

**Guru Tegh Bahadur Institute of Technology**



# **Operating Systems**

**Lab Manual**

**CIC-353**

Academic Plan for 5th Semester			
Operating Systems			
Lecture	Topic	Reference Books	
L1, L2	<b>INTRODUCTION</b>	[T1] [T2]	
	1.1	What is an Operating System, Simple Batch Systems	[R2][R3]
	1.2	Multiprogrammed Batches systems	
	1.3	Time-Sharing Systems	
	1.4	Personal-computer systems	
	1.5	Parallel systems	
	1.6	Distributed Systems	
	1.7	Real-Time Systems	
	1.8	OS – A Resource Manager	
L3, L4	<b>MEMORY MANAGEMENT</b>		
	2.1	Memory Organization, Memory Hierarchy, Memory Management Strategies	[T1] [T2]
	2.2	Contiguous versus non- Contiguous memory allocation, Partition Management Techniques	[R2][R3]
	2.3	Logical versus Physical Address space,swapping	
L5	2.5	Paging	
L6, L7	2.6	Segmentation	
	2.7	Segmentation with Paging	
L8, L9	<b>VIRTUAL MEMORY</b>	[T1] [T2]	
	3.1	Demand Paging	[R2][R3]
	3.2	Page Replacement	
	3.3	Page-replacement Algorithms	
L10	3.4	Performance of Demand Paging	
	3.5	Allocation of Frames	
	3.6	Thrashing	
	3.8	Demand Segmentation and Overlay concepts	
L11, L12, L13,L14	<b>PROCESSES</b>	[T1][T2][R3]	
	4.1	Introduction, Process states, process management	
	4.2	Interrupts, Interprocess Communication	
	4.3	Threads: Introduction, Thread states, Thread Operation, Threading Models.	
L15, L16, L17	<b>CPU SCHEDULING</b>	[T1][T2][R3]	
	5.1	Scheduling levels, pre emptive vs no pre emptive scheduling,priorities, scheduling objective	
	5.2	Scheduling Criteria	
	5.3	Scheduling Algorithms	
L18	5.4	Demand scheduling	
	5.5	Real-Time Scheduling	
	<b>PROCESS SYNCHRONIZATION</b>	[T1][T2][R3]	

L19, L20,	6.1	Mutual exclusion, software solution to Mutual exclusion problem,	
	6.2	The Critical-Section Problem	
	6.3	Synchronization Hardware	
	6.4	Semaphores	
	6.5	Case study on Dining philosopher problem, Barber shop problem etc.	
L21- L25	<b>DEADLOCKS</b>		[T1][T2][R1]
	7.1	Examples of deadlock, resource concepts, necessary conditions for deadlock	
	7.2	Deadlock solution, deadlock prevention, deadlock avoidance with Bankers algorithms	
	7.3	Deadlock detection, deadlock recovery	
L26- L30	<b>DEVICE MANAGEMENT</b>		[T1][T2][R1]
	8.1	Disk Scheduling Strategies	
	8.2	Rotational Optimization,	
	8.3	System Consideration,	
	8.4	Caching and Buffering	
L31- L40	<b>FILE - SYSTEM INTERFACE</b>		[T1] [T2]
	9.1	File System: Introduction, File Organization,	[R4][R5]
	9.2	Logical File System, Physical File System	
	9.3	File Allocation strategy	
	9.4	Free Space Management	
	9.5	File Access Control, Data Access Techniques	
	9.6	Data Integrity Protection	
	9.7	Case study on file system viz FAT32, NTFS, Ext2/Ext3 etc.	
<b>Text Books:</b>			
		[T1] Deitel & Dietel, "Operating System", Pearson, 3rd Ed., 2011	
		[T2] Silberschatz and Galvin, "Operating System Concepts", Pearson, 5th Ed., 2001	
		[T3] Madnick & Donovan, "Operating System", TMH, 1st Ed., 2001	
<b>Reference Books:</b>			
		[R1] Tannenbaum, "Operating Systems", PHI, 4th Edition, 2000	
		[R2] Godbole, "Operating Systems", Tata McGraw Hill, 3rd edition, 2014	
		[R3] Chauhan, "Principles of Operating Systems", Oxford Uni. Press, 2014	
		[R4] Dhamdhare, "Operating Systems", Tata McGraw Hill, 3rd edition, 2012	
		[R5] Loomis, "Data Management & File Structure", PHI, 2nd Ed.	

<b>Operating Systems</b>	<b>L</b>	<b>P</b>	<b>C</b>
	<b>4</b>		<b>4</b>

Discipline(s) / EAE / OAE	Semester	Group	Sub-group	Paper Code
CSE/IT/CST/ITE	5	PC	PC	CIC-305
OAE	7	CSE-OAE	CSE-OAE-4	OCSE-409

**Marking Scheme:**

1. Teachers Continuous Evaluation: 25 marks
2. Term end Theory Examinations: 75 marks

**Instructions for paper setter:**

1. There should be 9 questions in the term end examinations question paper.
2. The first (1st) question should be compulsory and cover the entire syllabus. This question should be objective, single line answers or short answer type question of total 15 marks.
3. Apart from question 1 which is compulsory, rest of the paper shall consist of 4 units as per the syllabus. Every unit shall have two questions covering the corresponding unit of the syllabus. However, the student shall be asked to attempt only one of the two questions in the unit. Individual questions may contain upto 5 sub-parts / sub-questions. Each Unit shall have a marks weightage of 15.
4. The questions are to be framed keeping in view the learning outcomes of the course / paper. The standard / level of the questions to be asked should be at the level of the prescribed textbook.
5. The requirement of (scientific) calculators / log-tables / data – tables may be specified if required.

**Course Objectives :**

1.	To understand the basics of OS and their functions. To learn the scheduling policies of various operating systems.
2.	Learn memory management methods.
3.	To understand the characterisation of deadlock, system deadlock, preventing deadlock, avoiding deadlock and related concepts.
4.	To understand the meaning of a file, structure of the directories, file structure system and implementation, free-space management

**Course Outcomes (CO)**

<b>CO 1</b>	Understand the role of operating system in a computing device, and Ability to understand paging and segmentation methods of memory binding and their pros & cons.
<b>CO 2</b>	Understand scheduling of process over a processor. Ability to use concepts of semaphore and its usage in process synchronization.
<b>CO 3</b>	Ability to synchronize programs and make the system deadlock free.
<b>CO 4</b>	Ability to understand file system like file access methods, directory structures, file space allocation in disk and free space management in disk. Ability to understand disk scheduling and disk recovery procedures.

**Course Outcomes (CO) to Programme Outcomes (PO) mapping (scale 1: low, 2: Medium, 3: High)**

	PO01	PO02	PO03	PO04	PO05	PO06	PO07	PO08	PO09	PO10	PO11	PO12
<b>CO 1</b>	3	3	2	-	3	-	-	-	-	-	-	-
<b>CO 2</b>	3	3	-	-	2	-	-	-	-	-	-	-
<b>CO 3</b>	3	2	3	-	2	-	-	-	-	-	-	-
<b>CO 4</b>	3	3	-	-	2	-	-	-	-	-	-	-

**UNIT-I**

Introduction: What is an Operating System, Simple Batch Systems, Multiprogrammed Batches systems, Time Sharing Systems, Personal-computer systems, Parallel systems, Distributed Systems, Real-Time Systems, OS – A Resource Manager.

Processes: Introduction, Process states, process management, Interrupts, Interprocess Communication

Threads: Introduction, Thread states, Thread Operation, Threading Models. Processor Scheduling: Scheduling levels, preemptive vs no preemptive scheduling, priorities, scheduling objective, scheduling criteria, scheduling algorithms, demand scheduling, real time scheduling.

#### **UNIT-II**

Process Synchronization: Mutual exclusion, software solution to Mutual exclusion problem, hardware solution to Mutual exclusion problem, semaphores, Critical section problems. Case study on Dining philosopher problem, Barber shop problem etc.

Memory Organization & Management: Memory Organization, Memory Hierarchy, Memory Management Strategies, Contiguous versus non- Contiguous memory allocation, Partition Management Techniques, Logical versus Physical Address space, swapping, Paging, Segmentation, Segmentation with Paging Virtual Memory: Demand Paging, Page Replacement, Page-replacement Algorithms, Performance of Demand Paging, Thrashing, Demand Segmentation, and Overlay Concepts.

