OPERATING SYSTEMS

LABORATORY MANUAL

OPERATING SYSTEMS LAB SYLLABUS

Implement the following programs on Linux platform using C language.

	Experiments
1	Develop a c program to implement the Process system calls (fork (), exec(), wait(), create process,terminate process)
2	Simulate the following CPU scheduling algorithms to find turnaround time and waiting time a) FCFS b) SJF c) Round Robin d) Priority.
3	Develop a C program to simulate producer-consumer problem using semaphores.
4	Develop a C program which demonstrates interprocess communication between a reader process and a writer process. Use mkfifo, open, read, write and close APIs in your program.
5	Develop a C program to simulate Bankers Algorithm for DeadLock Avoidance.
6	Develop a C program to simulate the following contiguous memory allocation Techniques: a) Worst fit b) Best fit c) First fit.
7	Develop a C program to simulate page replacement algorithms: a) FIFO b) LRU
8	Simulate following File Organization Techniques a) Single level directory b) Two level directory
9	Develop a C program to simulate the Linked file allocation strategies.
10	Develop a C program to simulate SCAN disk scheduling algorithm.

Course outcomes (Course Skill Set):

At the end of the course, the student will be able to:

CO 1. Explain the structure and functionality of operating system

CO 2. Apply appropriate CPU scheduling algorithms for the given problem.

CO 3. Analyse the various techniques for process synchronization and deadlock handling.CO 4. Apply the various techniques for memory management

CO 5. Explain file and secondary storage management strategies.CO 6. Describe the need for information protection mechanisms

Assessment Details (both CIE and SEE)

The weightage of Continuous Internal Evaluation (CIE) is 50% and for Semester End Exam (SEE) is 50%. Theminimum passing mark for the CIE is 40% of the maximum marks (20 marks out of 50) and for the SEE minimum passing mark is 35% of the maximum marks (18 out of 50 marks). A student shall be deemed tohave satisfied the academic requirements and earned the credits allotted to each subject/ course if the student secures a minimum of 40% (40 marks out of 100) in the sum total of the CIE (Continuous Internal Evaluation) and SEE (Semester End Examination) taken together.

CIE for the theory component of the IPCC (maximum marks 50)

- IPCC means practical portion integrated with the theory of the course.
- CIE marks for the theory component are **25 marks** and that for the practical component is **25 marks**.
- 25 marks for the theory component are split into **15 marks** for two Internal Assessment Tests (Two Tests,each of 15 Marks with 01-hour duration, are to be conducted) and **10 marks** for other assessment methods

PROGRAM 1

***Develop a c program to implement the Process system calls (fork (), exec(), wait(), create process,terminate process*)**

PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

int main() {
    // Variable to store the process ID.
    pid_t child_pid; char ch[3];
    // Create a child process using fork().
    child_pid = fork();
    if (child_pid == -1) {
        // Error occurred during fork.
        perror("fork");
        exit(EXIT_FAILURE);
    }
   if (child_pid == 0) {
        // This code executes in the child process.
        printf("Child process (PID: %d) is running.\n",
getpid());
        // Execute a different program using exec().
        //abort(); child terminated abnormally by sending
signal number 6 to the parent
     // return(-1); child terminates with exit status 255

     execl("/bin/date","date", NULL);

     exit(0);
    }
     else {
        // This code executes in the parent process.
        printf("Parent process (PID: %d) is waiting for the
child to terminate.\n", getpid());
        // Wait for the child process to terminate using
wait().
        int status;

     wait(&status);
     printf("parent resumes\n");
```

```
        if (WIFEXITED(status)) {
            printf("\nChild process (PID: %d) terminated with
status %d.\n", child_pid, WEXITSTATUS(status));
        } else if (WIFSIGNALED(status)) {
            printf("\nChild process (PID: %d) terminated due
to signal %d.\n", child_pid, WTERMSIG(status));
        } else {
            printf("\nChild process (PID: %d) terminated
abnormally.\n", child_pid);}

    }

    return 0;
}
```

TEST CASES:

Test case 1:

Parent process (PID: 11284) is waiting for the child to terminate.

Child process (PID: 11285) is running.

parent resumes

Child process (PID: 11285) terminated with status 255.

Test case 2:

guest-uivabk@student-OptiPlex-3040:~/Desktop$ ./a.out

Parent process (PID: 11356) is waiting for the child to terminate.

Child process (PID: 11357) is running.

parent resumes

Child process (PID: 11357) terminated due to signal 6.

Test Case 3:

guest-uivabk@student-OptiPlex-3040:~/Desktop$ ./a.out

Parent process (PID: 11433) is waiting for the child to terminate.

Child process (PID: 11434) is running.

Tue Oct 10 10:50:07 IST 2023

parent resumes

Child process (PID: 11434) terminated with status 0.

PROGRAM 2

Simulate the following CPU scheduling algorithms to find turnaround time and waiting time a) FCFS b) SJF c) Round Robin d) Priority.

DESCRIPTION

Assume all the processes arrive at the same time.

FCFS CPU SCHEDULING ALGORITHM:

For FCFS scheduling algorithm, read the number of processes/jobs in the system, their CPU burst times. The scheduling is performed on the basis of arrival time of the processes irrespective of their other parameters. Each process will be executed according to its arrival time. Calculate the waiting time and turnaround time of each of the processes accordingly.

SJF CPU SCHEDULING ALGORITHM:

For SJF scheduling algorithm, read the number of processes/jobs in the system, their CPU burst times. Arrange all the jobs in order with respect to their burst times. There may be two jobs in queue with the same execution time, and then FCFS approach is to be performed. Each process will be executed according to the length of its burst time. Then calculate the waiting time and turnaround time of each of the processes accordingly.

ROUND ROBIN CPU SCHEDULING ALGORITHM:

For round robin scheduling algorithm, read the number of processes/jobs in the system, their CPU burst times, and the size of the time slice. Time slices are assigned to each process in equal portions and in circular order, handling all processes execution. This allows every process to get an equal chance. Calculate the waiting time and turnaround time of each of the processes accordingly.

PRIORITY CPU SCHEDULING ALGORITHM:

For priority scheduling algorithm, read the number of processes/jobs in the system, their CPU burst times, and the priorities. Arrange all the jobs in order with respect to their priorities. There may be two jobs in queue with the same priority, and then FCFS approach is to be performed. Each process will be executed according to its priority. Calculate the waiting time and turnaround time of each of the processes accordingly.

PROGRAM: a) FCFS

