.

.

## OPERATING SYSTEMS LAB PROGRAMS

**Week-10:**

**NAME OF THE EXPERIMENT:CPU Scheduling Techniques FCFS , Priority**

**AIM:Write C Programs to simulate the following CPU scheduling algorithms:**

**a)FCFS b) Priority**

### 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**

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

**Turn around time for Process(n)= waiting time of Process(n)+ Burst time forprocess(n)**

**Step 6: Calculate**

**Average waiting time = Total waiting Time / Number of process**

**Average Turnaround time = Total Turnaround Time / Number of process**

**Step 7: Stop the process**

**SOURCE CODE:**

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

**Output:**

**b) 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**

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

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

**Step 6: Calculate**

**Average waiting time = Total waiting Time / Number of process**

**Average Turnaround time = Total Turnaround Time / Number of process**

**Step 7: Stop the process**

**SOURCE CODE:**

```
#include<stdio.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;

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);
}
```

**Output:**

---

**VIVA-VOCE**

**1.What is an Operating system?**

**Operating System (OS), program that manages a computer's resources, especially the allocation of those resources among other programs. Typical resources include the central processing unit (CPU), computer memory, file storage, input/output (I/O) devices, and network connections.**

## 2.What is a process ?

**A process is an ‘active’ entity, instead of a program, which is considered a ‘passive’ entity. A single program can create many processes when run multiple times; for example, when we open a .exe or binary file multiple times, multiple instances begin (multiple processes are created)**

**3.What Are The Advantages of A Multiprocessor System?**

**The advantages of the multiprocessing system are: Increased Throughput − By increasing the number of processors, more work can be completed in a unit time. Cost Saving − Parallel system shares the memory, buses, peripherals etc. Multiprocessor system thus saves money as compared to multiple single systems.**

**4. Explain starvation and Aging**

**Starvation is the problem that occurs when high priority processes keep executing and low priority processes get blocked for indefinite time. In heavily loaded computer system, a steady stream of higher-priority processes can prevent a low-priority process from ever getting the CPU.**

**Aging is a technique of gradually increasing the priority of processes that wait in the system for a long time. For example, if priority range from 127(low) to 0(high), we could increase the priority of a waiting process by 1 Every 15 minutes.**

**5.What are the functions of operating system?**

**An operating system has three main functions: (1) manage the computer's resources, such as the central processing unit, memory, disk drives, and printers, (2) establish a user interface, and (3) execute and provide services for applications software.**

## Week-11:

**NAME OF THE EXPERIMENT:CPU Scheduling Techniques SJF, Round Robin**

**AIM:Write C Programs to simulate the following CPU scheduling algorithms:**

**a)SJF b) Round Robin**

**a) 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**

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

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

**Step 7: Calculate**

**Average waiting time = Total waiting Time / Number of process**

**Average Turnaround time = Total Turnaround Time / Number of process**

**Step 8: Stop the process**

**SOURCE CODE:**

```
#include<stdio.h>

void main(){

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

float awt,atat;

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);

}
```

**OUTPUT :**

**b) 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**

**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)**

**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**

**Average waiting time = Total waiting Time / Number of process**

**Average Turnaround time = Total Turnaround Time / Number of process**

**Step 8: Stop the process**

**SOURCE CODE:**

```
#include<stdio.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;
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);

}
```

**OUTPUT :**

**VIVA –VOCE**

**1. List different CPU Scheduling algorithms.**

**The different CPU algorithms are:**

- **First Come First Serve.**
- **Shortest Job First.**
- **Shortest Remaining Time First.**
- **Round Robin Scheduling.**
- **Priority Scheduling.**
- **Multilevel Queue Scheduling.**
- **Multilevel Feedback Queue Scheduling.**

**2. What is FCFS Scheduling?**

**First Come First Serve (FCFS) is an operating system scheduling algorithm that automatically executes queued requests and processes in order of their arrival. It is the easiest and simplest CPU scheduling algorithm. In this type of algorithm, processes which requests the CPU first get the CPU allocation first.**

**3. What is SJF Scheduling?**

**Shortest Job First (SJF) is an algorithm in which the process having the smallest execution time is chosen for the next execution. This scheduling method can be preemptive or non-preemptive. It significantly reduces the average waiting time for other processes awaiting execution.**

**4. What is RR Scheduling?**

**Round Robin is a CPU scheduling algorithm where each process is assigned a fixed time slot in a cyclic way. It is simple, easy to implement, and starvation-free as all processes get fair share of CPU. One of the most commonly used technique in CPU scheduling as a core.**