#### **UNIT-III**

Deadlocks: examples of deadlock, resource concepts, necessary conditions for deadlock, deadlock solution, deadlock prevention, deadlock avoidance with Bankers algorithms, deadlock detection, deadlock recovery.

Device Management: Disk Scheduling Strategies, Rotational Optimization, System Consideration, Caching and Buffering.

#### **UNIT - IV**

File System: Introduction, File Organization, Logical File System, Physical File System, File Allocation strategy, Free Space Management, File Access Control, Data Access Techniques, Data Integrity Protection, Case study on file system viz FAT32, NTFS, Ext2/Ext3 etc.

#### **Textbook(s):**

1. Deitel & Dietel, "Operating System", Pearson, 3 rd Ed., 2011
2. Silberschatz and Galvin, "Operating System Concepts", Pearson, 5th Ed., 2001
3. Madnick & Donovan, "Operating System", TMH, 1st Ed., 2001

#### **References:**

1. Tannenbaum, "Operating Systems", PHI, 4th Edition, 2000
2. Godbole, "Operating Systems", Tata McGraw Hill, 3rd edition, 2014
3. Chauhan, "Principles of Operating Systems", Oxford Uni. Press, 2014
4. Dhamdhare, "Operating Systems", Tata McGraw Hill, 3rd edition, 2012
5. Loomis, "Data Management & File Structure", PHI, 2nd Ed.

<b>Operating Systems Lab</b>	<b>L</b>	<b>P</b>	<b>C</b>
		<b>2</b>	<b>1</b>

Discipline(s) / EAE / OAE	Semester	Group	Sub-group	Paper Code
CSE/IT/CST/ITE	5	PC	PC	CIC-353

**Marking Scheme:**

1. Teachers Continuous Evaluation: 40 marks
2. Term end Theory Examinations: 60 marks

**Instructions:**

1. The course objectives and course outcomes are identical to that of (Operating Systems) as this is the practical component of the corresponding theory paper.
2. The practical list shall be notified by the teacher in the first week of the class commencement under intimation to the office of the Head of Department / Institution in which the paper is being offered from the list of practicals below. Atleast 10 experiments must be performed by the students, they may be asked to do more. Atleast 5 experiments must be from the given list.

1. Write a program to implement CPU scheduling for first come first serve.
2. Write a program to implement CPU scheduling for shortest job first.
3. Write a program to perform priority scheduling.
4. Write a program to implement CPU scheduling for Round Robin.
5. Write a program for page replacement policy using a) LRU b) FIFO c) Optimal.
6. Write a program to implement first fit, best fit and worst fit algorithm for memory management.
7. Write a program to implement reader/writer problem using semaphore.
8. Write a program to implement Producer-Consumer problem using semaphores.
9. Write a program to implement Banker's algorithm for deadlock avoidance.
10. Write C programs to implement the various File Organization Techniques

## PRACTICAL 1

**AIM:** Write a program to implement CPU scheduling for First Come First serve.

**PROGRAM:**

```
#include<stdio.h>
int main()
{
intn,bt[20],wt[20],tat[20],avwt=0,avtat=0,i,j;
printf("Enter total number of processes(maximum 20):");
scanf("%d",&n);
printf("\nEnter Process Burst Time\n");
for(i=0;i<n;i++)
{
printf("P[%d]:",i+1);
scanf("%d",&bt[i]);
}
wt[0]=0;
for(i=1;i<n;i++)
{
wt[i]=0;
for(j=0;j<i;j++)
wt[i]+=bt[j];
}
printf("\nProcess\t\tBurst Time\tWaiting Time\tTurnaround Time");
for(i=0;i<n;i++)
{
tat[i]=bt[i]+wt[i];
avwt+=wt[i];
avtat+=tat[i];
printf("\nP[%d]\t\t%d\t\t%d\t\t%d",i+1,bt[i],wt[i],tat[i]);
}
avwt/=i;
avtat/=i;
printf("\n\nAverage Waiting Time:%d",avwt);
return 0;
}
```

## OUTPUT

```
[student@localhost ~]$ gcc -o f fcfs.c
fcfs.c: In function 'main':
fcfs.c:28: warning: 'return' with a value, in function returning void
[student@localhost ~]$ ./f
Enter the no. of processes: 4
Enter the burst time of process 1: 1
Enter the burst time of process 2: 2
Enter the burst time of process 3: 3
Enter the burst time of process 4: 4

Avg, waiting time: 2[student@localhost ~]$ ./f
Enter the no. of processes: 3
Enter the burst time of process 1: 24
Enter the burst time of process 2: 3
Enter the burst time of process 3: 3

Avg. waiting time: 17[student@localhost ~]$ █
```



## PRACTICAL 2

**AIM:** Write a program to implement CPU scheduling for Shortest Job First.

### **PROGRAM:**

```
#include<stdio.h>
void main()
{
intbt[20],p[20],wt[20],tat[20],i,j,n,total=0,pos,temp;
floatavg_wt,avg_tat;
printf("Enter number of process:");
scanf("%d",&n);

printf("\nEnter Burst Time:\n");
for(i=0;i<n;i++)
{
printf("p%d:",i+1);
scanf("%d",&bt[i]);
p[i]=i+1;    //contains process number
}

//sorting burst time in ascending order using selection sort
for(i=0;i<n;i++)
{
pos=i;
for(j=i+1;j<n;j++)
{
if(bt[j]<bt[pos])
pos=j;
}

temp=bt[i];
bt[i]=bt[pos];
bt[pos]=temp;

temp=p[i];
p[i]=p[pos];
p[pos]=temp;
}

wt[0]=0;    //waiting time for first process will be zero

//calculate waiting time
for(i=1;i<n;i++)
{
wt[i]=0;
for(j=0;j<i;j++)
wt[i]+=bt[j];

total+=wt[i];
}
```

```

avg_wt=(float)total/n; //average waiting time
total=0;

printf("\nProcess\t Burst Time \tWaiting Time\tTurnaround Time");
for(i=0;i<n;i++)
{
tat[i]=bt[i]+wt[i]; //calculate turnaround time
total+=tat[i];
printf("\np%d\tt %d\t\t %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);
}

avg_tat=(float)total/n; //average turnaround time
printf("\nAverage Waiting Time=%f",avg_wt);
printf("\nAverage Turnaround Time=%f\n",avg_tat);
}

```

## OUTPUT

The screenshot shows a terminal window with the following output:

```

[student@localhost ~]$ gedit dd2.c
[student@localhost ~]$ gcc -o h dd2.c
[student@localhost ~]$ ./h
Enter number of process:3

Enter Burst Time:
p1:24
p2:3
p3:3

Process      Burst Time      Waiting Time      Turnaround Time
p2           3                0                 3
p3           3                3                 6
p1          24                6                30

Average Waiting Time=3.000000
Average Turnaround Time=13.000000
[student@localhost ~]$ █

```

### PRACTICAL 3

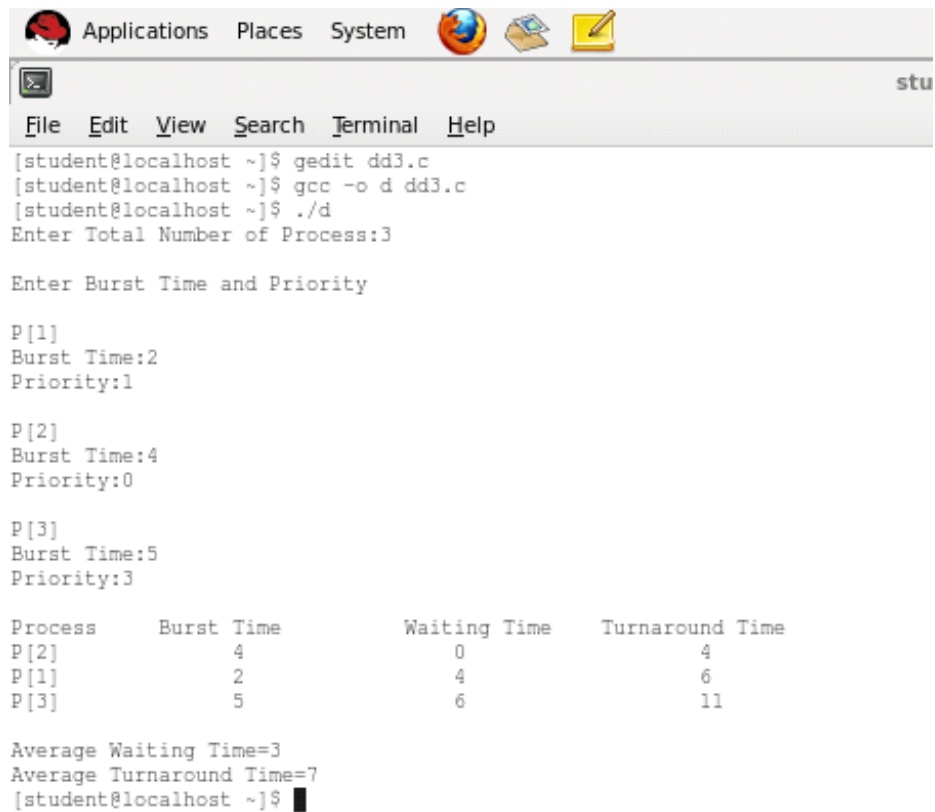
**AIM:** Write a program to implement CPU scheduling for Priority Scheduling.