```
#include<stdio.h>
int main()
{
int bt[20], wt[20], tat[20], i, n;
float wtavg, tatavg;
printf("\nEnter the number of processes -- ");
scanf("%d", &n);
```

```
for(i=0;i<n;i++)
{
printf("\nEnter Burst Time for Process %d -- ", i);
scanf("%d", &bt[i]);
}
wt[0] = wtavg = 0;
tat[0] = tatavg = bt[0];
for(i=1;i<n;i++)
{
wt[i] = wt[i-1] +bt[i-1];
tat[i] = tat[i-1] +bt[i];
wtavg = wtavg + wt[i];
tatavg = tatavg + tat[i];
}
printf("\t PROCESS \tBURST TIME \t WAITING TIME\t TURNAROUND
 TIME\n");
for(i=0;i<n;i++)
printf("\n\t P%d \t\t %d \t\t %d \t\t %d", i, bt[i], wt[i],
 tat[i]);
printf("\nAverage Waiting Time -- %f", wtavg/n);
printf("\nAverage Turnaround Time -- %f", tatavg/n);
}
```

TEST CASES:

Enter the number of processes -- 3

Enter Burst Time for Process 0 -- 24

Enter Burst Time for Process 1 -- 3

Enter Burst Time for Process 2 -- 3

PROCESS	BURST TIME	WAITING TIME	TURNAROUND TIME
P0	24	0	24
P1	3	24	27
P2	3	27	30

Average Waiting Time -- 17.000000

Average Turnaround Time -- 27.000000

PROGRAM: b) SJF

```
#include<stdio.h>
int main()
{
int p[20], bt[20], wt[20], tat[20], i, k, n, temp;
float wtavg, tatavg;
printf("\nEnter the number of processes -- ");
scanf("%d", &n);
for(i=0;i<n;i++)
{
p[i]=i;
printf("Enter Burst Time for Process %d -- ", i);
scanf("%d", &bt[i]);
}
for(i=0;i<n;i++)
for(k=i+1;k<n;k++)
if(bt[i]>bt[k])
{
temp=bt[i];
 bt[i]=bt[k];
bt[k]=temp;

temp=p[i];
p[i]=p[k];
p[k]=temp;
}
wt[0] = wtavg = 0;
tat[0] = tatavg = bt[0];
for(i=1;i<n;i++)
{
wt[i] = wt[i-1] +bt[i-1];
tat[i] = tat[i-1] +bt[i];
wtavg = wtavg + wt[i];
```

```
tatavg = tatavg + tat[i];
}
printf("\n\t PROCESS \tBURST TIME \t WAITING TIME\t TURNAROUND
TIME\n");
for(i=0;i<n;i++)
printf("\n\t P%d \t\t %d \t\t %d \t\t %d", p[i], bt[i], wt[i],
 tat[i]);
printf("\nAverage Waiting Time -- %f", wtavg/n);
printf("\nAverage Turnaround Time -- %f", tatavg/n);
}
```

TEST CASES:

guest-uivabk@student-OptiPlex-3040:~/Desktop$ cc sjf.c

guest-uivabk@student-OptiPlex-3040:~/Desktop$ ./a.out

Enter the number of processes -- 3

Enter Burst Time for Process 0 -- 10

Enter Burst Time for Process 1 -- 8

Enter Burst Time for Process 2 -- 7

PROCESS	BURST TIME	WAITING TIME	TURNAROUND TIME
P2	7	0	7
P1	8	7	15
P0	10	15	25

Average Waiting Time -- 7.333333

Average Turnaround Time -- 15.666667

PROGRAM: c) Round Robin

```
#include<stdio.h>
int main()
{
int i,j,n,bu[10],wa[10],tat[10],t,ct[10],max;
float awt=0,att=0,temp=0;
printf("Enter the no of processes -- ");
scanf("%d",&n);
for(i=0;i<n;i++)
{
```

```
printf("\nEnter Burst Time for process %d -- ", i+1);
scanf("%d",&bu[i]);
ct[i]=bu[i];
}
printf("\nEnter the size of time slice -- ");
scanf("%d",&t);
max=bu[0];

for(i=1;i<n;i++)
if(max<bu[i])
max=bu[i];

for(j=0;j<(max/t)+1;j++)
for(i=0;i<n;i++)
if(bu[i]!=0)
if(bu[i]<=t)
{
    tat[i]=temp+bu[i];
     temp=temp+bu[i];
     bu[i]=0;
}
else
{
bu[i]=bu[i]-t;
temp=temp+t;
}
for(i=0;i<n;i++)
{
wa[i]=tat[i]-ct[i];
att+=tat[i];
awt+=wa[i];
}
printf("\nThe Average Turnaround time is -- %f",att/n);
```

```
printf("\nThe Average Waiting time is -- %f ",awt/n);
printf("\n\tPROCESS\t BURST TIME \t WAITING TIME\tTURNAROUND
TIME\n");
for(i=0;i<n;i++)
printf("\t%d \t %d \t\t %d \t\t %d \n",i+1,ct[i],wa[i],tat[i]);
}
```

TEST CASES:

Test Case 1:

Enter the no of processes -- 3

Enter Burst Time for process 1 -- 10

Enter Burst Time for process 2 -- 8

Enter Burst Time for process 3 -- 7

Enter the size of time slice -- 5

The Average Turnaround time is -- 22.666666

The Average Waiting time is -- 14.333333

PROCESS	BURST TIME	WAITING TIME	TURNAROUND TIME
1	10	10	20
2	8	15	23
3	7	18	25

Test case 2:

guest-uivabk@student-OptiPlex-3040:~/Desktop$ ./a.out

Enter the no of processes -- 3

Enter Burst Time for process 1 -- 10

Enter Burst Time for process 2 -- 8

Enter Burst Time for process 3 -- 7

Enter the size of time slice -- 4

The Average Turnaround time is -- 22.666666

The Average Waiting time is -- 14.333333

PROCESS	BURST TIME	WAITING TIME	TURNAROUND TIME
1	10	15	25
2	8	12	20
3	7	16	23

PROGRAM: d) Priority

