the complexity of network topology, increases routing efficiency, and causes much less congestion because of fewer routing advertisements. With hierarchical routing, only core routers connected to the backbone are aware of all routes

5. What is the disadvantage of Hierarchical routing?

Once the hierarchy is imposed on the network, it is followed and possibility of direct paths is ignored

7. Implement the token bucket congestion control algorithm.

THEORY: The token bucket is an algorithm used in packet-switched and telecommunications networks. It can be used to check that data transmissions in the form of packets conform to defined limits on bandwidth and burstiness.

ALGORITHM

Step-1: Start

Step-2: A token is added to the bucket every $1/r$ seconds.

Step-3: The bucket can hold at the most b tokens. If a token arrives when the bucket is full, it is discarded.

Step-4: When a packet (network layer PDU) of n bytes arrives,

i. if at least n tokens are in the bucket, n tokens are removed from the bucket, and the packet is sent to the network.

ii. if fewer than n tokens are available, no tokens are removed from the bucket, and the packet is considered to be non-conformant.

Step-5: Stop

```
import java.io.*;
```

```
import java.util.*;
```

```
class Tokenbucket {
```

```
    public static void main (String[] args) {
```

```
        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 count is checked
```

```
        no_of_queries=3;
```

```
        //total no. of packets that can
```

```
        // be accomodated 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++)
```

```
        {
```

```
            size_left=bucket_size-storage; //space left
```

```
            if(input_pkt_size<=(size_left))
```

```
            {
```

```
        storage+=input_pkt_size;
        System.out.println("Buffer size= "+storage+
            " out of bucket size= "+bucket_size);
    }
    else
    {
        System.out.println("Packet loss = "
            +(input_pkt_size-
(size_left)));

        //full size
        storage=bucket_size;

        System.out.println("Buffer size= "+storage+
            " out of bucket size=
"+bucket_size);
    }
    storage-=output_pkt_size;
}
}
```

Output

```
Buffer size= 4 out of bucket size= 10
Buffer size= 7 out of bucket size= 10
Buffer size= 10 out of bucket size= 10
Packet loss = 3
Buffer size= 10 out of bucket size= 10
```

VIVA - VOCE:**1.What is Congestion?**

Network congestion in data networking and queueing theory is the reduced quality of service that occurs when a network node or link is carrying more data than it can handle. Typical effects include queueing delay, packet loss or the blocking of new connections

2.What are the effects of Congestion?

Typical effects include queuing delay, packet loss or the blocking of new connections. A consequence of congestion is that an incremental increase in offered load leads either only to a small increase or even a decrease in network throughput.

3.Explain Leaky bucket algorithm?

The leaky bucket is an algorithm based on an analogy of how a bucket with a constant leak will overflow if either the average rate at which water is poured in exceeds the rate at which the bucket leaks or if more water than the capacity of the bucket is poured in all at once.

4.Explain Token bucket algorithm?

The token bucket is an algorithm used in packet-switched and telecommunications networks. It can be used to check that data transmissions in the form of packets conform to defined limits on bandwidth and burstiness (a measure of the unevenness or variations in the traffic flow).

5.What is maximum burst size?

The maximum burst size is the maximum number of bytes a router can absorb without dropping a packet. This is determined by the size of the router queue, and by the current cross traffic at that router. Not exceeding the MBS is the key to obtaining good achievable throughput.

8) Write C programs to simulate the following CPU Scheduling algorithms:

- a) FCFS b) priority c) SJF d) Priority

THEORY: Scheduling is a fundamental operating system function.

CPU scheduling is the basis of multi programming operating system. CPU scheduling algorithm determines how the CPU will be allocated to the process. These are of two types.

1.Primitive scheduling algorithms

2.Non-Primitive scheduling algorithms

Primitive Scheduling algorithms: In this, the CPU can release the process even in the middle of execution. For example: the cpu executes the process p1, in the middle of execution the cpu received a request signal from process p2, then the OS compares the priorities of p1&p2. If the priority p1 is higher than the p2 then the cpu continue the execution of process p1. Otherwise the cpu preempt the process p1 and assigned to process p2.

Non-Primitive Scheduling algorithm: In this, once the cpu assigned to a process the processor do not release until the completion of that process. The cou will assign to some other job only after the previous job has finished.

Scheduling methodology:

Through put: It means how many jobs are completed by the CPU with in a time period.

Turn around time: The time interval between the submission of the process and the time of the completion is the turn around time.

$$\text{Turn around time} = \text{Finished time} - \text{arrival time}$$

Waiting time: it is the sum of the periods spent waiting by a process in the ready queue

$$\text{Waiting time} = \text{Starting time} - \text{arrival time}$$

Response time: it is the time duration between the submission and first response

$$\text{Response time} = \text{First response} - \text{arrival time}$$

CPU Utilization: This is the percentage of time that the processor is busy. CPU utilization may range from 0 to 100

First-come, first-serve scheduling(FCFS): In this, which process enter the ready queue first is served first. The OS maintains DS that is ready queue. It is the simplest CPU scheduling algorithm. If a process request the CPU then it is loaded into the ready queue, which process is the head of the ready queue, connect the CPU to that process.

Shortest job First: The criteria of this algorithm are which process having the smallest CPU burst, CPU is assigned to that next process. If two process having the same CPU burst time FCFS is used to break the tie.

Priority Scheduling: These are of two types.

One is internal priority, second is external priority. The cpu is allocated to the process with the highest priority. Equal priority processes are scheduled in the FCFS order. Priorities are generally some fixed range of numbers such as 0 to 409. The low numbers represent high priority

Round Robin: It is a primitive scheduling algorithm it is designed especially for time sharing systems. In this, the CPU switches between the processes. When the time quantum expired, the CPU switches to another job. A small unit of time called a quantum or time slice. A time quantum is generally is a circular queue new processes are added to the tail of the ready queue.

If the process may have a CPU burst of less than one time slice then the process release the CPU voluntarily. The scheduler will then process to next process ready queue otherwise; the process will be put at the tail of the ready queue.

A)FCFS ALGORITHM:

Step 1: Start the process

Step 2: Accept the number of processes in the ready Queue

Step 3: For each process in the ready Q, assign the process id and accept the CPU burst time

Step 4: Set the waiting of the first process as '0' and its burst time as its turnaround time

Step 5: for each process in the Ready Q calculate

(c) Waiting time for process(n)= waiting time of process(n-1)+Burst time of process(n-1)

(d) Turn around time for Process(n)= waiting time of Process(n)+ Burst time for process(n)

Step 6: Calculate

(e) Average waiting time = Total waiting Time / Number of process

(f) Average Turnaround time = Total Turnaround Time / Number of process

Step 7: Stop the process

SOURCE CODE:

```
#include<stdio.h>
#include<conio.h>
void main()
{
int pid[10],bt[10],wt[10],tat[10],n,twt=0,ttat=0,i;
```

```
float awt,atat;

clrscr();

printf("Enter no.of processes:");

scanf("%d",&n);

printf("\n Enter burst times:");

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

scanf("%d",&bt[i]);

wt[0]=0;

tat[0]=bt[0];

for(i=1;i<n;i++)

{

wt[i]=tat[i-1];

tat[i]=bt[i]+wt[i];

}

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

{

ttat= ttat+tat[i];

twt=twt+wt[i];

}

printf("\n PID \t BT \t WT \t TAT");

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

printf("\n %d\t%d\t%d\t%d",i+1,bt[i],wt[i],tat[i]);

awt=(float)twt/n;

atat=(float)ttat/n;

printf("\nAvg. Waiting Time=%f",awt);

printf("\nAvg. Turn around time=%f",atat);

getch();

}
```

OUTPUT:

```

Enter no.of processes:3
Enter burst times:2
5
4
PID    BT    WT    TAT
1       2     0     2
2       5     2     7
3       4     7    11
Avg. Waiting Time=3.000000
Avg. Turn around time=6.666667
Process returned 31 (0x1F)  execution time : 28.359 s
Press any key to continue.

```

B) SJF ALGORITHM:

Step 1: Start the process

Step 2: Accept the number of processes in the ready Queue

Step 3: For each process in the ready Q, assign the process id and accept the CPU burst time

Step 4: Start the Ready Q according the shortest Burst time by sorting according to lowest to highest burst time.

Step 5: Set the waiting time of the first process as '0' and its turnaround time as its burst time.

Step 6: For each process in the ready queue, calculate

(a) Waiting time for process(n)= waiting time of process (n-1)+Burst time of process(n-1)

(b) Turn around time for Process(n)= waiting time of Process(n)+ Burst time for process(n)

Step 6: Calculate

(c) Average waiting time = Total waiting Time / Number of process

(d) Average Turnaround time = Total Turnaround Time / Number of process

Step 7: Stop the process

SOURCE CODE:

```

#include<stdio.h>
#include<conio.h>

void main(){
int pid[10],bt[10],wt[10],tat[10],n,twt=0,ttat=0,i,j,t;

```

```
float awt,atat;

clrscr();

printf("Enter no.of processes:");

scanf("%d",&n);

printf("\n Enter burst times:");

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

scanf("%d",&bt[i]);

printf("\n Enter PID:");

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

scanf("%d",&pid[i]);

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

{

for(j=i+1;j<n;j++)

{

if(bt[i]>bt[j])

{

t=bt[i];

bt[i]=bt[j];

bt[j]=t;

}

}

}

t=pid[i];

pid[i]=pid[j];

pid[j]=t;

}

}

}

wt[0]=0;

tat[0]=bt[0];

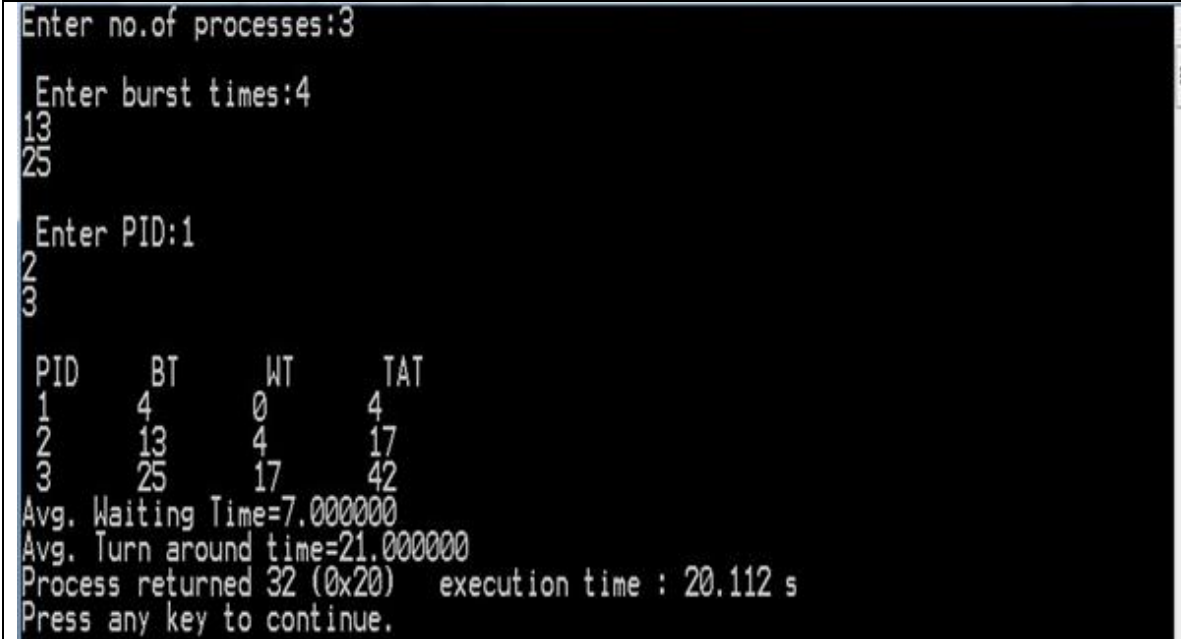
for(i=1;i<n;i++)

{

wt[i]=tat[i-1];

tat[i]=bt[i]+wt[i];
```

```
}  
for(i=0;i<n;i++)  
{  
    ttat= ttat+tat[i];  
    twt=twt+wt[i];  
}  
printf("\n PID \t BT \t WT \t TAT");  
for(i=0;i<n;i++)  
    printf("\n %d\t%d\t%d\t%d",pid[i],bt[i],wt[i],tat[i]);  
awt=(float)twt/n;  
atat=(float)ttat/n;  
printf("\nAvg. Waiting Time=%f",awt);  
printf("\nAvg. Turn around time=%f",atat);  
getch();  
}
```