**PROGRAM:**

```
#include<stdio.h>
int main()
{ int bt[20],p[20],wt[20],tat[20],pr[20],i,j,n,total=0,pos,temp,avg_wt,avg_tat;
printf("Enter Total Number of Process:");
scanf("%d",&n);
printf("\nEnter Burst Time and Priority\n");
for(i=0;i<n;i++)
{ printf("\nP[%d]\n",i+1);
printf("Burst Time:");
scanf("%d",&bt[i]);
printf("Priority:");
scanf("%d",&pr[i]);
p[i]=i+1;
}
for(i=0;i<n;i++)
{ pos=i;
for(j=i+1;j<n;j++)
{ if(pr[j]<pr[pos])
pos=j; }
temp=pr[i];
pr[i]=pr[pos];
pr[pos]=temp;
temp=bt[i];
bt[i]=bt[pos];
bt[pos]=temp;
temp=p[i];
p[i]=p[pos];
p[pos]=temp; }
wt[0]=0;
for(i=1;i<n;i++)
{ wt[i]=0;
for(j=0;j<i;j++)
wt[i]+=bt[j];
total+=wt[i]; }
avg_wt=total/n;
total=0;
printf("\nProcess\t Burst Time \tWaiting Time\tTurnaround Time");
for(i=0;i<n;i++)
{ tat[i]=bt[i]+wt[i];
total+=tat[i];
printf("\nP[%d]\t\t %d\t\t %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);
```

```
}
avg_tat=total/n;
printf("\n\nAverage Waiting Time=%d",avg_wt);
printf("\n\nAverage Turnaround Time=%d\n",avg_tat);
return 0;
```

## OUTPUT



```
[student@localhost ~]$ gedit dd3.c
[student@localhost ~]$ gcc -o d dd3.c
[student@localhost ~]$ ./d
Enter Total Number of Process:3

Enter Burst Time and Priority

P[1]
Burst Time:2
Priority:1

P[2]
Burst Time:4
Priority:0

P[3]
Burst Time:5
Priority:3

Process      Burst Time      Waiting Time      Turnaround Time
P[2]          4                0                  4
P[1]          2                4                  6
P[3]          5                6                 11

Average Waiting Time=3
Average Turnaround Time=7
[student@localhost ~]$ █
```

## PRACTICAL 4

**AIM:** Write a program to implement CPU scheduling for Round Robin.

**PROGRAM:**

```
#include<stdio.h>
int main()
{   intcount,j,n,time,remain,flag=0,time_quantum;
intwait_time=0,turnaround_time=0,at[10],bt[10],rt[10];
printf("Enter Total Process:\t ");
scanf("%d",&n);
remain=n;
for(count=0;count<n;count++)
{
printf("Enter Arrival Time and Burst Time for Process Process Number %d :",count+1);
scanf("%d",&at[count]);
scanf("%d",&bt[count]);
rt[count]=bt[count];
}
printf("Enter Time Quantum:\t");
scanf("%d",&time_quantum);
printf("\n\nProcess\t|TurnaroundTime|Waiting Time\n\n");
for(time=0,count=0;remain!=0;)
{
if(rt[count]<=time_quantum&&rt[count]>0)
{   time+=rt[count];
rt[count]=0;
flag=1;  }
else if(rt[count]>0)
{   rt[count]-=time_quantum;
time+=time_quantum;  }
if(rt[count]==0 && flag==1)
{   remain--;
printf("P[%d]\t|\t%d\t|\t%d\n",count+1,time-at[count],time-at[count]-bt[count]);
wait_time+=time-at[count]-bt[count];
turnaround_time+=time-at[count];
flag=0;  }
}
```

```

if(count==n-1)    count=0;  else if(at[count+1]<=time)    count++;  else    count=0;
}
printf("\nAverage Waiting Time= %f\n",wait_time*1.0/n);
printf("Avg Turnaround Time = %f",turnaround_time*1.0/n);
return 0;

```

## OUTPUT

```

[student@localhost ~]$ gedit aal.c
[student@localhost ~]$ gcc -o a aal.c
gcc: a: No such file or directory
gcc: unrecognized option '-o'
aal.c: In function 'main':
aal.c:30: error: expected declaration or statement at end of input
[student@localhost ~]$ gcc -o a aal.c
[student@localhost ~]$ ./a
Enter Total Process:      3
Enter Arrival Time and Burst Time for Process Process Number 1 :2
24
Enter Arrival Time and Burst Time for Process Process Number 2 :3
3
Enter Arrival Time and Burst Time for Process Process Number 3 :1
3
Enter Time Quantum:      2

Process |Turnaround Time|Waiting Time
P[2]    |      8      |      5
P[3]    |     11      |      8
P[1]    |     28      |      4

Average Waiting Time= 5.666667
[student@localhost ~]$ █