```
#include<stdio.h>
int main()
{
int p[20],bt[20],pri[20], wt[20],tat[20],i, k, n, temp;
float wtavg, tatavg;
printf("Enter the number of processes --- ");
scanf("%d",&n);
for(i=0;i<n;i++)
{
p[i] = i;
printf("Enter only positive numbers\n");
printf("Enter the Burst Time & Priority of Process %d --- ",i);
scanf("%d %d",&bt[i], &pri[i]);
}
for(i=0;i<n;i++)
for(k=i+1;k<n;k++)
if(pri[i] > pri[k])
{
temp=p[i];
p[i]=p[k];
p[k]=temp;

temp=bt[i];
bt[i]=bt[k];
bt[k]=temp;

temp=pri[i];
pri[i]=pri[k];
pri[k]=temp;
}
wtavg = wt[0] = 0;
```

```
tatavg = tat[0] = bt[0];
for(i=1;i<n;i++)
{
wt[i] = wt[i-1] + bt[i-1];
tat[i] = tat[i-1] + bt[i];
wtavg = wtavg + wt[i];
tatavg = tatavg + tat[i];
}
printf("\nPROCESS\t\tPRIORITY\tBURST TIME\tWAITING
TIME\tTURNAROUND TIME");
for(i=0;i<n;i++)
printf("\n%d \t\t %d \t\t %d \t\t %d \t\t %d
",p[i],pri[i],bt[i],wt[i],tat[i]);
printf("\nAverage Waiting Time is --- %f",wtavg/n);
printf("\nAverage Turnaround Time is --- %f",tatavg/n);
}
```

TEST CASES:

Enter the number of processes --- 3

Enter the Burst Time & Priority of Process 0 --- 10 3

Enter the Burst Time & Priority of Process 1 --- 8 1

Enter the Burst Time & Priority of Process 2 --- 7 2

PROCESS	PRIORITY	BURST TIME	WAITING TIME	TURNAROUND TIME
1	1	8	0	8
2	2	7	8	15
0	3	10	15	25

Average Waiting Time is --- 7.666667

Average Turnaround Time is --- 16.000000

PROGRAM 3

Develop a C program to simulate producer-consumer problem using semaphores.

DESCRIPTION:

Producer-consumer problem, is a common paradigm for cooperating processes. A producer process produces information that is consumed by a consumer process. One solution to the producer-consumer problem uses shared memory. To allow producer and consumer processes to run concurrently, there must be available a buffer of items that can be filled by the producer and emptied by the consumer. This buffer will reside in a region of memory that is shared by the producer and consumer processes. A producer can produce one item while the consumer is consuming another item. The producer and consumer must be synchronized, so that the consumer does not try to consume an item that has not yet been produced.

PROGRAM:

```
#include<stdio.h>
#include <stdlib.h>
int mutex = 1;
 int full = 0;
int empty = 3, x = 0;
void producer()
{
    --mutex;
    ++full;
    --empty;
    x++;
    printf("\nProducer produces item %d",x);
    ++mutex;
}

void consumer()
{
    --mutex;
    --full;
    ++empty;
    printf("\nConsumer consumes item %d",x);
    x--;
    ++mutex;
```

```
}
int main()
{
    int n, i;
    printf("\n1. Press 1 for Producer"
           "\n2. Press 2 for Consumer"
           "\n3. Press 3 for Exit");

#pragma omp critical

    for (i = 1; i > 0; i++)
{
       printf("\nEnter your choice:");
        scanf("%d", &n);
        // Switch Cases
        switch (n) {
        case 1:
            if ((mutex == 1) && (empty != 0))
            {
                producer();
            }
            else
            {
                printf("Buffer is full!");
            }
            break;
        case 2:
            if ((mutex == 1)&& (full != 0))
            {
                consumer();
            }
            else
            {
                printf("Buffer is empty!");
            }
```

```
            break;
          case 3:
            exit(0);
            break;
         }
     }
}
```

TEST CASES:

1. Press 1 for Producer
2. Press 2 for Consumer
3. Press 3 for Exit

Enter your choice:1

Producer produces item 1

Enter your choice:1

Producer produces item 2

Enter your choice:1

Producer produces item 3

Enter your choice:1

Buffer is full!

Enter your choice:2

Consumer consumes item 3

Enter your choice:2

Consumer consumes item 2

Enter your choice:2

Consumer consumes item 1

Enter your choice:2

Buffer is empty!

Enter your choice:3

PROGRAM 4

Develop a C program which demonstrates interprocess communication between a reader process and a writer process. Use mkfifo, open, read, write and close APIs in your program.

PROGRAM:

```
// Create seperate files for reader and writer processes
// two terminals to run the program
// unless the writer process writes into the buffer, reader can not read
/*Writer Process*/
#include <stdio.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
 int fd;
 char buf[1024]="Hello RNSIT";
 /* create the FIFO (named pipe) */
 char * myfifo = "/tmp/guest-uivabk/Desktop/tmp";
 mkfifo(myfifo, 0666);
 printf("Run Reader process to read the FIFO File\n");
 fd = open(myfifo, O_WRONLY);
 write(fd,buf,sizeof(buf));
 close(fd);
 unlink(myfifo); /* remove the FIFO */
 return 0;
}