OUTPUT:

```
Enter no.of processes:3  
Enter burst times:4  
13  
25  
Enter PID:1  
2  
3  


| PID | BT | WT | TAT |
|-----|----|----|-----|
| 1   | 4  | 0  | 4   |
| 2   | 13 | 4  | 17  |
| 3   | 25 | 17 | 42  |

  
Avg. Waiting Time=7.000000  
Avg. Turn around time=21.000000  
Process returned 32 (0x20)   execution time : 20.112 s  
Press any key to continue.
```

C) ROUND ROBIN ALGORITHM:

Step 1: Start the process

Step 2: Accept the number of processes in the ready Queue & time quantum or time slice

Step 3: For each process in the ready Q, assign the process id & accept the CPU burst time

Step 4: Calculate the no. of time slices for each process where No. of time slice for process(n) = burst time process(n)/time slice

Step 5: If the burst time is less than the time slice then the no. of time slices =1.

Step 6: Consider the ready queue is a circular Q, calculate

(a) Waiting time for process(n) = waiting time of process(n-1) + burst time of process(n-1) + the time difference in getting the CPU from process(n-1)

(b) Turn around time for process(n) = waiting time of process(n) + burst time of process(n) + the time difference in getting CPU from process(n).

Step 7: Calculate

(a) Average waiting time = Total waiting Time / Number of process

(b) Average Turnaround time = Total Turnaround Time / Number of process

Step 8: Stop the process

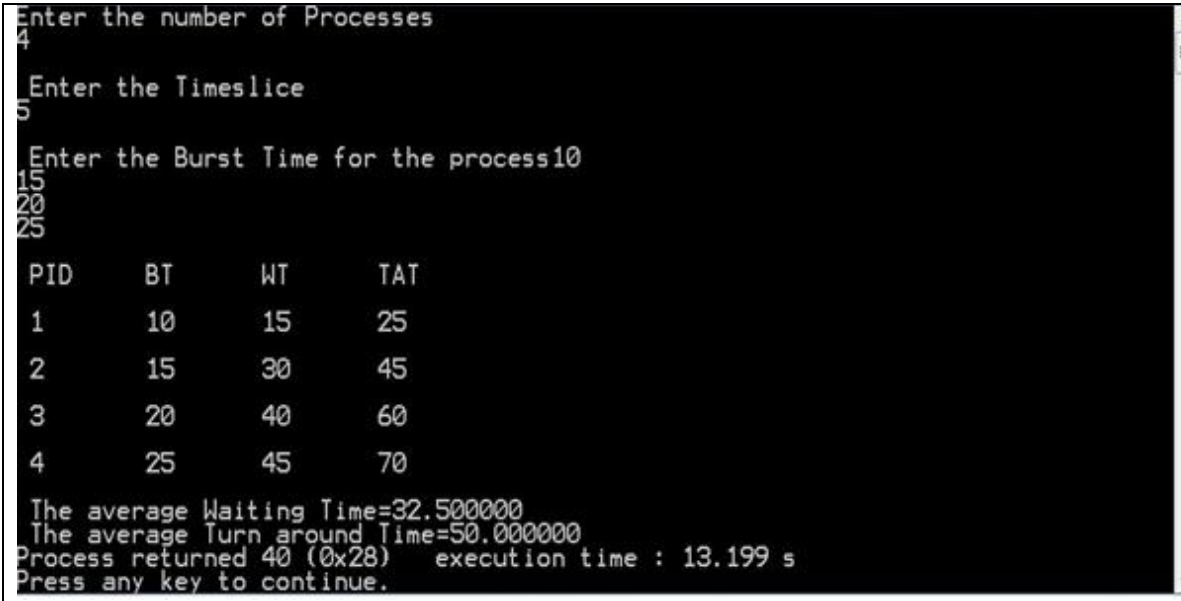
SOURCE CODE:

```
#include<stdio.h>
#include<conio.h>
void main()
{
int ts,bt1[10],wt[10],tat[10],i,j=0,n,bt[10],ttat=0,twt=0,tot=0;
float awt,atat;
clrscr();
printf("Enter the number of Processes \n");
scanf("%d",&n);
printf("\n Enter the Timeslice \n");
scanf("%d",&ts);
printf("\n Enter the Burst Time for the process");
for(i=1;i<=n;i++)
{
```

```
scanf("%d",&bt1[i]);
bt[i]=bt1[i];
}
while(j<n)
{
for(i=1;i<=n;i++)
{
if(bt[i]>0)
{
if(bt[i]>=ts)
{
tot+=ts;
bt[i]=bt[i]-ts;
if(bt[i]==0)
{
j++;
tat[i]=tot;
}
}
}
else
{
tot+=bt[i];
bt[i]=0;
j++;
tat[i]=tot;
}
}
}
}
for(i=1;i<=n;i++)
{
wt[i]=tat[i]-bt1[i];
```



```
twt=twt+wt[i];
ttat=ttat+tat[i];
}
awt=(float)twt/n;
atat=(float)ttat/n;
printf("\n PID \t BT \t WT \t TAT\n");
for(i=1;i<=n;i++) {
printf("\n %d \t %d \t %d \t %d \t\n",i,bt1[i],wt[i],tat[i]);
}
printf("\n The average Waiting Time=%f",awt);
printf("\n The average Turn around Time=%f",atat);
getch();
}
```

OUTPUT:

The screenshot shows the execution of a C program for process scheduling. It prompts the user to enter the number of processes (4), the timeslice (5), and the burst time for each process (10, 15, 20, 25). The output displays a table with columns PID, BT, WT, and TAT, showing the calculated values for each process. The average waiting time is 32.500000 and the average turn around time is 50.000000. The program also shows the execution time (13.199 s) and the return code (40).

```
Enter the number of Processes
4
Enter the Timeslice
5
Enter the Burst Time for the process10
15
20
25
PID    BT    WT    TAT
1      10    15    25
2      15    30    45
3      20    40    60
4      25    45    70
The average Waiting Time=32.500000
The average Turn around Time=50.000000
Process returned 40 (0x28)    execution time : 13.199 s
Press any key to continue.
```

D)PRIORITY ALGORITHM:

Step 1: Start the process

Step 2: Accept the number of processes in the ready Queue

Step 3: For each process in the ready Q, assign the process id, process priority and accept the CPU burst time

Step 4: Start the Ready Q according the highest priority by sorting according to highest to lowest priority.

Step 5: Set the waiting time of the first process as '0' and its turnaround time as its burst time.

Step 6: For each process in the ready queue, calculate

(a) Waiting time for process(n)= waiting time of process (n-1)+Burst time of process(n-1)

(b) Turn around time for Process(n)= waiting time of Process(n)+ Burst time for process(n)

Step 6: Calculate

(c) Average waiting time = Total waiting Time / Number of process

(d) Average Turnaround time = Total Turnaround Time / Number of process

Step 7: Stop the process

SOURCE CODE:

```
#include<stdio.h>

#include<conio.h>

void main()

{

int pid[10],bt[10],pr[10],wt[10],tat[10],n,twt=0,ttat=0,i,j,t;

float awt,atat;

clrscr();

printf("Enter no.of processes:");

scanf("%d",&n);

printf("\n Enter burst times:");

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

scanf("%d",&bt[i]);
```

```
printf("\n Enter PID:");  
  
for(i=0;i<n;i++)  
  
scanf("%d",&pid[i]);  
  
printf("\n Enter Priorities:");  
  
for(i=0;i<n;i++)  
  
scanf("%d",&pr[i]);  
  
for(i=0;i<n;i++)  
  
{  
  
for(j=i+1;j<n;j++)  
  
{  
  
if(pr[i]>pr[j])  
  
{  
  
t=pr[i];  
  
pr[i]=pr[j];  
  
pr[j]=t;  
  
  
  
t=bt[i];  
  
bt[i]=bt[j];  
  
bt[j]=t;  
  
  
  
t=pid[i];  
  
pid[i]=pid[j];  
  
pid[j]=t;  
  
}  
  
}  
  
}  
  
wt[0]=0;
```

```
tat[0]=bt[0];
for(i=1;i<n;i++)
{
wt[i]=tat[i-1];
tat[i]=bt[i]+wt[i];
}
for(i=0;i<n;i++)
{
ttat= ttat+tat[i];
twt=twt+wt[i];
}

printf("\n PID PRIORITY \t BT \t WT \t TAT");

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

printf("\n %d\t%d\t%d\t%d\t%d",pid[i],pr[i],bt[i],wt[i],tat[i]);

awt=(float)twt/n;

atat=(float)ttat/n;

printf("\nAvg. Waiting Time=%f",awt);

printf("\nAvg. Turn around time=%f",atat);

getch();
}
```

OUTPUT:

```

E:\krishna\priority.exe
Enter no.of processes:4
Enter burst times:2
6
4
5
Enter PID:3
6
4
5
Enter Priorities:3
2
6
1
PID PRIORITY    BT    WT    TAT
5      1      5      0      5
6      2      6      5     11
3      3      2     11     13
4      6      4     13     17
Avg. Waiting Time=7.250000
Avg. Turn around time=11.500000
Process returned 32 (0x20)  execution time : 86.108 s

```

VIVA-VOCE:**1.What is an Operating system?**

An operating system (OS) is system software that manages computer hardware and software resources and provides common services for computer programs.