```

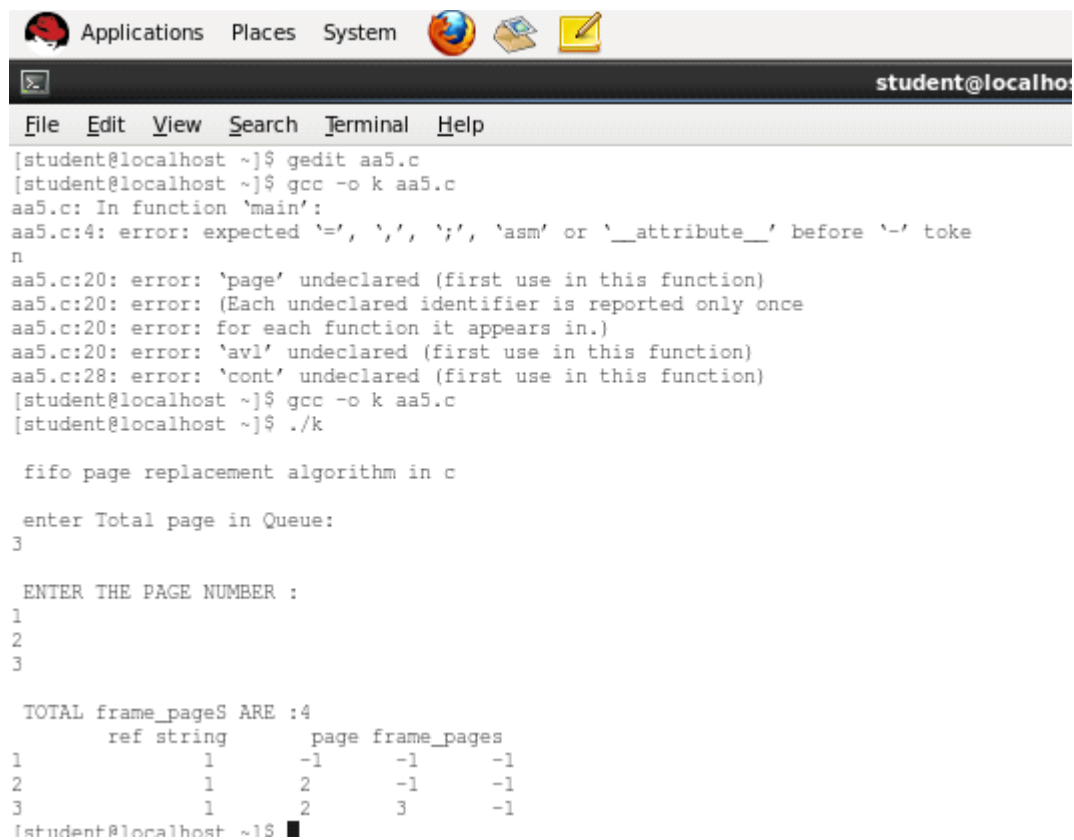
## PRACTICAL 5

**AIM:** Write a program for Page Replacement using FIFO.

### **PROGRAM:**

```
#include<stdio.h>
int main()
{
    inti,j,n,a[50],frame_page[6],no,k,page-avl,cont=0;
    printf("\n fifo page replacement algorithm in c \n")
    printf("\n enter Total page in Queue:\n");
    scanf("%d",&n);
    printf("\n ENTER THE PAGE NUMBER :\n");
    for(i=1;i<=n;i++)
    scanf("%d",&a[i]);
    printf("\n TOTAL frame_pageS ARE :");
    scanf("%d",&no);
    for(i=0;i<no;i++)
    frame_page[i]= -1;
    j=0;
    printf("\tref string\t page frame_pages\n");
    for(i=1;i<=n;i++)
    {
        printf("%d\t\t",a[i]);
        page-avl=0;
        for(k=0;k<no;k++)
        if(frame_page[k]==a[i])
        page-avl=1;
        if (page-avl==0)
        {
            frame_page[j]=a[i];
            j=(j+1)%no;
            cont++;
            for(k=0;k<no;k++)
            printf("%d\t",frame_page[k]);
        }
        printf("\n");
    }
    printf("Page Fault Is %d",cont);
    return 0;
}
```

## OUTPUT



```
[student@localhost ~]$ gedit aa5.c
[student@localhost ~]$ gcc -o k aa5.c
aa5.c: In function 'main':
aa5.c:4: error: expected '=', ',', ';', 'asm' or '__attribute__' before '-' token
aa5.c:20: error: 'page' undeclared (first use in this function)
aa5.c:20: error: (Each undeclared identifier is reported only once
aa5.c:20: error: for each function it appears in.)
aa5.c:20: error: 'avl' undeclared (first use in this function)
aa5.c:28: error: 'cont' undeclared (first use in this function)
[student@localhost ~]$ gcc -o k aa5.c
[student@localhost ~]$ ./k

fifo page replacement algorithm in c

enter Total page in Queue:
3

ENTER THE PAGE NUMBER :
1
2
3

TOTAL frame_pageS ARE :4
   ref string      page frame_pages
1         1        -1      -1      -1
2         1         2      -1      -1
3         1         2       3      -1
[student@localhost ~]$ █
```



## PRACTICAL 6

**AIM:** Write a program to implement first fit, best fit, worst fit for memory management.

### **PROGRAM:**

```
#include<stdio.h>
#include<conio.h>
void main()
{inti,j,temp,b[10],c[10],arr,n,ch,a;
clrscr();
printf("\t\tFIRST FIT, BEST FIT, WORST FIT\n");
printf("Enter the size of no. of blocks:");

scanf("%d",&n);
for(i=1;i<=n;i++)
{printf("Enter the size of %d block:",i);
scanf("%d",&b[i]);c[i]=b[i];}
printf("\nEnter the size of Arriving block:");
scanf("%d",&arr);
printf("\n1.First fit\n2.Best fit\n3.Worst fit\nEnter your choice:");
scanf("%d",&ch);
switch(ch)
{case 1:      for(i=1;i<=n;i++)
              {      if(b[i]>=arr)
                  {      printf("Arriving block is allocated to %d block.",i);
                      break;
                  }      else
                      continue;
              }      break;
case 2:      for(i=1;i<=n;i++)
              {      for(j=1;j<n-i;j++)
                  {      if(b[i]>=b[i+1])
                      {      temp=b[i];
                          b[i]=b[i+1];
                          b[i+1]=temp;          }
                  }      }
              for(i=1;i<=n;i++)
                  {      if(b[i]>=arr)
                      {
                          a=b[i];
                          break;          }
                  else
```

```

continue;    }
for(i=1;i<=n;i++)
    {        if(c[i]==a)
        {        printf("Arriving block is allocated to %d block.",i);
        }    }
break;
case 3:
for(i=1;i<=n;i++)
    {
for(j=1;j<n;j++)
    {
if(b[i]>=b[i+1])
    {
temp=b[i];
b[i]=b[i+1];
b[i+1]=temp;
    }
    }
}
for(i=1;i<=n;i++)
printf(" %d",b[i]);
break;
default:    printf("Enter the valid choice:");
    }
getch());

```

## OUTPUT

1)First Fit:

```

student@localhost ~$ gcc a42.c
student@localhost ~$ ./a42.c
Enter the size of 3 blocks:
1000
2000
3000
Enter the size of arriving blocks:
1
2
3
Arriving block is allocated to 1 block.(student@localhost ~)$

```

## 2)Best Fit:

```
Arriving block is allocated to 1 block.[student@localhost ~]$ ./s
FIRST FIT, BEST FIT, WORST FIT
Enter the size of no. of blocks:3
Enter the size of 1 block:100
Enter the size of 2 block:150
Enter the size of 3 block:200

Enter the size of Arriving block:2

1.First fit
2.Best fit
3.Worst fit
Enter your choice:2
Arriving block is allocated to 1 block.[student@localhost ~]$ █
```

## 3) Worst Fit:

```
FIRST FIT, BEST FIT, WORST FIT
Enter the size of no. of blocks:3
Enter the size of 1 block:100
Enter the size of 2 block:150
Enter the size of 3 block:200

Enter the size of Arriving block:2

1.First fit
2.Best fit
3.Worst fit
Enter your choice:3
100 150 -1075546520[student@localhost ~]$ █
```

## PRACTICAL 7

**AIM:** Write a program to implement reader/writer problem using semaphore.

```
#include<stdio.h>
#include<pthread.h>
#include<semaphore.h>

sem_t mutex,writeblock;
int data = 0,rcount = 0;

void *reader(void *arg)
{
    int f;
    f = ((int)arg);
    sem_wait(&mutex);
    rcount = rcount + 1;
    if(rcount==1)
        sem_wait(&writeblock);
    sem_post(&mutex);
    printf("Data read by the reader%d is %d\n",f,data);
    sleep(1);
    sem_wait(&mutex);
    rcount = rcount - 1;
    if(rcount==0)
        sem_post(&writeblock);
    sem_post(&mutex);
}

void *writer(void *arg)
{
    int f;
    f = ((int) arg);
    sem_wait(&writeblock);
    data++;
    printf("Data written by the writer%d is %d\n",f,data);
    sleep(1);
    sem_post(&writeblock);
}

int main()
{
    int i,b;
    pthread_t rtid[5],wtid[5];
    sem_init(&mutex,0,1);
    sem_init(&writeblock,0,1);
    for(i=0;i<=2;i++)
    {
```

```
pthread_create(&wtid[i],NULL,writer,(void *)i);
pthread_create(&rtid[i],NULL,reader,(void *)i);
}
for(i=0;i<=2;i++)
{
pthread_join(wtid[i],NULL);
pthread_join(rtid[i],NULL);
}
return 0;
}
```

## OUTPUT

```
File Edit View Terminal Help
student@cc-02-desktop:~$ cd 004
student@cc-02-desktop:~/004$ cc readerw.c -lpthread -o readerw
student@cc-02-desktop:~/004$ ./readerw

value of shared variable is 0
W :written =0
W going to sleep..
R :read=0 Reader no.=1
Reader1:going to sleep...
R :read=0 Reader no.=2
Reader2:going to sleep...
R :read=0 Reader no.=1
Reader1:going to sleep...
R :read=0 Reader no.=2
Reader2:going to sleep...
R :read=0 Reader no.=1
Reader1:going to sleep...
R :read=0 Reader no.=2
Reader2:going to sleep...
R :read=0 Reader no.=1
Reader1:going to sleep...
R :read=0 Reader no.=2
Reader2:going to sleep...
R :read=0 Reader no.=1
Reader1:going to sleep...
R :read=0 Reader no.=2
Reader2:going to sleep...Reader no.=1 livingReader no.=2 living
W :written =2
W going to sleep..
W :written =4
W going to sleep..Writer is livingstudent@cc-02-desktop:~/004$
```

## PRACTICAL 8

**AIM:** Write a program to implement producer consumer problem using semaphore.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
#include <unistd.h>
#include <pthread.h>
#include <semaphore.h>