/* Reader Process */
 #include <fcntl.h>
 #include <sys/stat.h>
```

```
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#define MAX_BUF 1024
int main()
{
int fd;
/* A temp FIFO file is not created in reader */
char *myfifo = "/tmp/guest-uivabk/Desktop/tmp";
char buf[MAX_BUF];
/* open, read, and display the message from the FIFO */
fd = open(myfifo, O_RDONLY);
read(fd, buf, MAX_BUF);
printf("Reader process has read : %s\n", buf);
close(fd);
return 0;
}
```

TEST CASES:

guest-uivabk@student-OptiPlex-3040:~/Desktop$ cc writer.c

guest-uivabk@student-OptiPlex-3040:~/Desktop$ ./a.out

Run Reader process to read the FIFO File

guest-uivabk@student-OptiPlex-3040:~/Desktop$ cc reader.c

guest-uivabk@student-OptiPlex-3040:~/Desktop$ ./a.out

Reader process has read : Hello RNSIT

PROGRAM 5

Develop a C program to simulate Bankers Algorithm for DeadLock Avoidance.

DESCRIPTION

In a multiprogramming environment, several processes may compete for a finite number of resources. A processrequests resources; if the resources are not available at that time, the process enters a waiting state. Sometimes, a waiting process is never again able to change state, because the resources it has requested are held by other waiting processes. This situation is called a deadlock. Deadlock avoidance is one of the techniques for handling deadlocks. This approach requires that the operating system be given in advance additional information concerning which resources a process will request and use during its lifetime. With this additional knowledge, it can decide for each request whether or not the process should wait. To decide whether the current request can be satisfied or must be delayed, the system must consider the resources currently available, the resources currently allocated to each process, and the future requests and releases of each process. Banker's algorithm is a deadlock avoidance algorithm that is applicable to a system with multiple instances of each resource type.

PROGRAM:

```
#include<stdio.h>
struct file
{
int all[10];
int max[10];
int need[10];
int flag;
};
void main()
{
struct file f[10]; int fl;
int i, j, k, p, b, n, r, g, cnt=0, id, newr;
int avail[10],seq[10];
printf("Enter number of processes -- ");
 scanf("%d",&n);
printf("Enter number of resources -- ");
 scanf("%d",&r);
for(i=0;i<n;i++)
```

```
{
printf("Enter details for P%d",i);
printf("\nEnter allocation\t -- \t");
for(j=0;j<r;j++)
scanf("%d",&f[i].all[j]);
printf("Enter Max\t\t -- \t");
for(j=0;j<r;j++)
scanf("%d",&f[i].max[j]);
f[i].flag=0;
}
printf("\nEnter Available Resources\t -- \t");
for(i=0;i<r;i++)
scanf("%d",&avail[i]);
for(i=0;i<n;i++)
{
for(j=0;j<r;j++)
{
 f[i].need[j]=f[i].max[j]-f[i].all[j];
if(f[i].need[j]<0)
f[i].need[j]=0;
}
}
cnt=0; fl=0;
while(cnt!=n)
{
g=0;
for(j=0;j<n;j++)
{
if(f[j].flag==0)
{
b=0;
for(p=0;p<r;p++)
```

```
{
if(avail[p]>=f[j].need[p])
    b=b+1;
else
    b=b-1;
}
if(b==r)
{
printf("\nP%d is visited",j);
seq[fl++]=j;
f[j].flag=1;
for(k=0;k<r;k++)
avail[k]=avail[k]+f[j].all[k];
cnt=cnt+1;
printf("(");
for(k=0;k<r;k++)
printf("%3d",avail[k]);
printf(")");
g=1;
}
}
}
if(g==0)
{
printf("\n REQUEST NOT GRANTED -- DEADLOCK OCCURRED");
printf("\n SYSTEM IS IN UNSAFE STATE");
goto y;
}
}
printf("\nSYSTEM IS IN SAFE STATE");
printf("\nThe Safe Sequence is -- (");
for(i=0;i<fl;i++)
```

```
printf("P%d ",seq[i]);
printf(")");
y: printf("\nProcess\t\tAllocation\t\tMax\t\t\tNeed\n");
for(i=0;i<n;i++)
{
printf("P%d\t",i);
for(j=0;j<r;j++)
printf("%6d",f[i].all[j]);
for(j=0;j<r;j++)
printf("%6d",f[i].max[j]);
for(j=0;j<r;j++)
printf("%6d",f[i].need[j]); printf("\n");
}
}
```

TEST CASES:

guest-yk70yi@student-OptiPlex-3040:~$ ./a.out
Enter number of processes -- 5
Enter number of resources -- 3
Enter details for P0
Enter allocation -- 0 1 0
Enter Max -- 7 5 3
Enter details for P1
Enter allocation -- 2 0 0
Enter Max -- 3 2 2
Enter details for P2
Enter allocation -- 3 0 2
Enter Max -- 9 0 2
Enter details for P3
Enter allocation -- 2 1 1
Enter Max -- 2 2 2
Enter details for P4
Enter allocation -- 0 0 2

Enter Max -- 4 3 3

Enter Available Resources -- 3 3 2

P1 is visited(5 3 2)

P3 is visited(7 4 3)

P4 is visited(7 4 5)

P0 is visited(7 5 5)

P2 is visited(10 5 7)

SYSTEM IS IN SAFE STATE

The Safe Sequence is -- (P1 P3 P4 P0 P2)

Process	Allocation	Max	Need
P0	0 1 0	7 5 3	7 4 3
P1	2 0 0	3 2 2	1 2 2
P2	3 0 2	9 0 2	6 0 0
P3	2 1 1	2 2 2	0 1 1
P4	0 0 2	4 3 3	4 3 1

PROGRAM 6

Develop a C program to simulate the following contiguous memory allocation Techniques:

a) Worst fit b) Best fit c) First fit.

DESCRIPTION

One of the simplest methods for memory allocation is to divide memory into several fixed-sized partitions. Each partition may contain exactly one process. In this multiple-partition method, when a partition is free, a process isselected from the input queue and is loaded into the free partition. When the process terminates, the partition becomes available for another process. The operating system keeps a table indicating which parts of memory are available and which are occupied. Finally, when a process arrives and needs memory, a memory section large enough for this process is provided. When it is time to load or swap a process into main memory, and if there is more than one free block of memory of sufficient size, then the operating system must decide which free block to allocate. Best-fit strategy chooses the block that is closest in size to the request. First-fit chooses the first available block that is large enough. Worst-fit chooses the largest available block.

PROGRAM: a) WORST FIT