2.What is a process ?

A *process* is an instance of program in execution. The operating system is responsible for managing all the processes that are running on a computer and allocated each process a certain amount of time to use the processor.

3. What Are The Advantages Of A Multiprocessor System?

With an increased number of processors, there is considerable increase in throughput. It can also save more money because they can share resources. Finally, overall reliability is increased as well.

4.Explain starvation and Aging?

Starvation or indefinite blocking is phenomenon associated with the Priority scheduling algorithms, in which a process ready to run for CPU can wait indefinitely because of low priority.

Aging is a technique of gradually increasing the priority of processes that wait in the system for a long time.

5.What are the functions of operating system?

The operating system controls and coordinates the use of hardware among the different processes and applications. It provides the various functionalities to the users. The following are the main job of operating system.

- Resource utilization
- Resource allocation
- Process management
- Memory management
- File management
- I/O management
- Device management

6.What do you know about interrupt?

Interrupt can be understood as a signal from a device causing context switch.

- To handle the interrupts, interrupt handlers or service routines are required.
- The address of each Interrupt service routine is provided in a list which is maintained in interrupt vector.

9. Write C programs to simulate the following CPU Scheduling algorithms:

- a) SJF

b) Priority

- a) Step 1: Start the process
- b) Step 2: Accept the number of processes in the ready Queue
- c) Step 3: For each process in the ready Q, assign the process id and accept the CPU burst time
- d) Step 4: Start the Ready Q according the shortest Burst time by sorting according to lowest to highest burst time.
- e) Step 5: Set the waiting time of the first process as '0' and its turnaround time as its burst time.
- f) Step 6: For each process in the ready queue, calculate
 - g) (a) Waiting time for process(n)= waiting time of process (n-1)+Burst time of process(n-1)
 - h) (b) Turn around time for Process(n)= waiting time of Process(n)+ Burst time for process(n)
- i) Step 6: Calculate
 - j) (c) Average waiting time = Total waiting Time / Number of process
 - k) (d) Average Turnaround time = Total Turnaround Time / Number of process
- l) Step 7: Stop the process

SOURCE CODE:

```
#include<stdio.h>

#include<conio.h>

void main(){

int pid[10],bt[10],wt[10],tat[10],n,tw=0,tt=0,i,j,t;

float awt,atat;

clrscr();

printf("Enter no.of processes:");

scanf("%d",&n);

printf("\n Enter burst times:");

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

scanf("%d",&bt[i]);

printf("\n Enter PID:");

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

scanf("%d",&pid[i]);

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

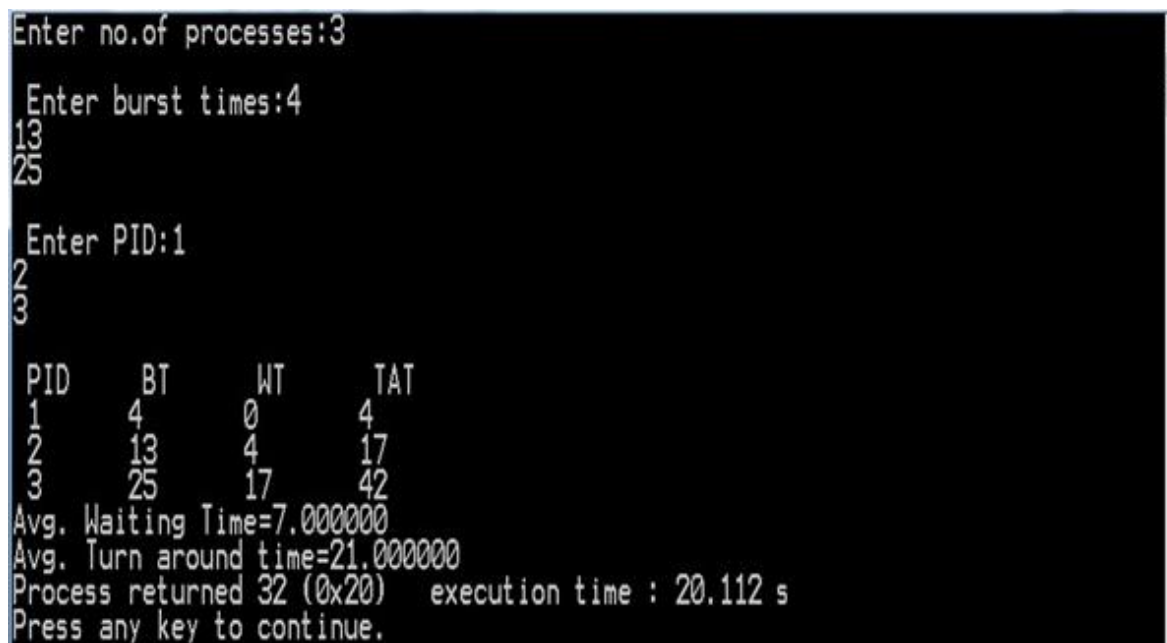
{
```

```
for(j=i+1;j<n;j++)
{
if(bt[i]>bt[j])
{
t=bt[i];
bt[i]=bt[j];
bt[j]=t;

t=pid[i];
pid[i]=pid[j];
pid[j]=t;
}
}
wt[0]=0;
tat[0]=bt[0];
for(i=1;i<n;i++)
{
wt[i]=tat[i-1];
tat[i]=bt[i]+wt[i];
}
for(i=0;i<n;i++)
{
ttat= ttat+tat[i];
tw=tw+wt[i];
}
printf("\n PID \t BT \t WT \t TAT");
for(i=0;i<n;i++)
printf("\n %d\t%d\t%d\t%d",pid[i],bt[i],wt[i],tat[i]);
aw=(float)tw/n;
at=(float)ttat/n;
printf("\nAvg. Waiting Time=%f",aw);
```



```
printf("\nAvg. Turn around time=%f",atat);  
getch();  
}
```



The screenshot shows the execution of a C program that simulates priority scheduling. It prompts the user to enter the number of processes (3), burst times (4, 13, 25), and process IDs (1, 2, 3). It then displays a table of results for each process, including PID, BT (Burst Time), WT (Waiting Time), and TAT (Turn Around Time). The average waiting time is 7.000000 and the average turn around time is 21.000000. The program also shows the process returned 32 (0x20) and the execution time is 20.112 s.

```
Enter no.of processes:3  
Enter burst times:4  
13  
25  
Enter PID:1  
2  
3  


| PID | BT | WT | TAT |
|-----|----|----|-----|
| 1   | 4  | 0  | 4   |
| 2   | 13 | 4  | 17  |
| 3   | 25 | 17 | 42  |

  
Avg. Waiting Time=7.000000  
Avg. Turn around time=21.000000  
Process returned 32 (0x20) execution time : 20.112 s  
Press any key to continue.
```

b) Priority

Step 1: Start the process

Step 2: Accept the number of processes in the ready Queue

Step 3: For each process in the ready Q, assign the process id, process priority and accept the CPU burst time

Step 4: Start the Ready Q according the highest priority by sorting according to highest to lowest priority.

Step 5: Set the waiting time of the first process as '0' and its turnaround time as its burst time.

Step 6: For each process in the ready queue, calculate

(a) Waiting time for process(n)= waiting time of process (n-1)+Burst time of process(n-1)

(b) Turn around time for Process(n)= waiting time of Process(n)+ Burst time for process(n)

Step 6: Calculate

(c) Average waiting time = Total waiting Time / Number of process

(d) Average Turnaround time = Total Turnaround Time / Number of process

Step 7: Stop the process

SOURCE CODE:

```
#include<stdio.h>

#include<conio.h>

void main()

{

int pid[10],bt[10],pr[10],wt[10],tat[10],n,twt=0,ttat=0,i,j,t;

float awt,atat;

clrscr();

printf("Enter no.of processes:");

scanf("%d",&n);

printf("\n Enter burst times:");

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

scanf("%d",&bt[i]);

printf("\n Enter PID:");

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

scanf("%d",&pid[i]);

printf("\n Enter Priorities:");

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

scanf("%d",&pr[i]);

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

{

for(j=i+1;j<n;j++)

{
```

```
if(pr[i]>pr[j])
{
    t=pr[i];
    pr[i]=pr[j];
    pr[j]=t;

    t=bt[i];
    bt[i]=bt[j];
    bt[j]=t;

    t=pid[i];
    pid[i]=pid[j];
    pid[j]=t;
}
}
}

wt[0]=0;
tat[0]=bt[0];
for(i=1;i<n;i++)
{
    wt[i]=tat[i-1];
    tat[i]=bt[i]+wt[i];
}
for(i=0;i<n;i++)
{
    ttat= ttat+tat[i];
    twt=twt+wt[i];
```

```

}

printf("\n PID PRIORITY \t BT \t WT \t TAT");

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

printf("\n %d\t%d\t%d\t%d\t%d",pid[i],pr[i],bt[i],wt[i],tat[i]);

awt=(float)twtn;

atat=(float)ttatn;

printf("\nAvg. Waiting Time=%f",awt);

printf("\nAvg. Turn around time=%f",atat);

getch();

}

```

```

E:\krishna\priority.exe
Enter no.of processes:4
Enter burst times:2
6
4
5
Enter PID:3
6
4
5
Enter Priorities:3
2
6
1
PID PRIORITY    BT    WT    TAT
5         1        5     0     5
6         2        6     5    11
3         3        2    11    13
4         6        4    13    17
Avg. Waiting Time=7.250000
Avg. Turn around time=11.500000
Process returned 32 (0x20)    execution time : 86.108 s

```

10. Write C programs to simulate the following file allocation strategies:

- a) Sequential b) Linked c) Indexed

a) Sequential

Theory: The most common form of file structure is the sequential file in this type of file, a fixed format is used for records. All records (of the system) have the same length, consisting of the same number of fixed length fields in a particular order because the length and position of each field are known, only the values of fields need to be stored, the field name and length for each field are attributes of the file structure.

ALGORITHM:

Step 1: Start the program.

Step 2: Get the number of files.

Step 3: Get the memory requirement of each file.

Step 4: Allocate the required locations to each in sequential order

a). Randomly select a location from available location $s1 = \text{random}(100)$;

Step 5: Print the results file no, length, Blocks allocated.