**5. What is Priority Scheduling?**

**Priority Scheduling is a method of scheduling processes that is based on priority. In this algorithm, the scheduler selects the tasks to work as per the priority. The processes with higher priority should be carried out first, whereas jobs with equal priorities are carried out on a round-robin or FCFS basis.**

## Week-12:

**NAME OF THE EXPERIMENT: File allocation techniques**

**AIM: Write C programs to simulate the following Fileallocation strategies**

**a) Sequential b)Linked c)Indexed**

**a) Sequential Algorithm:**

**Step 1: Start the program.**

**Step 2: Get the number of memory partition and their sizes.**

**Step 3: Get the number of processes and values of block size for each process.**

**Step 4: First fit algorithm searches the entire memory block until a hole which is big enough is encountered. It allocates that memory block for the requesting process.**

**Step 5: Best-fit algorithm searches the memory blocks for the smallest hole which can be allocated to requesting process and allocates if.**

**Step 6: Worst fit algorithm searches the memory blocks for the largest hole and allocates it to the process.**

**Step 7: Analyses all the three memory management techniques and display the best algorithm which utilizes the memory resources effectively and efficiently.**

**Step 8: Stop the program**

**SOURCE CODE:**

```
#include<stdio.h>

main()

{

int n,i,j,b[20],sb[20],t[20],x,c[20][20];

printf("Enter 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 %d \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]);
}
```

**Output:**

**b) LinkedAlgorithm:**

---

**Step 1: Start the program.**

**Step 2: Read the number of files**

**Step 3: For each file, read the file name, starting block and number of blocks and block numbers of the file.**

**Step 4: Start from the starting block and link each block of the file to the next block in a linked list fashion.**

**Step 5: Display the file name, starting block, size of the file & the blocks occupied by the file.**

**Step 6: Stop the program**

**SOURCE CODE:**

```
#include<stdio.h>

struct file

{

char fname[10];

int start,size,block[10];

}f[10];

main()

{

int i,j,n;

printf("Enter 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]);
}}
```

**Output:**

---

**c) IndexedAlgorithm:**

**Step 1: Start the program.**

**Step2: Read the number of files**

**Step 3: Read the index block for each file.**

**Step 4: For each file, read the number of blocks occupied and number of blocks of the file.**

**Step 5: Link all the blocks of the file to the index block.**

**Step 6: Display the file name, index block, and the blocks occupied by the file.**

**Step 7: Stop the program**

**SOURCE CODE:**

```
#include<stdio.h>
main()
{
int n,m[20],i,j,sb[20],b[20][20],x;
printf("\nEnter no. of files:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("\nEnter index block of file%d:",i+1);
```

```
scanf("%d",&sb[i]);
printf("\nEnter length of file%d:",i+1);
scanf("%d",&m[i]);
printf("enter blocks of file%d:",i+1);
for(j=0;j<m[i];j++)
scanf("%d",&b[i][j]);
}
printf("\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("\nEnter file name:");
scanf("%d",&x);
printf("\nfile name is:%d",x);
printf("\nIndex is:%d",sb[x-1]);
printf("\nBlocks occupied are:");
for(j=0;j<m[x-1];j++)
printf("%4d",b[x-1][j]);
}
```

**OUTPUT:**

**VIVA-VOCE**

**1. List File access methods.**

**There are three ways to access a file into a computer system: Sequential-Access, Direct Access, Index sequential Method.**

- **Sequential Access – It is the simplest access method. ...**
- **Direct Access – Another method is direct access method also known as relative access method. ...**
- **Index sequential method**

**2. Explain Sequential file allocation method.**

**In this allocation strategy, each file occupies a set of contiguously blocks on the disk. This strategy is best suited. For sequential files, the file allocation table consists of a single entry for each file. It shows the filenames, starting block of the file and size of the file.**

**3. Explain Linked file allocation method.**

**Each file is a linked list of disk blocks which need not be contiguous. The disk blocks can be scattered anywhere on the disk.**
**The directory entry contains a pointer to the starting and the ending file block. Each block contains a pointer to the next block occupied by the file.**

**4. Explain Indexed file allocation method.**

**Indexed allocation scheme stores all the disk pointers in one of the blocks called as indexed block. Indexed block doesn't hold the file data, but it holds the pointers to all the disk blocks allocated to that particular file. Directory entry will only contain the index block address.**

**5. What is file system mounting?**

**Mounting a file system attaches that file system to a directory (mount point) and makes it available to the system. The root ( / ) file system is always mounted. Any other file system can be connected or disconnected from the root ( / ) file system.**