```
#include<stdio.h>
#define max 25
void main()
{
int frag[max],b[max],f[max],i,j,nb,nf,temp,highest=0;
static int bf[max],ff[max];
printf("\n\tMemory Management Scheme - Worst Fit");
printf("\nEnter the number of blocks:");
scanf("%d",&nb);
printf("Enter the number of files:");
scanf("%d",&nf);
printf("\nEnter the size of the blocks:-\n");
for(i=1;i<=nb;i++)
{
printf("Block %d:",i);
scanf("%d",&b[i]);
}
printf("Enter the size of the files :-\n");
```

```
for(i=1;i<=nf;i++)
{
printf("File %d:",i);
scanf("%d",&f[i]);
}
for(i=1;i<=nf;i++)
{
for(j=1;j<=nb;j++)
{
if(bf[j]!=1) //if bf[j] is not allocated
{
temp=b[j]-f[i]; if(temp>=0)
if(highest<temp)
{
 ff[i]=j; highest=temp;
}
}
}
frag[i]=highest;
bf[ff[i]]=1;
highest=0;
}
printf("\nFile_no:\tFile_size
 :\tBlock_no:\tBlock_size:\tFragement");
for(i=1;i<=nf;i++)
printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],fra
g[i]);
}
```

TEST CASES:

guest-uivabk@student-OptiPlex-3040:~/Desktop$ cc worstfit.c

guest-uivabk@student-OptiPlex-3040:~/Desktop$ ./a.out

Memory Management Scheme - Worst Fit

Enter the number of blocks:3

Enter the number of files:2

Enter the size of the blocks:-

Block 1:5

Block 2:2

Block 3:7

Enter the size of the files :-

File 1:1

File 2:4

File_no:	File_size :	Block_no:	Block_size:	Fragement
1	1	3	7	6
2	4	1	5	1

PROGRAM: b) BEST FIT

```
#include<stdio.h>
#define max 25
void main()
{
int frag[max],b[max],f[max],i,j,nb,nf,temp,lowest=10000;
static int bf[max],ff[max];
printf("\n\tMemory Management Scheme - Best Fit");
printf("\nEnter the number of blocks:");
scanf("%d",&nb);
printf("Enter the number of files:");
scanf("%d",&nf);
printf("\nEnter the size of the blocks:-\n");
for(i=1;i<=nb;i++)
{
printf("Block %d:",i);
scanf("%d",&b[i]);
}
printf("Enter the size of the files :-\n");
for(i=1;i<=nf;i++)
```

```
{
printf("File %d:",i);
scanf("%d",&f[i]);
}
for(i=1;i<=nf;i++)
{
for(j=1;j<=nb;j++)
{
if(bf[j]!=1)
{
temp=b[j]-f[i]; if(temp>=0)
if(lowest>temp)
{
ff[i]=j;
lowest=temp;
}
}
}
frag[i]=lowest; bf[ff[i]]=1; lowest=10000;
}
printf("\nFile No\tFile Size \tBlock No\tBlock
 Size\tFragment");
for(i=1;i<=nf && ff[i]!=0;i++)
printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],fra
g[i]);
}
```

TEST CASES:

guest-uivabk@student-OptiPlex-3040:~/Desktop$ cc bestfit.c

guest-uivabk@student-OptiPlex-3040:~/Desktop$ ./a.out

Memory Management Scheme - Best Fit

Enter the number of blocks:3

Enter the number of files:2

Enter the size of the blocks:-

Block 1:5

Block 2:2

Block 3:7

Enter the size of the files :-

File 1:1

File 2:4

File No	File Size	Block No	Block Size	Fragment
1	1	2	2	1
2	4	1	5	1

PROGRAM: c) FIRST FIT

```
#include<stdio.h>
#define max 25
void main()
{
int frag[max],b[max],f[max],i,j,nb,nf,temp;
static int bf[max],ff[max];
printf("\n\tMemory Management Scheme - First Fit");
printf("\nEnter the number of blocks:");
scanf("%d",&nb);
printf("Enter the number of files:");
scanf("%d",&nf);
printf("\nEnter the size of the blocks:-\n");
for(i=1;i<=nb;i++)
{
printf("Block %d:",i);
scanf("%d",&b[i]);
}
printf("Enter the size of the files :-\n");
for(i=1;i<=nf;i++)
{
printf("File %d:",i);
```