Step 6: Stop the program

SOURCE CODE:

```
#include<stdio.h>
#include<conio.h>
main()
{
int n,i,j,b[20],sb[20],t[20],x,c[20][20];
//clrscr();
printf("\t\t Sequential File allocation algorithm");
printf("\n\nEnter no.of files:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("Enter no. of blocks occupied by file%d",i+1);
scanf("%d",&b[i]);
printf("Enter the starting block of file%d",i+1);
scanf("%d",&sb[i]);
t[i]=sb[i];
for(j=0;j<b[i];j++)
c[i][j]=sb[i]++;
}
printf("Filename\tStart block\tlength\n");
for(i=0;i<n;i++)
printf("%d\t\t %d \t\t%d\n",i+1,t[i],b[i]);
printf("Enter file name:");
```

```
scanf("%d",&x);
printf("\nFile name is:%d",x);
printf("\nlength is:%d",b[x-1]);
printf("\nblocks occupied:");
for(i=0;i<b[x-1];i++)
printf("%4d",c[x-1][i]);
getch();

}
```

OUTPUT:

```
Enter no. of files:3
Enter file name:A
Enter starting block:10
Enter no.of blocks:3
Enter block numbers:10
20
Enter file name:B
Enter starting block:20
Enter no.of blocks:3
Enter block numbers:30
32
Enter file name:C
Enter starting block:20
Enter no.of blocks:3
Enter block numbers:50
32
File  start  size  block
A    10    3    10--->10--->20
B    20    3    20--->30--->32
C    20    3    20--->50--->32
```

b) Linked

In the chained method file allocation table contains a field which points to starting block of memory. From it for each bloc a pointer is kept to next successive block. Hence, there is no external fragmentation.

ALGORITHM:

Step 1: Start the program.

Step 2: Get the number of files.

Step 3: Get the memory requirement of each file.

Step 4: Allocate the required locations by selecting a location randomly $q = \text{random}(100)$;

a) Check whether the selected location is free .

b) If the location is free allocate and set $\text{flag}=1$ to the allocated locations.

$q = \text{random}(100)$;

```
{
  if(b[q].flag==0)
    b[q].flag=1;
    b[q].fno=j;
    r[i][j]=q;
```

Step 5: Print the results file no, length ,Blocks allocated.

Step 6: Stop the program

SOURCE CODE:

```
#include<stdio.h>
#include<conio.h>
struct file
{
  char fname[10];
  int start,size,block[10];
}f[10];
main()
{
  int i,j,n;
  //clrscr();
  printf("\t\t Linked File allocation algorithm");
  printf("\n\nEnter no. of files:");
  scanf("%d",&n);
  for(i=0;i<n;i++)
  {
    printf("Enter file name:");
    scanf("%s",&f[i].fname);
    printf("Enter starting block:");
    scanf("%d",&f[i].start);
    f[i].block[0]=f[i].start;
    printf("Enter no.of blocks:");
    scanf("%d",&f[i].size);
    printf("Enter block numbers:");
```

```

for(j=1;j<f[i].size;j++)
{
scanf("%d",&f[i].block[j]);
}
}
printf("File\tstart\tsize\tblock\n");
for(i=0;i<n;i++)
{
printf("%s\t%d\t%d\t",f[i].fname,f[i].start,f[i].size);
for(j=0;j<f[i].size-1;j++)
printf("%d--->",f[i].block[j]);
printf("%d\n",f[i].block[j]);
}
getch();
}

```

OUTPUT:

```

Enter no. of files:3
Enter file name:1
Enter starting block:50
Enter no.of blocks:5
Enter block numbers:10
20
60
70
Enter file name:1
Enter starting block:21
Enter no.of blocks:3
Enter block numbers:101
201
Enter file name:3
Enter starting block:31
Enter no.of blocks:3
Enter block numbers:234
456
File  start  size  block
1    50    5    50--->10--->20--->60--->70
1    21    3    21--->101--->201
3    31    3    31--->234---> 456

```

c) Indexed

In the chained method file allocation table contains a field which points to starting block of memory. From it for each bloc a pointer is kept to next successive block. Hence, there is no external fragmentation

ALGORTHIM:

Step 1: Start the program.

Step 2: Get the number of files.

Step 3: Get the memory requirement of each file.

Step 4: Allocate the required locations by selecting a location randomly

q= random(100);

a) Check whether the selected location is free .

b) If the location is free allocate and set flag=1 to the allocated locations.

While allocating next location address to attach it to previous location

for(i=0;i0)

{ }

}

p=r[i][j-1];

b[p].next=q;

}

Step 5: Print the results file no, length ,Blocks allocated.

Step 6: Stop the program

SOURCE CODE:

```
#include<stdio.h>
#include<conio.h>
main()
{
int n,m[20],i,j,sb[20],b[20][20],x;
//clrscr();
printf("\t\t Indexed File allocation algorithm");
printf("\n\nEnter no. of files:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("\n\nEnter index block of file%d:",i+1);
scanf("%d",&sb[i]);
printf("\n\nEnter length of file%d:",i+1);
scanf("%d",&m[i]);
printf("\n\nenter blocks of file%d:",i+1);
for(j=0;j<m[i];j++)
scanf("%d",&b[i][j]);
}
printf("\n\nFile\t Index\tLength\n");
for(i=0;i<n;i++)
{
printf("%d\t%d\t%d\n",i+1,sb[i],m[i]);
}
printf("\n\nEnter file name:");
scanf("%d",&x);
printf("\n\nfile name is:%d",x);
printf("\n\nIndex is:%d",sb[x-1]);
```

```
printf("\nBlocks occupied are:");
for(j=0;j<m[x-1];j++)
printf("%4d",b[x-1][j]);
getch();
}
```

OUTPUT:

```
Enter no. of files:3
Enter file name:A
Enter starting block:10
Enter no.of blocks:5
Enter block numbers:10
44
12
34
Enter file name:B
Enter starting block:50
Enter no.of blocks:3
Enter block numbers:21
23
Enter file name:10
Enter starting block:20
Enter no.of blocks:5
Enter block numbers:56
65
43
32
File  start  size  block
A    10    5    10--->10--->44--->12--->34
B    50    3    50--->21--->23
10   20    5    20--->56--->65--->43--->32
```

VIVA-VOCE:**1.Explain file allocation in Operating system?**

A file allocation table (FAT) is a table that an operating system maintains on a hard disk that provides a map of the clusters (the basic units of logical storage on a hard disk) that a file has been stored in.

2.What is contiguous and noncontiguous memory allocations?

The basic difference between contiguous and noncontiguous memory allocation is that contiguous allocation allocates one single contiguous block of memory to the process whereas,

the noncontiguous allocation divides the process into several blocks and place them in the different address space of the memory.

3. Explain Sequential allocation?

In this scheme, each file occupies a contiguous set of blocks on the disk. For example, if a file requires n blocks and is given a block b as the starting location, then the blocks assigned to the file will be: $b, b+1, b+2, \dots, b+n-1$. This means that given the starting block address and the length of the file (in terms of blocks required). The directory entry for a file with contiguous allocation contains

- Address of starting block
- Length of the allocated portion.

4.What is Linked allocation?

In linked allocation, each file is a linked list of disk blocks. The directory contains a pointer to the first and optionally the last block of the file. For example, a file of 5 blocks which starts at block 4, might continue at block 7, then block 16, block 10, and finally block 27.

5.Explain Indexed allocation?

Provides solutions to problems of contiguous and linked allocation. An index block is created having all pointers to files. Each file has its own index block which stores the addresses of disk space occupied by the file. Directory contains the addresses of index blocks of files.

6.Explain FAT advantages and disadvantages?

Advantages of FAT: The main advantage of FAT is its efficient use of disk space. FAT can place the parts of the file wherever they fit. File names can be up to 255 characters and file extensions longer than 3 characters. Easy to recover file names that have been deleted.

Disadvantages of FAT: Overall performance slows down as more files are stored on the drive. Drives can become fragmented quite easily. FAT lacks many of the security features in NTFS such as being able to assign access rights to files and directories. It can also have file integrity problems such as lost clusters, invalid files and directories and allocation errors.

11. Write C programs to simulate the following memory management techniques:
a) Paging b) Segmentation

Theory: Paging is an efficient memory management scheme because it is non-contiguous memory allocation method. The basic idea of paging is the physical memory (main memory) is divided into fixed sized blocks called frames, the logical address space is divided into fixed sized blocks, called pages, but page size and frame size should be equal. The size of the frame or a page is depending on operating system.

In this scheme the operating system maintains a data structure that is page table, it is used for mapping purpose. The page table specifies the some useful information, it tells which frames are there and so on. The page table consisting of two fields, one is the page number and other one is frame number. Every address generated by the CPU divided into two parts, one is page number and second is page offset or displacement. The pages are loaded into available free frames in the physical memory.

Step 1: Read all the necessary input from the keyboard.

Step 2: Pages - Logical memory is broken into fixed - sized blocks.

Step 3: Frames – Physical memory is broken into fixed – sized blocks.

Step 4: Calculate the physical address using the following

Physical address = (Frame number * Frame size) + offset

Step 5: Display the physical address.

Step 6: Stop the process

SOURCE CODE:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int i,j,temp,framearr[20],pages,pageno,frames,memsize,log,pagesize,prosize,base;
    clrscr();
    printf("Enter the Process size: ");
    scanf("%d",&prosize);
    printf("\nEnter the main memory size: ");
    scanf("%d",&memsize);
    printf("\nEnter the page size: ");
    scanf("%d",&pagesize);
```

```

pages=prosize/pagesize;
printf("\nThe process is divided into %d pages",pages);
frames = memsize/pagesize;
printf("\n\nThe main memory is divided into %d frames\n",frames);
for(i=0;i<frames;i++)
    framearr[i]=-1;    /* Initializing array elements with -1 */
for(i=0;i<pages;i++)
{
pos:   printf("\nEnter the frame number of page %d: ",i);
       scanf("%d",&temp); /* storing frameno in temporary variable */
       if(temp>=frames) /*checking wether frameno is valid or not*/
       {
           printf("\n\t****Invalid frame number****\n");
           goto pos;
       }
       /* storing pageno (i.e 'i' ) in framearr at framno (i.e temp ) index */
       for(j=0;j<frames;j++)
           if(temp==j)
               framearr[temp]=i;
}
printf("\n\nFrameno\tpageno\tValidationBit\n-----\t-----\t-----");
for(i=0;i<frames;i++)
{
    printf("\n  %d \t %2d \t",i,framearr[i]);
    if(framearr[i]==-1)
        printf("  0");
    else
        printf("  1");
}
printf("\nEnter the logical address: ");
scanf("%d",&log);
printf("\nEnter the base address: ");

```

```
scanf("%d",&base);
pageno = log/pagesize;
for(i=0;i<frames;i++)
    if(framearr[i]==pageno)
    {
        temp = i;
        break;
    }
j = log%pagesize;    /* here 'j' is displacement */
temp = base + (temp*pagesize)+j;    /* here lhs 'temp' is physical address and rhs 'temp' is
frame number */
printf("\nPhysical address is : %d",temp);
getch();
}
```

OUTPUT:

```
E:\krishna\paging.exe
Enter the Process size: 8
Enter the main memory size: 16
Enter the page size: 2
The process is divided into 4 pages
The main memory is divided into 8 frames
Enter the frame number of page 0: 5
Enter the frame number of page 1: 6
Enter the frame number of page 2: 2
Enter the frame number of page 3: 3