## Week-13:

**NAME OF THE EXPERIMENT:Memory management techniques**

**AIM: Write a C program to simulate the following memory management techniques**

**a) Paging** **b) Segmentation**

**a) PAGING Algorithm:**

**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>
void main(){
    int i,j,temp,framearr[20],pages,pageno,frames,memsize,log,pagesize,prosize,base;
    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; //lhs 'temp' is physical address rhs and 'temp' is frame num
    printf("\nPhysical address is : %d",temp);
    }
```

**Output:**

**b) SEGMENTATION ALGORITHM**

**Step 1: Start the program.**

**Step 2: Get the number of segments.**

**Step 3: Get the base address and length for each segment.**

**Step 4: Get the logical address.**

**Step 5: Check whether the segment number is within the limit, if not display the error message.**

**Step 6: Check whether the byte reference is within the limit, if not display the error message.**

**Step 7: Calculate the physical memory and display it.**

**Step 8: Stop the program.**

**SOURCE CODE:**

```
#include<stdio.h>

void main(){

    int i,j,m,size,val[10][10],badd[20],limit[20],ladd;

    printf("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]);
 }  }}
```

**OUTPUT:**

**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 non contiguous. It avoids the considerable problem of having to fit varied sized memory chunks onto the backing store.**

**2.What is fragmentation?**

**Fragmentation is an unwanted problem in the operating system in which the processes are loaded and unloaded from memory, and free memory space is fragmented. Processes can't be assigned to memory blocks due to their small size, and the memory blocks stay unused.**

## 3.What is thrashing?

**Thrashing is caused by under allocation of the minimum number of pages required by a process, forcing it to continuously page fault.**

**4.Differentiate between logical and physical address.**

**Logical Address is generated by CPU while a program is running. The logical address is virtual address as it does not exist physically, therefore, it is also known as Virtual Address.**

**Physical Address identifies a physical location of required data in a memory. The user never directly deals with the physical address but can access by its corresponding logical address.**

**5.Explain internal fragmentation and external fragmentation.**

**Internal Fragmentation**

**When a process is allocated to a memory block, and if the process is smaller than the amount of memory requested, a free space is created in the given memory block. Due to this, the free space of the memory block is unused, which causes internal fragmentation.**

**External Fragmentation**

**External fragmentation happens when a dynamic memory allocation method allocates some memory but leaves a small amount of memory unusable. The quantity of available memory is**

substantially reduced if there is too much external fragmentation. There is enough memory space to complete a request, but it is not contiguous. It's known as external fragmentation

## Week-14:

**NAME OF THE EXPERIMENT:Page Replacement Techniques**

**AIM: Write C programs to simulate the following Page Replacement Techniques:**

**a) FIFO b) LRU c)OPTIMAL**

**a)FIFO Algorithm:**

**Step1: Start**

**Step2: Read the number of frames**

**Step3: Read the number of pages**

**Step4: Read the page numbers**

**Step5: Initialize the values in frames to -1**

**Step6:Allocate the pages in to frames in First in first out order.**

**Step7: Display the number of page faults.**

**Step8: 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:**

**b) LRU Algorithm:**

**Step1: Start**

**Step2: Read the number of frames**

**Step3: Read the number of pages**

**Step4: Read the page numbers**

**Step5: Initialize the values in frames to -1**

**Step6: Allocate the pages in to frames by selecting the page that has not been used for the longest period of time.**

**Step7: Display the number of page faults.**

**Step8: 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 :**

**c) Optimal Algorithm:**

**Step1: Start**

**Step2: Read the number of frames**

**Step3: Read the number of pages**

**Step4: Read the page numbers**

**Step5: Initialize the values in frames to -1**

**Step6: Allocate the pages in to frames by selecting the page that will not be used for the longest period of time.**

**Step7: Display the number of page faults.**

**Step8: 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("\nLFU 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] = l-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:**

**VIVA-VOCE**

**1.Why do we use page replacement algorithms?**

**Page replacement algorithms are an important part of virtual memory management and it helps the OS to decide which memory page can be moved out, making space for the currently needed page.However, the ultimate objective of all page replacement algorithms is to reduce the number of page faults**

**2.Which is best page replacement algorithm and why?**

**LRU resulted to be the best algorithm for page replacement to implement. LRU maintains a linked list of all pages in the memory, in which, the most recently used page is placed at the front, and the least recently used page is placed at the rear.**

**3.Explain LRU algorithm.**

**LRU stands for Least Recently Used.LRU replaces the line in the cache that has been in the cache the longest with no reference to it. It works on the idea that the more recently used blocks are more likely to be referenced again.**

**4. When does a page fault occur?**

**Page fault occurs when a requested page is mapped in virtual address space but not present in memory.**

**5.What are page replacement algorithms in OS?**

**1. First In First Out (FIFO)**

**2. Optimal Page replacement**

**3. Least Recently Used**