```
scanf("%d",&f[i]);
}
for(i=1;i<=nf;i++)
{
for(j=1;j<=nb;j++)
{
if(bf[j]!=1)
{
temp=b[j]-f[i]; if(temp>=0)
{
ff[i]=j;
break;
}
}
}
frag[i]=temp; bf[ff[i]]=1;
}
printf("\nFile_no:\tFile_size
:\tBlock_no:\tBlock_size:\tFragement");
for(i=1;i<=nf;i++)
printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],fra
g[i]);
}
```

TEST CASES:

TEST CASE 1:

guest-uivabk@student-OptiPlex-3040:~/Desktop$ ./a.out

Memory Management Scheme - First Fit

Enter the number of blocks:3

Enter the number of files:2

Enter the size of the blocks:-

Block 1:5

Block 2:2

Block 3:7

Enter the size of the files :-

File 1:1

File 2:4

File_no:	File_size :	Block_no:	Block_size:	Fragement
1	1	1	5	4
2	4	3	7	3

TEST CASE 2:

guest-uivabk@student-OptiPlex-3040:~/Desktop$ ./a.out

Memory Management Scheme - First Fit

Enter the number of blocks:3

Enter the number of files:2

Enter the size of the blocks:-

Block 1:5

Block 2:2

Block 3:7

Enter the size of the files :-

File 1:4

File 2:1

File_no:	File_size :	Block_no:	Block_size:	Fragement
1	4	1	5	1
2	1	2	2	1

PROGRAM 7

Develop a C program to simulate page replacement algorithms:

a) FIFO b) LRU

DESCRIPTION:

Page replacement is basic to demand paging. It completes the separation between logical memory and physical memory. With this mechanism, an enormous virtual memory can be provided for programmers on a smaller physical memory. There are many different page-replacement algorithms. Every operating system probably has its own replacement scheme. A FIFO replacement algorithm associates with each page the time when that page was brought into memory. When a page must be replaced, the oldest page is chosen. If the recent past is used as an approximation of the near future, then the page that has not been used for the longest period of time can be replaced. This approach is the Least Recently Used (LRU) algorithm. LRU replacement associates with each page the time of that page's last use. When a page must be replaced, LRU chooses the page that has not been used for the longest period of time. Least frequently used (LFU) page-replacement algorithm requires that the page with the smallest count be replaced. The reason for this selection is that an actively used page should have a large reference count.

PROGRAM: a) FIFO

```
#include<stdio.h>
void main()
{
int i, j, k, f, pf=0, count=0, rs[25], m[10], n;
printf("\n Enter the length of reference string -- ");
scanf("%d",&n);
printf("\n Enter the reference string -- ");
for(i=0;i<n;i++)
scanf("%d",&rs[i]);
printf("\n Enter no. of frames -- ");
scanf("%d",&f);
for(i=0;i<f;i++)
m[i]=-1;
printf("\n The Page Replacement Process is -- \n");
for(i=0;i<n;i++)
{
for(k=0;k<f;k++)
```

```
{
if(m[k]==rs[i])
break;
}
if(k==f)
{
m[count++]=rs[i];
pf++;
}
for(j=0;j<f;j++)
printf("\t%d",m[j]);
if(k==f)
printf("\tPF No. %d",pf);
printf("\n");
if(count==f)
count=0;
}
printf("\n The number of Page Faults using FIFO are %d",pf);
}
```

TEST CASES:

guest-yk70yi@student-OptiPlex-3040:~$ ./a.out

Enter the length of reference string -- 20

Enter the reference string -- 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

Enter no. of frames -- 3

The Page Replacement Process is --

7	-1	-1	PF No. 1
7	0	-1	PF No. 2
7	0	1	PF No. 3
2	0	1	PF No. 4
2	0	1	
2	3	1	PF No. 5
2	3	0	PF No. 6

4	3	0	PF No. 7
4	2	0	PF No. 8
4	2	3	PF No. 9
0	2	3	PF No. 10
0	2	3	
0	2	3	
0	1	3	PF No. 11
0	1	2	PF No. 12
0	1	2	
0	1	2	
7	1	2	PF No. 13
7	0	2	PF No. 14
7	0	1	PF No. 15

The number of Page Faults using FIFO are 15

PROGRAM: b) LRU

```
#include<stdio.h>
int main()
{
int i, j , k, min, rs[25], m[10], count[10], flag[25], n, f,
pf=0, next=1;
printf("Enter the length of reference string -- ");
scanf("%d",&n);
printf("Enter the reference string -- ");
for(i=0;i<n;i++)
{
scanf("%d",&rs[i]);
flag[i]=0;
}
printf("Enter the number of frames -- ");
scanf("%d",&f);
for(i=0;i<f;i++)
{
```

```
count[i]=0;
m[i]=-1;
}
printf("\nThe Page Replacement process is -- \n");
for(i=0;i<n;i++)
{
for(j=0;j<f;j++)
{
if(m[j]==rs[i])
{
flag[i]=1;
 count[j]=next;
next++;
}
}
if(flag[i]==0)
{
if(i<f)
{
m[i]=rs[i];
count[i]=next;
next++;
}
else
{
min=0;
for(j=1;j<f;j++)
if(count[min] > count[j])
    min=j;

m[min]=rs[i];
count[min]=next;
```

```
next++;
}
pf++;
}
for(j=0;j<f;j++)
printf("%d\t", m[j]);
if(flag[i]==0)
printf("PF No. -- %d" , pf);
printf("\n");
}
printf("\nThe number of page faults using LRU are %d",pf);
}
```

TEST CASES:

guest-yk70yi@student-OptiPlex-3040:~$ ./a.out

Enter the length of reference string -- 20

Enter the reference string -- 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

Enter the number of frames -- 3

The Page Replacement process is --

7	-1	-1	PF No. -- 1
7	0	-1	PF No. -- 2
7	0	1	PF No. -- 3
2	0	1	PF No. -- 4
2	0	1	
2	0	3	PF No. -- 5
2	0	3	
4	0	3	PF No. -- 6
4	0	2	PF No. -- 7
4	3	2	PF No. -- 8
0	3	2	PF No. -- 9
0	3	2	
0	3	2	
1	3	2	PF No. -- 10

1 3 2

1 0 2 PF No. -- 11

1 0 2

1 0 7 PF No. -- 12

1 0 7

1 0 7

The number of page faults using LRU are 12

PROGRAM 8

Simulate following File Organization Techniques a) Single level directory b) Two level directory

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: a) Single Level Directory

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_FILES 10
#define MAX_FILENAME_LENGTH 10
char directoryName[MAX_FILENAME_LENGTH];
char filenames[MAX_FILES][MAX_FILENAME_LENGTH];
int fileCount = 0;
void createFile() {
if (fileCount < MAX_FILES) {
char filename[MAX_FILENAME_LENGTH];
printf("Enter the name of the file: ");
scanf("%s", filename);
strcpy(filenames[fileCount], filename);
fileCount++;
printf("File %s created.\n", filename);
} else {
```

```
printf("Directory is full, cannot create more files.\n");
}
}
void deleteFile() {
if (fileCount == 0) {
printf("Directory is empty, nothing to delete.\n");
return;
}
char filename[MAX_FILENAME_LENGTH];
printf("Enter the name of the file to delete: ");
scanf("%s", filename);
int found = 0;
for (int i = 0; i < fileCount; i++) {
if (strcmp(filename, filenames[i]) == 0) {
found = 1;
printf("File %s is deleted.\n", filename);
strcpy(filenames[i], filenames[fileCount - 1]);
fileCount--;
break;
}
}
if (!found) {
printf("File %s not found.\n", filename);
}
}
void displayFiles() {
if (fileCount == 0) {
printf("Directory is empty.\n");
} else {
printf("Files in the directory %s:\n", directoryName);
for (int i = 0; i < fileCount; i++) {
printf("\t%s\n", filenames[i]);
```

```
}
}
}
int main() {
printf("Enter the name of the directory: ");
scanf("%s", directoryName);
while (1) {
printf("\n1. Create File\t2. Delete File\t3. Search File\n4.
Display Files\t5. Exit\nEnter your choice:");
int choice;
scanf("%d", &choice);
switch (choice) {
case 1:
createFile();
break;
case 2:
deleteFile();
break;
case 3:
printf("Enter the name of the file: ");
char searchFilename[MAX_FILENAME_LENGTH];
scanf("%s", searchFilename);
int found = 0;
for (int i = 0; i < fileCount; i++) {
if (strcmp(searchFilename, filenames[i]) == 0) {
found = 1;
printf("File %s is found.\n", searchFilename);
break;
}
}
if (!found) {
printf("File %s not found.\n", searchFilename);
```

```
}
break;
case 4:
displayFiles();
break;
case 5:
exit(0);
default:
printf("Invalid choice, please try again.\n");
}
}
return 0;
}
```