Frameno  pageno  ValidationBit
-----  -
0         -1       0
1         -1       0
2         2       1
3         3       1
4         -1       0
5         0       1
6         1       1
7         -1       0
Enter the logical address: 3
Enter the base address: 1000
Physical address is : 1013
Process returned 27 (0x1B)   execution time : 76.522 s
Press any key to continue.
```

b) Segmentation

Theory: A process is divided into Segments. The chunks that a program is divided into which are not necessarily all of the same sizes are called segments. Segmentation gives user's view of the process which paging does not give. Here the user's view is mapped to physical memory.

Algorithm:

1. Input the number of memory blocks and their sizes and initializes all the blocks as free.
2. Input the number of processes and their sizes.
3. Start by picking each process and check if it can be assigned to the current block, if yes, allocate it the required memory and check for next process but from the block where we left not from starting.
4. If the current block size is smaller then keep checking the further blocks.

SOURCE CODE:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int i,j,m,size,val[10][10],badd[20],limit[20],ladd;
    //clrscr();
    printf("\n \t\tSegmentation");
    printf("\n Enter the size of the segment table:");
    scanf("%d",&size);
    for(i=0;i<size;i++)
    {
        printf("\nEnter infor about segment %d",i+1);
        printf("\nEnter base address:");
        scanf("%d",&badd[i]);
        printf("\nEnter the limit:");
        scanf("%d",&limit[i]);
        for(j=0;j<limit[i];j++)
        {
            printf("\nEnter %d address values:",badd[i]+j);
            scanf("%d",&val[i][j]);
        }
    }
    printf("\n\n*****SEGMENT TABLE*****");
    printf("\nseg.no\tbase\tlimit\n");
    for(i=0;i<size;i++)
    {
        printf("%d\t%d\t%d\n",i+1,badd[i],limit[i]);
    }
    printf("\nEnter segment no.:");
```



```
scanf("%d",&i);
if(i>=size)
{
printf("invalid");
}
else
{
printf("\nEnter the logical address:");
scanf("%d",&ladd);
if(ladd>=limit[i])
printf("invalid");
else
{
m=badd[i]+ladd;
printf("\nmapped physical address is=%d",m);
printf("\nthe value is %d ",val[i][ladd]);
}
}
getch();
}
```

OUTPUT:

```
Enter segment number:1
Enter base value:2000
Enter value for limit:100
Enter segment number:2
Enter base value:2500
Enter value for limit:100
Enter segmentation number:-1
Enter offset:90
Enter segment number:2
Address in physical memory 2590
```

VIVA-VOCE:**1. What is the basic function of paging?**

Paging is a memory management scheme that permits the physical address space of a process to be noncontiguous. It avoids the considerable problem of having to fit varied sized memory chunks onto the backing store.

2.What is fragmentation?

Fragmentation is memory wasted. It can be internal if we are dealing with systems that have fixed-sized allocation units, or external if we are dealing with systems that have variable-sized allocation units.

3.What is Thrashing?

Thrashing is a situation when the performance of a computer degrades or collapses. Thrashing occurs when a system spends more time processing page faults than executing transactions.

4.Differentiate between logical and physical address.

- Physical addresses are actual addresses used for fetching and storing data in main memory when the process is under execution.
- Logical addresses are generated by user programs. During process loading, they are converted by the loader into physical address.

5. Explain compaction.

During the process of loading and removal of process into and out of the memory, the free memory gets broken into smaller pieces. These pieces lie scattered in the memory. Compaction means movement of these pieces close to each other to form a larger chunk of memory which works as a resource to run larger processes.

6.Explain Internal fragmentation and external fragmentation?

Internal fragmentation is the wasted space within each allocated block because of rounding up from the actual requested allocation to the allocation granularity.

External fragmentation is the various free spaced holes that are generated in either memory or disk space.

12. Write a C program to simulate Bankers Algorithm for Deadlock Avoidance

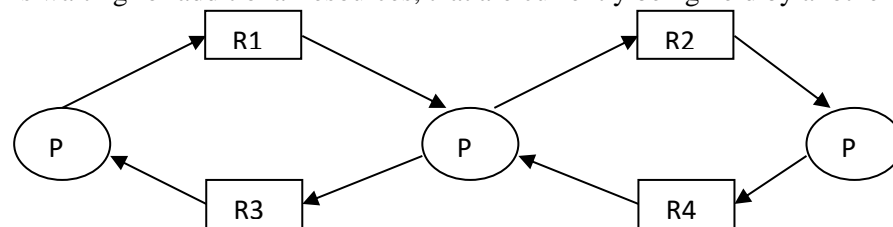
THEORY: Deadlock:

A process request the resources, the resources are not available at that time, so the process enter into the waiting state. The requesting resources are held by another waiting process, both are in waiting state, this situation is said to be Deadlock.

A deadlocked system must satisfied the following 4 conditions. These are:

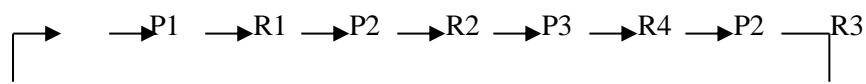
(i) Mutual Exclusion: Mutual Exclusion means resources are in non-sharable mode only, it means only one process at a time can use a process.

(ii) Hold and Wait: Each and every process is the deadlock state, must holding at least one resource and is waiting for additional resources, that are currently being held by another process.



(iii) No Preemption: No Preemption means resources are not released in the middle of the work, they released only after the process has completed its task.

(iv) Circular Wait: If process P1 is waiting for a resource R1, it is held by P2, process P2 is waiting for R2, R2 held by P3, P3 is waiting for R4, R4 is held by P2, P2 waiting for resource R3, it is held by P1.



Deadlock Avoidance: It is one of the method of dynamically escaping from the deadlocks. In this scheme, if a process request for resources, the avoidance algorithm checks before the allocation of resources about the state of system. If the state is safe, the system allocate the resources to the requesting process otherwise (unsafe) do not allocate the resources. So taking care before the allocation said to be deadlock avoidance.

Banker's Algorithm: It is the deadlock avoidance algorithm, the name was chosen because the bank never allocates more than the available cash.

Available: A vector of length 'm' indicates the number of available resources of each type. If $available[j]=k$, there are 'k' instances of resource types R_j available.

Allocation: An $n \times m$ matrix defines the number of resources of each type currently allocated to each process. If $allocation[i,j]=k$, then process P_i is currently allocated 'k' instances of resources type R_j .

Max: An $n \times m$ matrix defines the maximum demand of each process. If $max[i,j]=k$, then P_i may request at most 'k' instances of resource type R_j .

Need: An $n \times m$ matrix indicates the remaining resources need of each process. If $\text{need}[i,j]=k$, then P_i may need 'k' more instances of resource type R_j to complete this task. Therefore, $\text{Need}[i,j] = \text{Max}[i,j] - \text{Allocation}[i,j]$

Safety Algorithm

1. Work and Finish be the vector of length m and n respectively, $\text{Work} = \text{Available}$ and $\text{Finish}[i] = \text{False}$.
2. Find an i such that both $\text{Finish}[i] = \text{False}$ $\text{Need} \leq \text{Work}$
If no such i exists go to step 4.
3. $\text{work} = \text{work} + \text{Allocation}$, $\text{Finish}[i] = \text{True}$;
4. if $\text{Finish}[i] = \text{True}$ for all i , then the system is in safe state.

Resource request algorithm

Let Request i be request vector for the process P_i , If request $i[j]=k$, then process P_i wants k instances of resource type R_j .

1. if $\text{Request} \leq \text{Need}$ I go to step 2. Otherwise raise an error condition.
2. if $\text{Request} \leq \text{Available}$ go to step 3. Otherwise P_i must since the resources are available.
3. Have the system pretend to have allocated the requested resources to process P_i by modifying the state as follows;

$\text{Available} = \text{Available} - \text{Request } i$;

$\text{Allocation } i = \text{Allocation} + \text{Request } i$; $\text{Need } i = \text{Need } i - \text{Request } i$;

If the resulting resource allocation state is safe, the transaction is completed and process P_i is allocated its resources. However if the state is unsafe, the P_i must wait for Request i and the old resource-allocation state is restored.

ALGORITHM:

1. Start the program.
2. Get the values of resources and processes.
3. Get the avail value.
4. After allocation find the need value.
5. Check whether it's possible to allocate.
6. If it is possible then the system is in safe state.
7. Else system is not in safety state.
8. If the new request comes then check that the system is in safety.
9. Or not if we allow the request.
10. Stop the program.

SOURCE CODE:

```
#include<stdio.h>
#include<conio.h>

int k,rq[10],i,j,count,temp=0,R[10],n,r,max[10][10],
alloc[10][10],need[10][10],tr[10],p,q,avail[10],availb[10][10],avala[10][10];

void main()
{
clrscr();
printf("enter no of process:");
scanf("%d",&n);
printf("\nenter no of resources:");
scanf("%d",&r);
for(i=1;i<=n;i++)
{
printf("process %d",i);
for(j=1;j<=r;j++)
{
printf("maximum value of resource:%d",i);
scanf("%d",&max[i][j]);
}
for(j=1;j<=r;j++)
{
printf("allocated from resource %d:",i);
scanf("%d",&alloc[i][j]);
}
}
for(i=1;i<=r;i++)
{
printf("enter total value of resource %d",i);
scanf("%d",&tr[i]);
}for(i=1;i<=n;i++)
{
for(j=1;j<=r;j++)
```

```
{
need[i][j]=max[i][j]-alloc[i][j];
}}
printf("\nresources\tallocated\tneeded\n");
for(i=1;i<=n;i++)
{printf("p%d\t",i);
for(j=1;j<=r;j++)
{printf("%d",max[i][j]);
}printf("\t");
for(j=1;j<=r;j++)
{printf("%d",alloc[i][j]);
}
printf("\t");
for(j=1;j<=r;j++)
{
printf("%d",need[i][j]);
}printf("\n");
}
R[1]=0;
for(j=1;j<=r;j++)
{
for (i=1;i<=n;i++)
{ R[j]=R[j]+alloc[i][j];
}
avail[j]=tr[j]-R[j];
}
printf("total\tavail\n");
for(i=1;i<=r;i++)
{
printf("%d",tr[i]);
} printf("\t");
for(i=1;i<=r;i++)
```

```
{
printf("%d",avail[i]);
}
for(i=1;i<=n;i++)
{ count=0;
for(j=1;j<=r;j++)
{
if(avail[j]>=need[i][j])
{ count=count+1;
if(count==r)
{
printf("the above sequence is a safe sequence");
q=i,p=i;
goto pos;
}
}temp=temp+1;
}
if(temp==n)
{
printf("\ndead lock");
exit();
}
pos:printf("\n\tavailb\tavaila\n");
printf("p%d\t",p);
for(i=1;i<=r;i++)
{
availb[p][i]=avail[i]-need[p][i];
printf("%d",availb[p][i]);
} printf("\t");
for(i=1;i<=r;i++)
{
```

```
availa[p][i]=max[p][i]+availb[p][i];
printf("%d",availa[p][i]);
}
printf("\n");
for(i=1;i<=n;i++)
{
    if(i!=q)
    { printf("p%d\t",i);
      for(j=1;j<=r;j++)
      {
          availb[i][j]=availa[p][j]-need[i][j];
          printf("%d",availb[i][j]);
          printf("\t");
          for(j=1;j<=r;j++)
          {
              availa[i][j]=availb[i][j]+max[i][j];
              printf("%d",availa[i][j]);
          }
          p=i;
      }
      printf("\n");
    }
}
printf("enter the process to be requested");
scanf("%d",&k);
printf("\nenter the request value");
for(i=1;i<=r;i++)
{
    scanf("%d",&rq[i]);
}
count=0;
for(i=1;i<=r;i++)
{
```



```
if(need[k][i]>=rq[i]&&avail[i]>=rq[i])
{
count=count+1;
if(count==r)
{
for(i=1;i<=r;i++)
{
avail[i]=avail[i]-rq[i];
}
}
}temp=0;
for(i=1;i<=n;i++)
{
count=0;
if(i!=k)
{
for(j=1;j<=r;j++)
{
if(need[i][j]>=avail[j])
{
count=count+1;
if(count==r)
{
printf("request granted\n");
exit();
}
}
}
}temp=temp+1;
}
```

```
if(temp==n)
{
printf("request rejected");
exit();
}
getch();
}
```