/*
use the pthread flag with gcc to compile this code
~$ gcc -pthread producer_consumer.c -o producer_consumer
*/

pthread_t *producers;
pthread_t *consumers;

sem_t buf_mutex,empty_count,fill_count;

int *buf,buf_pos=-1,prod_count,con_count,buf_len;

int produce(pthread_t self){
    int i = 0;
    int p = 1 + rand()%40;
    while(!pthread_equal(*(producers+i),self) && i < prod_count){
        i++;
    }
    printf("Producer %d produced %d \n",i+1,p);
    return p;
}

void consume(int p,pthread_t self){
    int i = 0;
    while(!pthread_equal(*(consumers+i),self) && i < con_count){
        i++;
    }

    printf("Buffer:");
    for(i=0;i<=buf_pos;++i)
        printf("%d ",*(buf+i));
    printf("\nConsumer %d consumed %d \nCurrent buffer len: %d\n",i+1,p,buf_pos);
}

}
```

```

void* producer(void *args){

    while(1){
        int p = produce(pthread_self());
        sem_wait(&empty_count);
        sem_wait(&buf_mutex);
        ++buf_pos;
        *(buf + buf_pos) = p;
        sem_post(&buf_mutex);
        sem_post(&fill_count);
        sleep(1 + rand()%3);
    }

    return NULL;
}

```

```

void* consumer(void *args){
    int c;
    while(1){
        sem_wait(&fill_count);
        sem_wait(&buf_mutex);
        c = *(buf+buf_pos);
        consume(c,pthread_self());
        --buf_pos;
        sem_post(&buf_mutex);
        sem_post(&empty_count);
        sleep(1+rand()%5);
    }

    return NULL;
}

```

```

int main(void){

    int i,err;

    srand(time(NULL));

    sem_init(&buf_mutex,0,1);
    sem_init(&fill_count,0,0);

    printf("Enter the number of Producers:");
    scanf("%d",&prod_count);
    producers = (pthread_t*) malloc(prod_count*sizeof(pthread_t));

    printf("Enter the number of Consumers:");
    scanf("%d",&con_count);
    consumers = (pthread_t*) malloc(con_count*sizeof(pthread_t));

    printf("Enter buffer capacity:");
    scanf("%d",&buf_len);
    buf = (int*) malloc(buf_len*sizeof(int));
}

```



```
sem_init(&empty_count,0,buf_len);

for(i=0;i<prod_count;i++){
    err = pthread_create(producers+i,NULL,&producer,NULL);
    if(err != 0){
        printf("Error creating producer %d: %s\n",i+1,strerror(err));
    }else{
        printf("Successfully created producer %d\n",i+1);
    }
}

for(i=0;i<con_count;i++){
    err = pthread_create(consumers+i,NULL,&consumer,NULL);
    if(err != 0){
        printf("Error creating consumer %d: %s\n",i+1,strerror(err));
    }else{
        printf("Successfully created consumer %d\n",i+1);
    }
}

for(i=0;i<prod_count;i++){
    pthread_join(*(producers+i),NULL);
}
for(i=0;i<con_count;i++){
    pthread_join(*(consumers+i),NULL);
}

return 0;
}
```

## PRACTICAL 9

**AIM:** Write a program to implement Banker's for deadlock avoidance.

**PROGRAM:**

```
#include <stdio.h>
#include <stdlib.h>
int main()
{ int Max[10][10], need[10][10], alloc[10][10], avail[10], completed[10], safeSequence[10];
int p, r, i, j, process, count;
count = 0;
printf("Enter the no of processes : ");
scanf("%d", &p);
for(i = 0; i < p; i++)
completed[i] = 0;
printf("\n\nEnter the no of resources : ");
scanf("%d", &r);
printf("\n\nEnter the Max Matrix for each process : ");
for(i = 0; i < p; i++)
{ printf("\nFor process %d : ", i + 1);
for(j = 0; j < r; j++)
scanf("%d", &Max[i][j]); }
printf("\n\nEnter the allocation for each process : ");
for(i = 0; i < p; i++)
{ printf("\nFor process %d : ", i + 1);
for(j = 0; j < r; j++)
scanf("%d", &alloc[i][j]); }
printf("\n\nEnter the Available Resources : ");
for(i = 0; i < r; i++)
scanf("%d", &avail[i]);
for(i = 0; i < p; i++)
for(j = 0; j < r; j++)
need[i][j] = Max[i][j] - alloc[i][j];
```

```

do
    {      printf("\n Max matrix:\tAllocation matrix:\n");
for(i = 0; i < p; i++)
    {      for( j = 0; j < r; j++)
printf("%d ", Max[i][j]);
printf("\t\t");
for( j = 0; j < r; j++)
printf("%d ", alloc[i][j]);
printf("\n");  }
process = -1;
for(i = 0; i < p; i++)
{ if(completed[i] == 0)//if not completed
    { process = i ;
for(j = 0; j < r; j++)
{ if(avail[j] < need[i][j])
    { process = -1;
      Break;  }}}
if(process != -1)
break;  }
if(process != -1)
    { printf("\nProcess %d runs to completion!", process + 1);
safeSequence[count] = process + 1;
count++;
for(j = 0; j < r; j++)
{ avail[j] += alloc[process][j];
alloc[process][j] = 0;
Max[process][j] = 0;
completed[process] = 1;  }}}
while(count != p && process != -1);
if(count == p)
{ printf("\nThe system is in a safe state!!\n");
printf("Safe Sequence : < ");
for( i = 0; i < p; i++)
printf("%d ", safeSequence[i]);

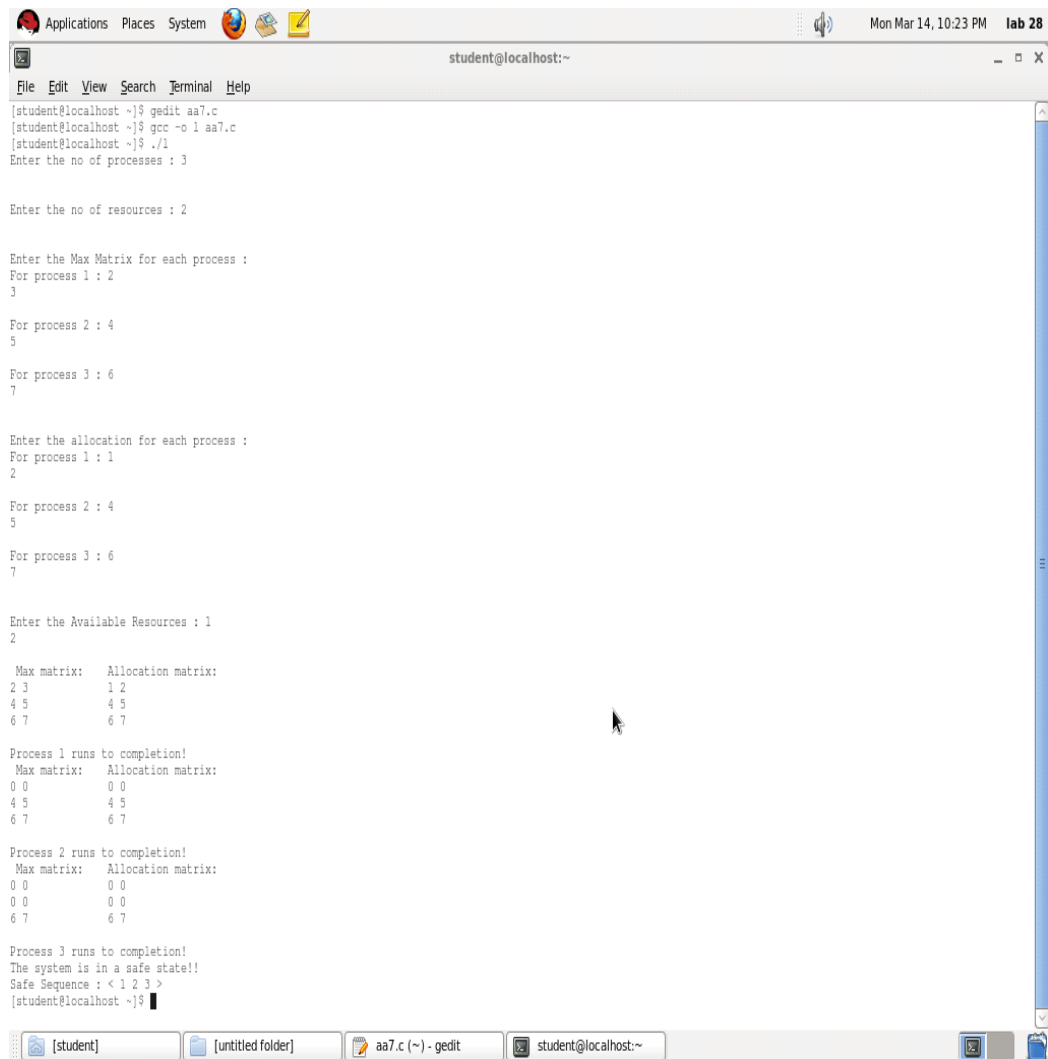
```

```
printf(">\n"); }
```

```
else
```

```
printf("\nThe system is in an unsafe state); }
```

## OUTPUT



```
Applications Places System student@localhost:~
student@localhost:~
File Edit View Search Terminal Help
[student@localhost ~]$ gedit aa7.c
[student@localhost ~]$ gcc -o 1 aa7.c
[student@localhost ~]$ ./1
Enter the no of processes : 3