OUTPUT:

```

E:\krishna\bankers.exe
enter no of process:4
enter no of resources:3
process 1maximum value of resource:13
maximum value of resource:12
maximum value of resource:12
allocated from resource 1:1
allocated from resource 1:0
allocated from resource 1:0
process 2maximum value of resource:26
maximum value of resource:21
maximum value of resource:23
allocated from resource 2:5
allocated from resource 2:1
allocated from resource 2:1
process 3maximum value of resource:33
maximum value of resource:31
maximum value of resource:34
allocated from resource 3:2
allocated from resource 3:1
allocated from resource 3:1
process 4maximum value of resource:44
maximum value of resource:42
maximum value of resource:42
allocated from resource 4:0
allocated from resource 4:0
allocated from resource 4:2
enter total value of resource 19
enter total value of resource 23
enter total value of resource 36

resources      allocated      needed
p1      322      100      222
p2      613      511      102
p3      314      211      103
p4      422      002      420
total    avail
936      112the above sequence is a safe sequence
availb availa
p2      010      623
p1      401      723
p3      620      934
p4      514      936
enter the process to be requested1

enter the request value1 0 0
request rejected

```

VIVA-VOCE:**1.What is deadlock?**

Deadlock is a situation when two or more processes wait for each other to finish and none of them ever finish. Consider an example when two trains are coming toward each other on same track and there is only one track, none of the trains can move once they are in front of each other. Similar situation occurs in operating systems when there are two or more processes hold some resources and wait for resources held by other(s).

2.What are the necessary conditions for deadlock?

Mutual Exclusion: There is a resource that cannot be shared.

Hold and Wait: A process is holding at least one resource and waiting for another resource which is with some other process.

No Preemption: The operating system is not allowed to take a resource back from a process until process gives it back.

Circular Wait: A set of processes are waiting for each other in circular form.

3.What is resource allocation graph?

A resource allocation graph tracks which resource is held by which process and which process is waiting for a resource of a particular type. It is very powerful and simple tool to illustrate how interacting processes can deadlock. If a process is *using* a resource, an arrow is drawn from the resource node to the process node. If a process is *requesting* a resource, an arrow is drawn from the process node to the resource node.

4.Explain safe state?

A **state** is **safe** if the system can allocate all resources requested by all processes (up to their stated maximums) without entering a deadlock **state**. ... If a **safe** sequence does not exist, then the system is in an unsafe **state**, which MAY lead to deadlock.

5.What is a critical section?

It is a section of code which can be executed only by one process at a time.

6.What is the purpose of Banker's algorithm?

The banker's algorithm is a resource allocation and deadlock avoidance algorithm that tests for safety by simulating the allocation for predetermined maximum possible amounts of all resources, then makes an "s-state" check to test for possible activities, before deciding whether allocation should be allowed to continue

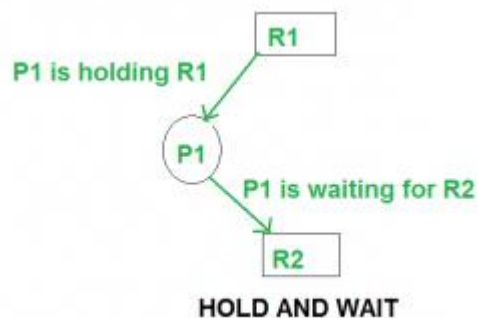
13) Write a C program to simulate Bankers algorithm for Deadlock Prevention

We can prevent Deadlock by eliminating any of the above four conditions.

Eliminate Mutual Exclusion

It is not possible to dis-satisfy the mutual exclusion because some resources, such as the tap drive and printer, are inherently non-shareable.

1. Allocate all required resources to the process before the start of its execution, this way hold and wait condition is eliminated but it will lead to low device utilization. for example, if a process requires printer at a later time and we have allocated printer before the start of its execution printer will remain blocked till it has completed its execution.
2. The process will make a new request for resources after releasing the current set of resources. This solution may lead to starvation.



Eliminate No Preemption

Preempt resources from the process when resources required by other high priority processes.

Eliminate Circular Wait

Each resource will be assigned with a numerical number. A process can request the resources only in increasing order of numbering. For Example, if P1 process is allocated R5 resources, now next time if P1 ask for R4, R3 lesser than R5 such request will not be granted, only request for resources more than R5 will be granted.

Source Code:

```
include<stdio.h>
#define size 20
void main()
{
    int i,j,temp,sum;
    int pid,p[size],rid,valid[size],totalres[size],res[size][size],allocated[size][size],
    avail[size],need[size][size];
    //clrscr();
    printf("Deadlock Prevention ");
    printf("ENTER THE NO. OF PROCESSES: ");
    scanf("%d",&pid);
    printf("ENTER THE NO. OF RESOURCES: ");
    scanf("%d",&rid);
    for(i=0;i<rid;i++)
```

```

{
    printf("\nENTER TOTAL VALUE OF RESOURCE %d: ",i+1);
    scanf("%d",&totalres[i]);
}
for(i=0;i<pid;i++)
{
    p[i]=i+1;
    valid[i]=1;
    printf("\n-----\nPROCESS %d",i+1);
    for(j=0;j<rid;j++)
    {
        printf("\nMAXIMUM VALUE FOR RESOURCE %d: ",j+1);
        scanf("%d",&res[i][j]);
    }
    for(j=0;j<rid;j++)
    {
        printf("\nALLOCATED FROM RESOURCE %d: ",j+1);
        scanf("%d",&allocated[i][j]);
    }
}
for(i=0;i<rid;i++)
{
    sum=0;
    for(j=0;j<pid;j++)
    {
        sum = sum + allocated[j][i];
    }
    if(sum>totalres[i])
    {
        printf("\nDeadlock occured");
        getch();
        return;
    }
    avail[i] = totalres[i]-sum;
}
for(i=0;i<pid;i++)
{
    for(j=0;j<rid;j++)
    {
        need[i][j] = res[i][j] - allocated[i][j];
    }
}
printf("\nPID\tResources\tAllocated\tNeed\n---\t-----\t-----\t----");
for(i=0;i<pid;i++)
{
    printf("\nP%d\t",i+1);

```

```
    for(j=0;j<rid;j++)
    {
        printf("%d",res[i][j]);
    }
    printf("\t\t");
    for(j=0;j<rid;j++)
    {
        printf("%d",allocated[i][j]);
    }
    printf("\t\t");
    for(j=0;j<rid;j++)
    {
        printf("%d",need[i][j]);
    }
}
printf("\nTotal Resources: ");
for(j=0;j<rid;j++)
{
    printf("%d",totalres[j]);
}
printf("\nAvailable Resources: ");
for(j=0;j<rid;j++)
{
    printf("%d",avail[j]);
}
printf("\n\nPID\tAVAIL BEFORE\tAVAIL AFTER\n");
pos:for(i=0;i<pid;i++)
{
    temp=0;
    for(j=0;j<rid;j++)
    {
        if(need[i][j]>avail[j]||valid[i]==0)
        {
            temp = 1;
            break;
        }
    }
}
if(temp==0)
{
    printf("\nP%d\t",p[i]);
    for(j=0;j<rid;j++)
    {
        avail[j] = avail[j] - need[i][j];
        printf("%d",avail[j]);
    }
    printf("\t\t");
    for(j=0;j<rid;j++)
```

```

        {
            avail[j] = avail[j] + res[i][j];
            printf("%d",avail[j]);
        }
        valid[i]=0;
        goto pos;
    }
}
for(i=0;i<pid;i++)
    if(valid[i]!=0)
    {
        printf("\n\nDeadlock Occurred");
        getch();
        return;
    }
printf("\n\nTHE ABOVE SEQUENCE IS A SAFE SEQUENCE");
getch();
}

```

OUTPUT:

Enter the matrix 2 2 2

Enter the claim matrix 2 2 2

Enter allocated matrix 2 2 2 2

The need matrix

0 0

0 0

Enter available matrix 0 2 3

Claim matrix:

2 2

2 2

allocated matrix:

2 2

2 2

available matrix: 0 0 0

VIVA-VOCE:

1.What is Deadlock in the Operating System?

Deadlock is the condition that occurs when two processes wait for each other to complete and halt without proceeding further. Here two processes wait for the resources acquired by each other.

2.What are the Necessary Conditions to Occur Deadlock?

There are multiple necessary conditions to occur deadlock in the operating system. Even if one condition does not get satisfied, deadlock cannot occur. These conditions are as follows.

- Mutual Exclusion
- Hold and Wait
- No Preemption
- Circular Wait

If you want not to occur deadlock, you should have to care not to satisfy any one of the conditions mentioned above.

3.What are the different types of Deadlock Types and Handling Strategies?

There are multiple deadlocks handling strategies to avoid deadlock. We can prevent the deadlock not to occur and it is the optimal solution. But preventing the system not to occur deadlock is not easy as we require prior information about processes as well as resources.

There are four strategies to handle deadlock. (Types of Deadlock)

- Prevention of Deadlock
- Avoidance of Deadlock
- Deadlock Detection and Recover
- Deadlock Ignorance

4.What is Deadlock Prevention?

We take care of eliminating any deadlock causing condition:

1. Circularwait

Processes form the cycle of waiting for relationships.

2. Holdandwait

Processes hold up resources which they have and w8 on resources they want.

3. NoPreemption

Once, a process is in execution it is not preempted.

4. MutualExclusion

When a process is executing on a variable no other process can access that variable.

5.What is the difference between Deadlock Prevention and Deadlock Avoidance in an operating system?

In deadlock prevention, we restrict resource requests to prevent four conditions of deadlock, However, deadlock prevention reduces resource utilization and slows down the execution of processes.

Deadlock avoidance allows for more concurrently than prevention. With deadlock avoidance in the operating system, a decision is made dynamically whether the current resource allocation request (if granted) will potentially lead to a deadlock or not lead to deadlock. If there is no possibility of deadlock, the request is fulfilled otherwise not.

14. Write C programs to simulate the following Page Replacement Techniques:

a) FIFO b) LRU c) OPTIMAL

THEORY:

a) FIFO (First in First Out) algorithm: FIFO is the simplest page replacement algorithm, the idea behind this is, “Replace a page that page is oldest page of main memory” or “Replace the page that has been in memory longest”. FIFO focuses on the length of time a page has been in the memory rather than how much the page is being used.

b) LRU (Least Recently Used): the criteria of this algorithm is “Replace a page that has been used for the longest period of time”. This strategy is the page replacement algorithm looking backward in time, rather than forward.

c) LFU (Least Frequently Used): The least frequently used algorithm “select a page for replacement, if the page has not been used for the often in the past” or “Replace page that page has smallest count” for this algorithm each page maintains as counter which counter value shows the least count, replace that page. The frequency counter is reset each time is page is loaded.

A)FIFO Algorithm:

1. Start
2. Read the number of frames
3. Read the number of pages
4. Read the page numbers
5. Initialize the values in frames to -1
6. Allocate the pages in to frames in First in first out order.
7. Display the number of page faults.
8. Stop

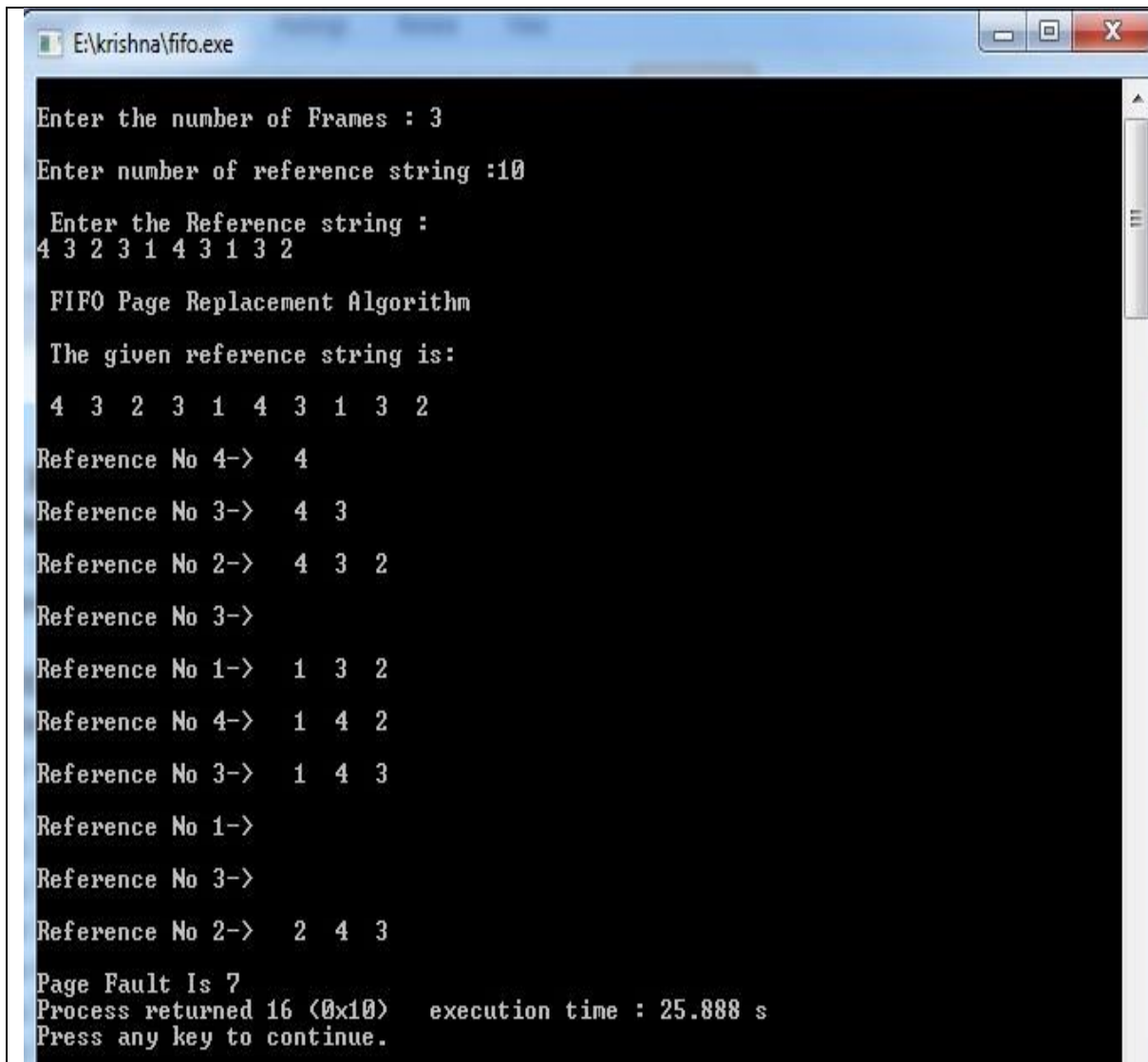
SOURCE CODE:

```
#include<stdio.h>

void main()
{
    int i,j,n,a[50],frame[10],fno,k,avail,pagefault=0;
    printf("\nEnter the number of Frames : ");
    scanf("%d",&fno);
    printf("\nEnter number of reference string :");
    scanf("%d",&n);
    printf("\n Enter the Reference string :\n");
    for(i=0;i<n;i++)
        scanf("%d",&a[i]);
```

```
for(i=0;i<fno;i++)
    frame[i]= -1;
j=0;
printf("\n FIFO Page Replacement Algorithm\n\n The given reference string is:\n\n");
for(i=0;i<n;i++)
{
    printf(" %d ",a[i]);
}
printf("\n");
for(i=0;i<n;i++)
{
    printf("\nReference No %d-> ",a[i]);
    avail=0;
    for(k=0;k<fno;k++)
        if(frame[k]==a[i])
            avail=1;
    if (avail==0)
    {
        frame[j]=a[i];
        j = (j+1) % fno;
        pagefault++;
        for(k=0;k<fno;k++)
            if(frame[k]!=-1)
                printf(" %2d",frame[k]);
    }
    printf("\n");
}
printf("\nPage Fault Is %d",pagefault);
}
```

OUTPUT:



```
E:\krishna\fifo.exe

Enter the number of Frames : 3
Enter number of reference string :10
Enter the Reference string :
4 3 2 3 1 4 3 1 3 2

FIFO Page Replacement Algorithm
The given reference string is:
4 3 2 3 1 4 3 1 3 2

Reference No 4-> 4
Reference No 3-> 4 3
Reference No 2-> 4 3 2
Reference No 3->
Reference No 1-> 1 3 2
Reference No 4-> 1 4 2
Reference No 3-> 1 4 3
Reference No 1->
Reference No 3->
Reference No 2-> 2 4 3

Page Fault Is 7
Process returned 16 (0x10) execution time : 25.888 s
Press any key to continue.
```

B) LRU Algorithm:

1. Start
2. Read the number of frames
3. Read the number of pages
4. Read the page numbers
5. Initialize the values in frames to -1

6. Allocate the pages in to frames by selecting the page that has not been used for the longest period of time.
7. Display the number of page faults.
8. Stop

SOURCE CODE:

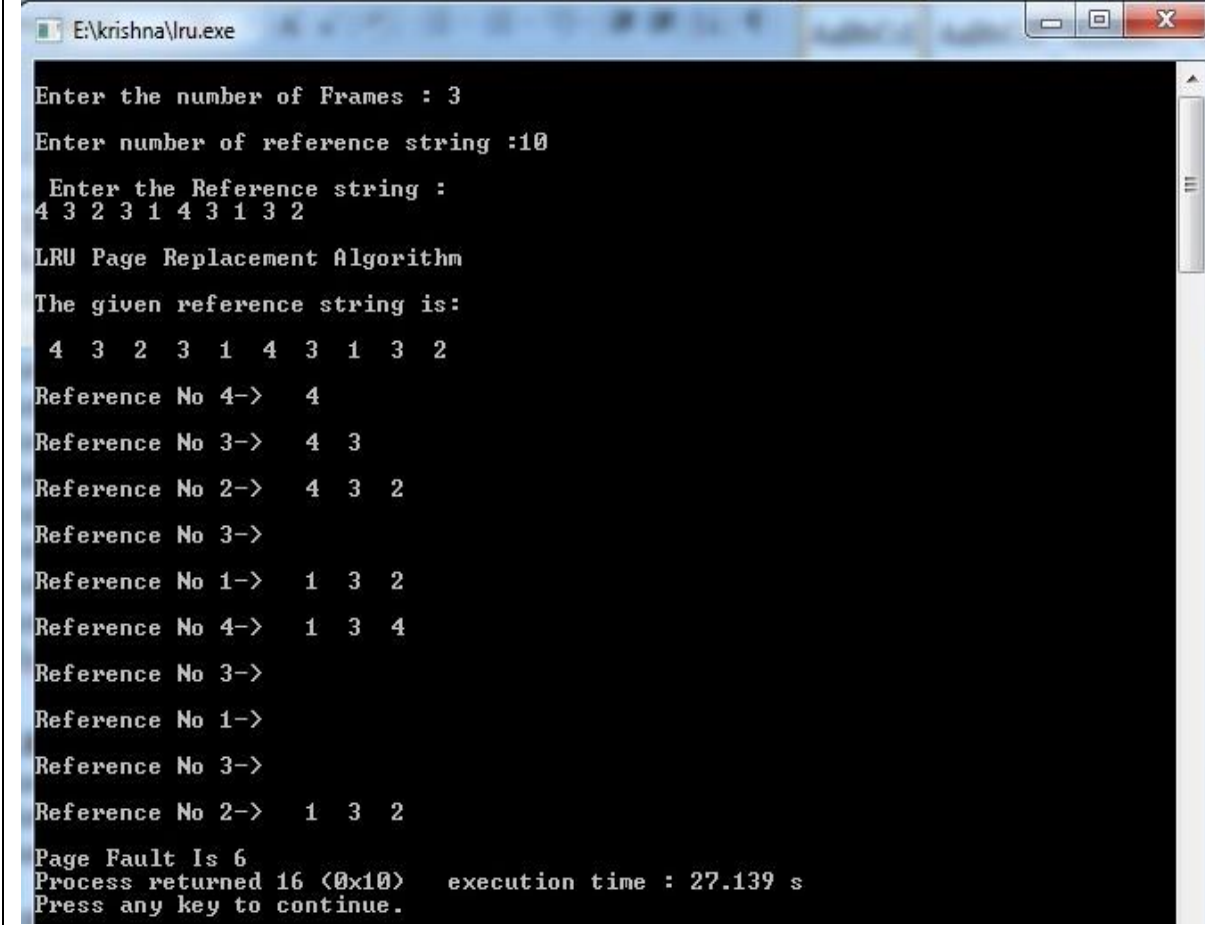
```
#include<stdio.h>

void main()
{
    int i,j,l,max,n,a[50],frame[10],flag,fno,k,avail,pagefault=0,lru[10];
    printf("\nEnter the number of Frames : ");
    scanf("%d",&fno);
    printf("\nEnter number of reference string :");
    scanf("%d",&n);
    printf("\n Enter the Reference string :\n");
    for(i=0;i<n;i++)
        scanf("%d",&a[i]);
    for(i=0;i<fno;i++)
    {
        frame[i]= -1;
        lru[i] = 0;
    }
    printf("\nLRU Page Replacement Algorithm\n\nThe given reference string is:\n\n");
    for(i=0;i<n;i++)
    {
        printf(" %d ",a[i]);
    }
    printf("\n");
    j=0;
    for(i=0;i<n;i++)
    {
        max = 0;
        flag=0;
```

```
printf("\nReference No %d-> ",a[i]);
avail=0;
for(k=0;k<fno;k++)
    if(frame[k]==a[i])
    {
        avail=1;
        lru[k]=0;
        break;
    }
if(avail==1)
{
    for(k=0;k<fno;k++)
        if(frame[k]!=-1)
            ++lru[k];
    max = 0;
    for(k=1;k<fno;k++)
        if(lru[k]>lru[max])
            max = k;
    j = max;
}
if(avail==0)
{
    lru[j]=0;
    frame[j]=a[i];
    for(k=0;k<fno;k++)
    {
        if(frame[k]!=-1)
            ++lru[k];
        else
        {
            j = k;
            flag = 1;
        }
    }
}
```

```
        break;
    }
}
if(flag==0){
    max = 0;
    for(k=1;k<fno;k++)
        if(lru[k]>lru[max])
            max = k;
    j = max;
}
pagefault++;
for(k=0;k<fno;k++)
    if(frame[k]!=-1)
        printf(" %2d",frame[k]);
}
printf("\n");
}
printf("\nPage Fault Is %d",pagefault);
}
```