Enter the no of resources : 2

Enter the Max Matrix for each process :
For process 1 : 2
3
For process 2 : 4
5
For process 3 : 6
7

Enter the allocation for each process :
For process 1 : 1
2
For process 2 : 4
5
For process 3 : 6
7

Enter the Available Resources : 1
2

Max matrix:   Allocation matrix:
2 3           1 2
4 5           4 5
6 7           6 7

Process 1 runs to completion!
Max matrix:   Allocation matrix:
0 0           0 0
4 5           4 5
6 7           6 7

Process 2 runs to completion!
Max matrix:   Allocation matrix:
0 0           0 0
0 0           0 0
6 7           6 7

Process 3 runs to completion!
The system is in a safe state!!
Safe Sequence : < 1 2 3 >
[student@localhost ~]$
```

## PRACTICAL 10

**AIM:** Write a program to implement various file organisation techniques.

The various file organisation techniques are:

a) Single level directory b) Two level directory c) Hierarchical

### **DESCRIPTION**

The directory structure is the organization of files into a hierarchy of folders. In a single-level directory system, all the files are placed in one directory. There is a root directory which has all files. It has a simple architecture and there are no sub directories. Advantage of single level directory system is that it is easy to find a file in the directory. In the two-level directory system, each user has own user file directory (UFD). The system maintains a master block that has one entry for each user. This master block contains the addresses of the directory of the users. When a user job starts or a user logs in, the system's master file directory (MFD) is searched. When a user refers to a particular file, only his own UFD is searched. This effectively solves the name collision problem and isolates users from one another. Hierarchical directory structure allows users to create their own subdirectories and to organize their files accordingly. A tree is the most common directory structure. The tree has a root directory, and every file in the system has a unique path name. A directory (or subdirectory) contains a set of files or subdirectories.

### **PROGRAM**

#### SINGLE LEVEL DIRECTORY ORGANIZATION

```
#include<stdio.h>
struct
{
char dname[10],fname[10][10];
int fcnt;
}dir;
void main()
{
int i,ch;
char f[30];
clrscr();
dir.fcnt = 0;
printf("\nEnter name of directory -- ");
scanf("%s", dir.dname);
while(1)
{
printf("\n\n 1. Create File\t2. Delete File\t3. Search File \n 4. Display Files\t5. Exit\nEnter your choice -- ");
scanf("%d",&ch);
switch(ch)
{
case 1: printf("\n Enter the name of the file -- ");
scanf("%s",dir.fname[dir.fcnt]);
dir.fcnt++;
break;
case 2: printf("\n Enter the name of the file -- ");
scanf("%s",f);
for(i=0;i<dir.fcnt;i++)
```

```

{
if(strcmp(f, dir.fname[i])==0)
{
printf("File %s is deleted ",f);
strcpy(dir.fname[i],dir.fname[dir.fcnt-1]);
break;
}
}
if(i==dir.fcnt)
printf("File %s not found",f);
else
dir.fcnt--;
break;
case 3: printf("\n Enter the name of the file -- ");
scanf("%s",f);
for(i=0;i<dir.fcnt;i++)
{
if(strcmp(f, dir.fname[i])==0)
{
printf("File %s is found ", f);
break;
}
}
if(i==dir.fcnt)
printf("File %s not found",f);
break;
case 4: if(dir.fcnt==0)
printf("\n Directory Empty");
else
{
printf("\n The Files are -- ");
for(i=0;i<dir.fcnt;i++)
printf("\t%s",dir.fname[i]);
}
break;
default: exit(0);
}
}
getch();
}

```

OUTPUT:

Enter name of directory -- CSE

1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit Enter your choice – 1

Enter the name of the file -- A

1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit Enter your choice – 1

Enter the name of the file -- B

1. Create File 2. Delete File 3. Search File

4. Display Files 5. Exit Enter your choice – 1

Enter the name of the file -- C

1. Create File 2. Delete File 3. Search File  
4. Display Files 5. Exit Enter your choice – 4

The Files are -- A B C

1. Create File 2. Delete File 3. Search File  
4. Display Files 5. Exit Enter your choice – 3

Enter the name of the file – ABC

File ABC not found

1. Create File 2. Delete File 3. Search File  
4. Display Files 5. Exit Enter your choice – 2

Enter the name of the file – B

File B is deleted

1. Create File 2. Delete File 3. Search File  
4. Display Files 5. Exit Enter your choice – 5

## TWO LEVEL DIRECTORY ORGANIZATION

```
#include<stdio.h>
struct
{
char dname[10],fname[10][10];
int fcnt;
}dir[10];
void main()
{
int i,ch,dcnt,k;
char f[30], d[30];
clrscr();
dcnt=0;
while(1)
{
printf("\n\n 1. Create Directory\t 2. Create File\t 3. Delete File");
printf("\n 4. Search File \t \t 5. Display \t 6. Exit \t Enter your choice -- ");
scanf("%d",&ch);
switch(ch)
{
case 1: printf("\n Enter name of directory -- ");
scanf("%s", dir[dcnt].dname);
dir[dcnt].fcnt=0;
dcnt++;
printf("Directory created");
break;
case 2: printf("\n Enter name of the directory -- ");
scanf("%s",d);
for(i=0;i<dcnt;i++)
if(strcmp(d,dir[i].dname)==0)
```

```

{
printf("Enter name of the file -- ");
scanf("%s",dir[i].fname[dir[i].fcnt]);
dir[i].fcnt++;
printf("File created");
break;
}
if(i==dcnt)
printf("Directory %s not found",d);
break;
case 3: printf("\nEnter name of the directory -- ");
scanf("%s",d);
for(i=0;i<dcnt;i++)
{
if(strcmp(d,dir[i].dname)==0)
{
printf("Enter name of the file -- ");
scanf("%s",f);
for(k=0;k<dir[i].fcnt;k++)
{
if(strcmp(f, dir[i].fname[k])==0)
{
printf("File %s is deleted ",f);
dir[i].fcnt--;
strcpy(dir[i].fname[k],dir[i].fname[dir[i].fcnt]);
goto jmp;
}
}
printf("File %s not found",f);
goto jmp;
}
}
printf("Directory %s not found",d);
jmp : break;
case 4: printf("\nEnter name of the directory -- ");
scanf("%s",d);
for(i=0;i<dcnt;i++)
{

if(strcmp(d,dir[i].dname)==0)
{
printf("Enter the name of the file -- ");
scanf("%s",f);
for(k=0;k<dir[i].fcnt;k++)
{
if(strcmp(f, dir[i].fname[k])==0)
{
printf("File %s is found ",f);
goto jmp1;
}
}
printf("File %s not found",f);
goto jmp1;
}
}
}

```