OUTPUT:



```
E:\krishna\lru.exe
Enter the number of Frames : 3
Enter number of reference string :10
Enter the Reference string :
4 3 2 3 1 4 3 1 3 2
LRU Page Replacement Algorithm
The given reference string is:
4 3 2 3 1 4 3 1 3 2
Reference No 4-> 4
Reference No 3-> 4 3
Reference No 2-> 4 3 2
Reference No 3->
Reference No 1-> 1 3 2
Reference No 4-> 1 3 4
Reference No 3->
Reference No 1->
Reference No 3->
Reference No 2-> 1 3 2
Page Fault Is 6
Process returned 16 (0x10) execution time : 27.139 s
Press any key to continue.
```

C) Optimal Algorithm:

1. Start
2. Read the number of frames
3. Read the number of pages
4. Read the page numbers
5. Initialize the values in frames to -1
6. Allocate the pages in to frames by selecting the page that will not be used for the longest period of time.
7. Display the number of page faults.
8. Stop

SOURCE CODE:

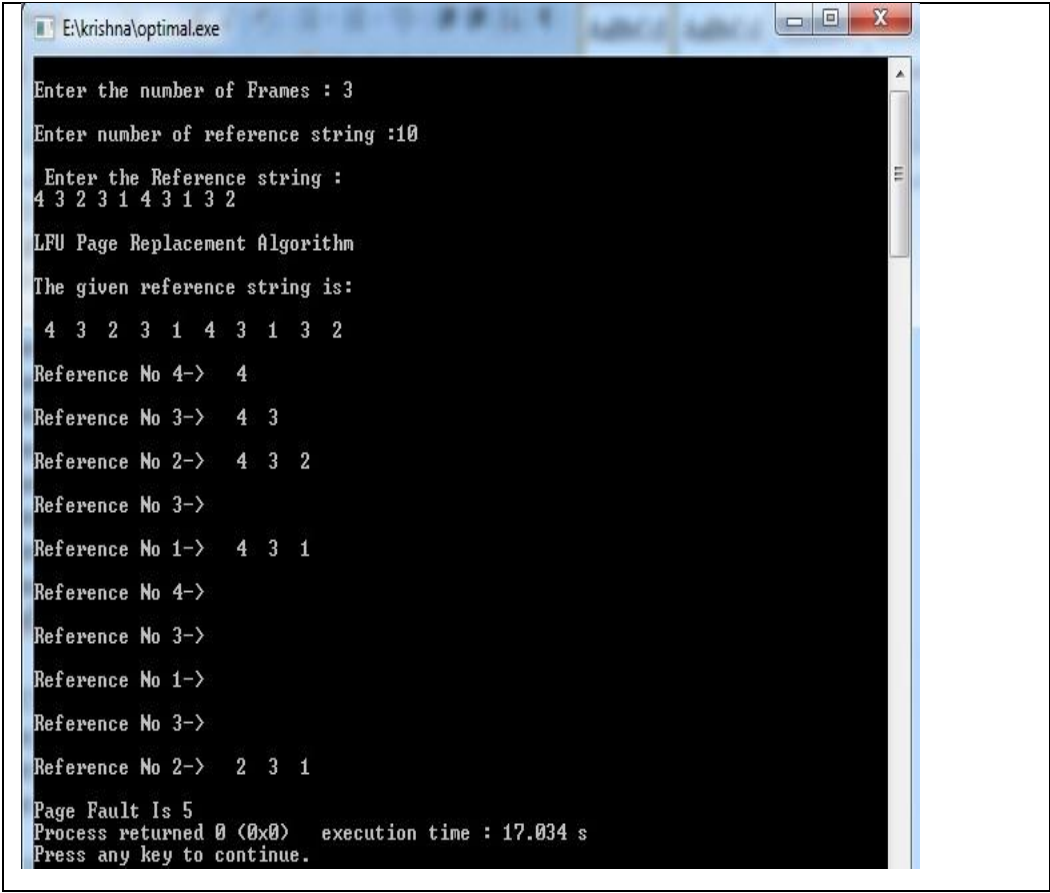
```
#include<stdio.h>

int main()
{
    int i,j,l,min,flag1,n,a[50],temp,frame[10],flag,fno,k,avail,pagefault=0,opt[10];
    printf("\nEnter the number of Frames : ");
    scanf("%d",&fno);
    printf("\nEnter number of reference string :");
    scanf("%d",&n);
    printf("\n Enter the Reference string :\n");
    for(i=0;i<n;i++)
        scanf("%d",&a[i]);
    for(i=0;i<fno;i++)
    {
        frame[i]= -1;
        opt[i]=0;
    }
    printf("\n Optimal Page Replacement Algorithm\n\nThe given reference string is:\n\n");
    for(i=0;i<n;i++)
        printf(" %d ",a[i]);
    printf("\n");
    j=0;
```

```
for(i=0;i<n;i++)
{
    flag=0;
    flag1=0;
    printf("\nReference No %d-> ",a[i]);
    avail=0;
    for(k=0;k<fno;k++)
        if(frame[k]==a[i])
        {
            avail=1;
            break;
        }
    if(avail==0)
    {
        temp = frame[j];
        frame[j]=a[i];
        for(k=0;k<fno;k++)
        {
            if(frame[k]==-1)
            {
                j = k;
                flag = 1;
                break;
            }
        }
        if(flag==0)
        {
            for(k=0;k<fno;k++)
            {
                opt[k]=0;
                for(l=i;l<n;l++)
                {
```

```
        if(frame[k]==a[l])
        {
            flag1 = 1;
            break;
        }
    }
    if(flag1==1)
        opt[k] = 1-i;
    else
    {
        opt[k] = -1;
        break;
    }
}
min = 0;
for(k=0;k<fno;k++)
    if(opt[k]<opt[min]&&opt[k]!=-1)
        min = k;
    else if(opt[k]==-1)
    {
        min = k;
        frame[j] = temp;
        frame[k] = a[i];
        break;
    }
j = min;
}
pagefault++;
for(k=0;k<fno;k++)
    if(frame[k]!=-1)
        printf(" %2d",frame[k]);
}
```

```
    printf("\n");  
}  
printf("\nPage Fault Is %d",pagefault);  
return 0;  
}
```

OUTPUT:

```
E:\krishna\optimal.exe  
Enter the number of Frames : 3  
Enter number of reference string :10  
Enter the Reference string :  
4 3 2 3 1 4 3 1 3 2  
LRU Page Replacement Algorithm  
The given reference string is:  
4 3 2 3 1 4 3 1 3 2  
Reference No 4-> 4  
Reference No 3-> 4 3  
Reference No 2-> 4 3 2  
Reference No 3->  
Reference No 1-> 4 3 1  
Reference No 4->  
Reference No 3->  
Reference No 1->  
Reference No 3->  
Reference No 2-> 2 3 1  
Page Fault Is 5  
Process returned 0 (0x0) execution time : 17.034 s  
Press any key to continue.
```

VIVA-VOCE:

1.What is a page fault?

A page fault occurs when a program attempts to access data or code that is in its address space, but is not currently located in the system RAM.

2. Why do we use page replacement algorithms?

Whenever a new page is referred and not present in memory, page fault occurs and **Operating System replaces one of the existing pages with newly needed page.**

3. Explain virtual memory?

Virtual memory is a memory management capability of an OS that uses hardware and software to allow a computer to compensate for physical memory shortages by temporarily transferring data from random access memory (RAM) to disk storage. Virtual address space is increased using active memory in RAM and inactive memory in hard disk drives (HDDs) to form contiguous addresses that hold both the application and its data.

4. Explain LRU with example.

Consider the following reference string as an example for better understanding of the LRU algorithm.

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0
1 7 0 1

Calculate the number of page faults when the Least Recently Used (LRU) page replacement policy is used. Also consider the page frame size as three.

Soln:

Reference string:

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

7	7	7	2	2	2	2	4	4	4	0	0	0	1	1	1	1	1	1	1
	0	0	0	0	0	0	0	0	3	3	3	3	3	3	0	0	0	0	0
		1	1	1	3	3	3	2	2	2	2	2	2	2	2	2	7	7	7
F	F	F	F		F		F	F	F	F			F		F		F		