```

printf("Directory %s not found",d);
jmp1: break;
case 5: if(dcnt==0)
printf("\nNo Directory's ");
else
{
printf("\nDirectory\tFiles");
for(i=0;i<dcnt;i++)
{
printf("\n%s\t\t",dir[i].dname);
for(k=0;k<dir[i].fcnt;k++)
printf("\t%s",dir[i].fname[k]);
}
}
break;
default:exit(0);
}
}
getch();
}

```

OUTPUT:

1. Create Directory 2. Create File 3. Delete File  
4. Search File 5. Display 6. Exit Enter your choice -- 1

Enter name of directory -- DIR1  
Directory created

1. Create Directory 2. Create File 3. Delete File  
4. Search File 5. Display 6. Exit Enter your choice -- 1

Enter name of directory -- DIR2  
Directory created

1. Create Directory 2. Create File 3. Delete File  
4. Search File 5. Display 6. Exit Enter your choice -- 2

Enter name of the directory -- DIR1  
Enter name of the file -- A1  
File created

1. Create Directory 2. Create File 3. Delete File  
4. Search File 5. Display 6. Exit Enter your choice -- 2

Enter name of the directory -- DIR1  
Enter name of the file -- A2  
File created

1. Create Directory 2. Create File 3. Delete File  
4. Search File 5. Display 6. Exit Enter your choice -- 2

Enter name of the directory -- DIR2  
Enter name of the file -- B1

File created

1. Create Directory
  2. Create File
  3. Delete File
  4. Search File
  5. Display
  6. Exit
- Enter your choice -- 5

Directory Files

DIR1 A1 A2

DIR2 B1

1. Create Directory
  2. Create File
  3. Delete File
  4. Search File
  5. Display
  6. Exit
- Enter your choice -- 4

Enter name of the directory – DIR

Directory not found

1. Create Directory
  2. Create File
  3. Delete File
  4. Search File
  5. Display
  6. Exit
- Enter your choice -- 3

Enter name of the directory – DIR1

Enter name of the file -- A2

File A2 is deleted

1. Create Directory
  2. Create File
  3. Delete File
  4. Search File
  5. Display
  6. Exit
- Enter your choice – 6

## HIERARCHICAL DIRECTORY ORGANIZATION

```
#include<stdio.h>
#include<graphics.h>
struct tree_element
{
char name[20];
int x,y,ftype,lx,rx,nc,level;
struct tree_element *link[5];
};
typedef struct tree_element
node; void main()
{
int gd=DETECT,gm;
node *root;
root=NULL;
clrscr();
create(&root,0,"root",0,639,320);
clrscr();
initgraph(&gd,&gm,"c:\\tc\\BGI");
display(root);
getch();
closegraph();
}
create(node **root,int lev,char *dname,int lx,int rx,int x)
{
int i,gap;
if(*root==NULL)
{
(*root)=(node *)malloc(sizeof(node));
```

```

printf("Enter name of dir/file(under %s) :",dname);
fflush(stdin);
gets((*root)->name);
printf("enter 1 for Dir/2 forfile :");
scanf("%d",&(*root)->ftype);
(*root)->level=lev;
(*root)->y=50+lev*50;
(*root)->x=x;
(*root)->lx=lx;
(*root)->rx=rx;
for(i=0;i<5;i++)
(*root)->link[i]=NULL;
if((*root)->ftype==1)
{
printf("No of sub directories/files(for %s):",(*root)->name); scanf("%d",&(*root)->nc);
if((*root)->nc==0)
gap=rx-lx;
else gap=(rx-lx)/(*root)->nc;
for(i=0;i<(*root)->nc;i++)
create(&((*root)->link[i]),lev+1,(*root)->name,lx+gap*i,lx+gap*i+gap,lx+gap*i+gap/2);
}
else (*root)->nc=0;
}
}
display(node *root)
{
int i;
settextstyle(2,0,4);
settextjustify(1,1);
setfillstyle(1,BLUE);
setcolor(14); if(root!=NULL)
{
for(i=0;i<root->nc;i++)
{
line(root->x,root->y,root->link[i]->x,root->link[i]->y);
}
if(root->ftype==1) bar3d(root->x-20,root->y-10,root->x+20,root->y+10,0,0); else
fillellipse(root->x,root->y,20,20);
outtextxy(root->x,root->y,root->name); for(i=0;i<root->nc;i++)
{
display(root->link[i]);
}
}
}
}

```

#### OUTPUT:

```

Enter Name of dir/file (under root): ROOT
Enter 1 for Dir / 2 For File : 1
No of subdirectories / files (for ROOT) :2
Enter Name of dir/file (under ROOT):USER 1
Enter 1 for Dir /2 for file:1
No of subdirectories /files (for USER 1):1
Enter Name of dir/file (under USER 1):SUBDIR
Enter 1 for Dir /2 for file:1
No of subdirectories /files (for SUBDIR):2

```

Enter Name of dir/file (under USER 1):  
JAVA Enter 1 for Dir /2 for file:1  
No of subdirectories /files (for JAVA): 0  
Enter Name of dir/file (under SUBDIR):VB  
Enter 1 for Dir /2 for file:1  
No of subdirectories /files (for VB): 0

Enter Name of dir/file (under ROOT):USER2  
Enter 1 for Dir /2 for file:1  
No of subdirectories /files (for USER2):2  
Enter Name of dir/file (under ROOT):A  
Enter 1 for Dir /2 for file:2  
Enter Name of dir/file (under USER2):SUBDIR 2  
Enter 1 for Dir /2 for file:1  
No of subdirectories /files (for SUBDIR 2):2

Enter Name of dir/file (under SUBDIR2):PPL  
Enter 1 for Dir /2 for file:1  
No of subdirectories /files (for PPL):2  
Enter Name of dir/file (under PPL):B  
Enter 1 for Dir /2 for file:2  
Enter Name of dir/file (under PPL):C  
Enter 1 for Dir /2 for file:2  
Enter Name of dir/file (under SUBDIR):AI  
Enter 1 for Dir /2 for file:1  
No of subdirectories /files (for AI): 2  
Enter Name of dir/file (under AI):D  
Enter 1 for Dir /2 for file:2  
Enter Name of dir/file (under AI):E

## Question Bank

### **Ques 1. Why is the operating system important?**

OS is the heart of a system without which it cannot function and is useless. It is an essential component as it acts as a link between computer software and users. It helps communicate with the system hardware and acts as a resource manager. It provides services to a user and performs all application tasks.

### **Ques 2. State the main purpose of an OS and types of OS?**

An OS executes user programs so that users can understand and interact with computer systems easily. It also improves system performance by managing all computational activities in the system. Apart from this, it also manages processes, the operation of all hardware and software, and computer memory.

Following are the five main types of OS:

- Batch Operating System
- Time-sharing operating System
- Distributed operating System
- Network operating system
- Real-time operating system

### **Ques 3. State the benefits of a multiprocessor system?**

As the name suggests, this OS consists of multiple processors that share a common physical memory and that operate under a single OS. The system divides a task into subtasks that execute parallelly in different processors whose working is transparent to the users.

Following are the benefits of a multiprocessor system:

- It improves the performance of systems that concurrently run multiple programs.
- Increased system throughput and speed and high computing power.
- Multiple processors help complete a larger number of tasks in less time.
- It is cost-effective.
- It improves system reliability.

### **Ques 4. What is the importance of Kernel in an OS?**

Kernel helps the OS manage the operations of the computer system and hardware, basically the memory and CPU time. It uses inter-process communication and system calls to act as a bridge between applications and data processing performed at the hardware level.

It handles disk management, task management, and memory management and decides which process to allocate to the CPU for execution. It also manages communication between the software and hardware.

### **Ques 5. Explain the kinds of operations that are possible on a semaphore in detail.**

The two atomic operations possible on a semaphore are:

- **Wait():** The wait (sleep or down) operation decrements the value of an argument. No operation occurs if the argument is negative or zero.
- **Signal():** The signal (wake-up or up) operation increases the value of an argument.

**Ques 6. Define Scheduling Algorithms and name different types of scheduling algorithms.**

Scheduling algorithm schedules processes on the processor in an efficient and effective manner. A Process Scheduler performs scheduling. It maximizes CPU utilization by increasing throughput.

Following are the types of process scheduling algorithms:

1. First-Come, First-Served (FCFS) Scheduling
2. Shortest-Job-Next (SJN) Scheduling
3. Priority Scheduling
4. Shortest Remaining Time
5. Round Robin(RR) Scheduling
6. Multiple-Level Queues Scheduling
7. Multilevel Feedback Queues Scheduling
8. Highest Response Ratio Next

**Ques 7. What is demand paging and how can the user perform demand paging in a system?**

Demand paging loads pages into system memory on demand. We use this method in virtual memory. A page enters into the memory when the OS references a location on that particular page during execution.

Following are the steps to perform demand paging:

- Make an attempt to access the page.
- Continue processing instructions as normal if the page is present in the memory.
- If the page is unavailable, then a situation of page-fault trap occurs.
- Make sure that the memory reference is a valid reference to a location on secondary memory in order to page in the required page. If it is not, then the process terminates due to illegal memory access.
- We need to schedule disk operation so that the desired page can be read into the main memory.
- Now, the interrupted instruction can restart.

**Ques 8. Can you relate the microprocessor to the OS?**

The OS controls a microprocessor and an OS with a microprocessor is known as a micro-controller.

**Ques 9. How is a hard deadline different from a soft deadline?**

A hard deadline system is very strict with deadlines and does not miss a single deadline. If it misses a deadline then the system fails. Whereas a soft deadline system is more lenient and the user can have some misses. We can fix the miss number and frequency with the help of algorithms. The system fails if these conditions fail.

**Ques 10. What is process synchronization?**

Process Synchronization means coordinating the execution of processes such that no two processes access the same shared resources and data. It is required in a multi-process system where multiple processes run together, and more than one process tries to gain access to the same shared resource or data at the same time. Process synchronization has two types, namely, Independent Process and Cooperative Process.

**Ques 11. What is IPC? Name the different IPC mechanisms?**

IPC stands for Interprocess Communication. It helps exchange data between multiple threads in one (or more) processes or programs. It doesn't matter whether the process is running on single or multiple computers (connected by a network). It allows coordination of activities among various program processes running concurrently in an OS.

Following are the different IPC Mechanisms:

- Pipes
- Message Queuing
- Semaphores
- Socket
- Shared Memory
- Signals

**Ques 12. What are overlays in the OS?**

A programming method divides processes into multiple pieces in order to save the important instructions in memory. This method is known as an overlay and needs no support from the OS. Programs bigger in size than physical memory can also be run using this as it stores the important data and instructions only.

**Ques 13. Define thrashing in OS?**

In thrashing, the CPU performs more swapping or paging work as compared to productive work. When the process does not have enough pages, the page-fault rate increases, degrading or collapsing the system performance. Thrashing reduces CPU utilization and multiprogramming.

A high degree of Multiprogramming, unequal number of frames and processes requirement, and more swapping of processes when CPU utilization is low are some causes of thrashing.

**Ques 14. What do you understand by the term daemon?**

A computer program running as a background process, instead of being under the direct control of an interactive user is known as a daemon. The process names of a daemon end with 'd' so that the user can differentiate between a daemon and a normal computer program.

**Ques 15. What do you mean by a thread?**

A thread is the basic unit of the process code and is called a lightweight process within a process that cannot exist outside a process. It has its own program counter. Threads share information like code segments, open files, and data segments with each other. If there is any change in the information of one thread all the other threads can see that.

A thread keeps track of:

- instructions to execute next.
- system registers that have the current working variables of a process.
- a stack containing the execution history.

**Ques 16. Define a process and its different states.**

The program that the OS is currently executing is known as a process. It is the basic unit of work that the OS implements in the system and it takes place in a sequence. A program during execution becomes a process and performs all the tasks for the user. A process has four sections. These sections are as follows: Stack, Heap, Text, and Data. The different states of a process are: Start, Ready, Running, Waiting, and Terminated or Exit.

**Ques 17. What do you mean by the First-Come-First-Serve scheduling algorithm?**

FCFS is an OS scheduling algorithm that executes requests and processes automatically. The OS stores these processes in the form of a queue in order of their arrival. Processes requesting the CPU first, get the CPU allocation first in this easy and simple algorithm with the help of a FIFO queue.

After entering the ready queue, the PCB of a process links itself with the tail of the queue. Thus, when the CPU becomes free, it is assigned to the process that is at the beginning of the queue. The process that has the least arrival time receives the processor first.

**Ques 18. What is a bootstrap program in OS?**

An OS initializes a bootstrap program during startup i.e., it is the first code that executes when a system starts. Booting is a bootstrapping process or program that loads the OS and ensures the correct working of the OS. Boot blocks store the OS at a fixed location on the disk where the OS locates the kernel and loads it into the main memory. After this process performs, the program starts its execution.

**Ques 19. State the difference between paging and segmentation?**

The following table states the differences between paging and segmentation:

<b>Paging</b>	<b>Segmentation</b>
Fix sized pages	Variable sized segments
Internal fragmentation	No internal fragmentation
Hardware decides the page size	The user decides the segment size
Faster memory access	Memory access is slower as compared to the paging
Page table stores data	Segmentation table stores data
Sharing of procedures is not allowed	Sharing of procedures is allowed
Cannot distinguish and secure procedures and data	Can separate and secure procedures and data
1-D address space	Multiple independent address spaces
A single integer address is divided into page numbers and offset by the hardware	The user divides address in segment number and offset

**Ques 20. What is the main objective of multiprogramming?**

In this, the multiple tasks are stored in the system memory that is acquired from the job pool and the OS picks one task and starts executing it. The OS fetches another job from the



memory when the current job requires an I/O. In the case of multiple jobs in a ready state, which job to choose is decided through the process of CPU Scheduling. It never leaves a CPU idle and maximizes CPU usage.

**Ques 21. What is RAID configuration in an OS? Also, state the different levels of RAID.**

RAID stands for Redundant Arrays of Independent Disks. It is a method that helps store data on multiple hard disks and helps achieve data redundancy that reduces data loss. It is a data storage virtualization technology that balances data protection, storage space, system performance, etc. It also improves the overall performance and reliability of data storage and increases the storage capacity of a system.

Following are the different levels of RAID:

RAID 0: Non-redundant striping: Increase server performance.

RAID 1: Mirroring and duplexing: Performs disk mirroring and helps implement fault tolerance in a simple manner.

RAID 2: Memory-style error-correcting codes: Uses dedicated hamming code parity.

RAID 3: Bit-interleaved Parity: Stores parity information using a dedicated parity drive.

RAID 4: Block-interleaved Parity: Similar to RAID 5, just confines all the parity data to a single drive.

RAID 5: Block-interleaved distributed Parity: Provides better performance as compared to RAID 1.

RAID 6: P+Q Redundancy: Provides fault tolerance for two drive failures.

**Ques 22. What is a Pipe? When is a pipe used?**

The pipe is used for inter-process communication. This half-duplex method allows communication between two related processes. A half-duplex method allows the first process to communicate with the second process. In order to achieve a full-duplex, we need to add another pipe.

**Ques 23. What do you mean by Reentrancy?**

A function in which various clients use and share a single copy of a program at the same time is known as Reentrancy. It doesn't deal with concurrency and has two major functions:

Program code is unable to change or modify itself.

Different disks store local data for every client process.

**Ques 24. State the differences between Multitasking and Multiprocessing.**

The following table states the differences between Multitasking and Multiprocessing:

<b>Multitasking</b>	<b>Multiprocessing</b>
Performs multiple tasks at the same time using a single processor	Performs multiple tasks at the same time using multiple processors
Only one CPU	More than one processes
More economical	Less economical

Less efficient

More efficient

Fast switching among tasks

Smooth processing of multiple tasks at a time

More time for execution

Less time for job processing

**Ques 25. Define Sockets in OS.**

A socket, used in client-server-based systems is the endpoint for IPC i.e., a combination of an IP address and port number. They make the creation of network-enabled programs easy and allow communication between two processes on the same or different machines. There are four types of sockets namely, Stream Sockets, Datagram Sockets, Sequenced Packet Sockets, and Raw Sockets.

**Ques 26. What is a zombie process?**

A defunct process that terminates or completes but the PCB is still present in the main memory as there is still an entry in the process table to report to its parent process. This process is known as a zombie process. In a way, it is dead as it doesn't consume any resources, but it exists too as it shows that resources are in possession of the process.

**Ques 27. What is cascading termination?**

When a process creates a new process, the identity of the child process passes onto the parent process. When the OS initiates the termination of the parent process or if the parent process exits the child process also needs to terminate.

**Ques 28. What is swap space?**

Swap space specifies the space that Linux uses to hold concurrent running processes temporarily. The user uses it when the RAM doesn't have enough space to hold all the executing programs.

**Ques 29. How does a system call works?**

Following are the steps on how a System Call works:

Step 1: The processor executes a process in the user mode until a system call interrupts it.

Step 2: Then on a priority basis, the system call is executed in the kernel mode.

Step 3: After the completion of system call execution, control returns to user mode.,

Step 4: The execution resumes in Kernel mode.

**Ques 30. What happens during a remote procedure call?**

There are two activities that take place during a Remote Procedure Call(RPC):

1. The OS suspends and transfers the calling environment and procedure parameters respectively, across the network and to the environment where the procedure executes.

2. The OS transfers back the result produced by a procedure to the calling environment. Execution also resumes just like a regular procedure call.