TEST CASES:

guest-yk70yi@student-OptiPlex-3040:~$ ./a.out
Enter the name of the directory: rns

1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit
Enter your choice:1
Enter the name of the file: cse
File cse created.

1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit
Enter your choice:1
Enter the name of the file: ise
File ise created.

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

```
Files in the directory rns:
        cse
        ise

1. Create File  2. Delete File  3. Search File
4. Display Files        5. Exit
Enter your choice:3
Enter the name of the file: ai
File ai not found.

1. Create File  2. Delete File  3. Search File
4. Display Files        5. Exit
Enter your choice:3
Enter the name of the file: ise
File ise is 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 to delete: ise
File ise is deleted.

1. Create File  2. Delete File  3. Search File
4. Display Files        5. Exit
Enter your choice:4
Files in the directory rns:
        cse

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

PROGRAM: b) Two Level Directory

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
struct Directory {
char dname[10];
char fname[10][10];
int fcnt;
};
int main() {
struct Directory dir[10];
int i, ch, dcnt = 0, k;
char f[30], d[30];
while (1) {
printf("\n\n1. Create Directory\t2. Create File\t3. Delete
File\n4. Search File\t\t5. Display\t6.
Exit\nEnter your choice: ");
scanf("%d", &ch);
switch (ch) {
case 1:
if (dcnt < 10) {
printf("\nEnter name of directory: ");
scanf("%s", dir[dcnt].dname);
dir[dcnt].fcnt = 0;
dcnt++;
printf("Directory created\n");
} else {
printf("Maximum directory limit reached.\n");
}
break;
case 2:
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", dir[i].fname[dir[i].fcnt]);
dir[i].fcnt++;
printf("File created\n");
break;
}
}
if (i == dcnt) {
printf("Directory %s not found\n", 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\n", f);
dir[i].fcnt--;
strcpy(dir[i].fname[k], dir[i].fname[dir[i].fcnt]);
break;
}
}
if (k == dir[i].fcnt) {
printf("File %s not found\n", f);
}
break;
```

```
}
}
if (i == dcnt) {
printf("Directory %s not found\n", d);
}
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\n", f);
break;
}
}
if (k == dir[i].fcnt) {
printf("File %s not found\n", f);
}
break;
}
}
if (i == dcnt) {
printf("Directory %s not found\n", d);
}
break;
case 5:
if (dcnt == 0) {
printf("\nNo Directories\n");
```

```
} else {
printf("\nDirectory\tFiles\n");
for (i = 0; i < dcnt; i++) {
printf("%s\t\t", dir[i].dname);
for (k = 0; k < dir[i].fcnt; k++) {
printf("\t%s", dir[i].fname[k]);
}
printf("\n");
}
}
break;
case 6:
exit(0);
default:
printf("Invalid choice, please try again.\n");
}
}
return 0;
}
```

TEST CASES:

PROGRAM 9

Develop a C program to simulate the Linked file allocation strategies.

DESCRIPTION:

A file is a collection of data, usually stored on disk. As a logical entity, a file enables to divide data into meaningful groups. As a physical entity, a file should be considered in terms of its organization. The term "file organization" refers to the way in which data is stored in a file and, consequently, the method(s) by which it can be accessed.

LINKED FILE ALLOCATION:

With linked allocation, each file is a linked list of disk blocks; the disk blocks may be scattered anywhere on the disk. The directory contains a pointer to the first and last blocks of the file. Each block contains a pointer to the next block.

PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
int f[50], p, i, st, len, j, c, k, a, fn=0;
  for (i = 0; i < 50; i++)
        f[i] = 0;
   /* printf("Enter how many blocks already allocated: ");
    scanf("%d", &p);
    printf("Enter blocks already allocated: ");
    for (i = 0; i < p; i++) {
        scanf("%d", &a);
        f[a] = 1;
    }*/
x:
     fn=fn+1;
     printf("Enter index starting block and length: ");
     scanf("%d%d", &st, &len);
    k = len;
    if (f[st] == 0)
```

```
    {
        for (j = st; j < (st + k); j++){
            if (f[j] == 0)
            {
                f[j] = fn;
                printf("%d-------->%d\n", j, f[j]);
            }
            else{
                printf("%d Block is already allocated \n", j);
                k++;
            }
        }
    }
    else
        printf("%d starting block is already allocated \n",
st);
    printf("Do you want to enter more file(Yes - 1/No - 0)");
    scanf("%d", &c);
    if (c == 1)
     goto x;
    else
        exit(0);
    return 0;
}
```

TEST CASES:

guest-uivabk@student-OptiPlex-3040:~/Desktop$ cc pgm9.c

guest-uivabk@student-OptiPlex-3040:~/Desktop$ ./a.out

Enter index starting block and length: 1 3

1------>1

2------>1

3------>1

Do you want to enter more file(Yes - 1/No - 0)1

Enter index starting block and length: 5 3

5------>2

6------>2

7------>2

Do you want to enter more file(Yes - 1/No - 0)1

Enter index starting block and length: 4 5

4------>3

5 Block is already allocated

6 Block is already allocated

7 Block is already allocated

8------>3

9------>3

10------>3

11------>3

Do you want to enter more file(Yes - 1/No - 0)0

PROGRAM 10

Develop a C program to simulate SCAN disk scheduling algorithm.

DESCRIPTION:

One of the responsibilities of the operating system is to use the hardware efficiently. For the disk drives, meeting this responsibility entails having fast access time and large disk bandwidth. Both the access time and the bandwidth can be improved by managing the order in which disk I/O requests are serviced which is called as diskscheduling. The simplest form of disk scheduling is, of course, the first-come, first-served (FCFS) algorithm. This algorithm is intrinsically fair, but it generally does not provide the fastest service. In the SCAN algorithm, the disk arm starts at one end, and moves towards the other end, servicing requests as it reaches each cylinder, until it gets to the other end of the disk. At the other end, the direction of head movement is reversed, and servicing continues. The head continuously scans back and forth across the disk. C-SCAN is a variant of SCAN designed to provide a more uniform wait time. Like SCAN, C-SCAN moves the head from one end of the disk to the other, servicing requests along the way. When the head reaches the other end, however, it immediately returns to the beginning of the disk without servicing any requests on the return trip.

PROGRAM:

```
// head movement towards left
#include<stdio.h>
int main()
{
int t[20], d[20], h, i, j, n, temp, k, atr[20], tot, p, sum=0;
printf("enter the no of tracks to be traveresed");
scanf("%d'",&n);
printf("enter the position of head");
scanf("%d",&h);
t[0]=0;
t[1]=h;
printf("enter the tracks");
for(i=2;i<n+2;i++)
scanf("%d",&t[i]);
for(i=0;i<n+2;i++)
{
for(j=0;j<(n+2)-i-1;j++)
```

```
{
if(t[j]>t[j+1])
{
temp=t[j];
t[j]=t[j+1];
t[j+1]=temp;
}
}
}
for(i=0;i<n+2;i++)
if(t[i]==h)
{
    j=i;
k=i;
p=0;
}
while(t[j]!=0)
{
atr[p]=t[j];
j--;
p++;
}
atr[p]=t[j];
for(p=k+1;p<n+2;p++,k++)
atr[p]=t[k+1];
printf("seek sequence is:");
for(j=0;j<n+1;j++)
{
if(atr[j]>atr[j+1])
    d[j]=atr[j]-atr[j+1];
else
    d[j]=atr[j+1]-atr[j];
```

```
sum+=d[j];
printf("\n%d", atr[j]);
}
printf("\nTotal head movements:%d",sum);
}
```

TEST CASES:

guest-uivabk@student-OptiPlex-3040:~/Desktop$ ./a.out
enter the no of tracks to be traveresed8
enter the position of head50
enter the tracks176
79
34
60
92
11
41
114
seek sequence is:
50
41
34
11
0
60
79
92
114
Total head movements:226

LAB QUESTIONS &ASSIGNMENTS

PROGRAM 2

PRE-LAB QUESTIONS

1. Define operating system?
2. What are the different types of operating systems?
3. Define a process?
4. What is CPU Scheduling?
5. Define arrival time, burst time, waiting time, turnaround time?

POST-LAB QUESTIONS

6. What is the advantage of round robin CPU scheduling algorithm?
7. Which CPU scheduling algorithm is for real-time operating system?
8. In general, which CPU scheduling algorithm works with highest waiting time?
9. Is it possible to use optimal CPU scheduling algorithm in practice?
10. What is the real difficulty with the SJF CPU scheduling algorithm?

ASSIGNMENT QUESTIONS

11. Write a C program to implement round robin CPU scheduling algorithm for the following given scenario. All the processes in the system are divided into two categories – system processes and user processes. System processes are to be given higher priority than user processes. Consider the time quantum size for the system processes and user processes to be 5 msec and 2 msec respectively.
12. Write a C program to simulate pre-emptive SJF CPU scheduling algorithm.

PRE-LAB QUESTIONS

1. What is multi-level queue CPU Scheduling?
2. Differentiate between the general CPU scheduling algorithms like FCFS, SJF etc and multi-level queue CPUScheduling?
3. What are CPU-bound I/O-bound processes?

POST-LAB QUESTIONS

4. What are the parameters to be considered for designing a multilevel feedback queue scheduler?
5. Differentiate multi-level queue and multi-level feedback queue CPU scheduling algorithms?
6. What are the advantages of multi-level queue and multi-level feedback queue CPU scheduling algorithms?

ASSIGNMENT QUESTIONS

7. Write a C program to simulate multi-level queue scheduling algorithm considering the following scenario. All the processes in the system are divided into two categories – system processes and user processes. System processes are to be given higher priority than user processes. Consider each process priority to be from 1 to 3. Use priority scheduling for the processes in each queue.

PROGRAM 3

PRE-LAB QUESTIONS

1. What is the need for process synchronization?
2. Define a semaphore?
3. Define producer-consumer problem?

POST-LAB QUESTIONS

4. Discuss the consequences of considering bounded and unbounded buffers in producer-consumerproblem?
5. Can producer and consumer processes access the shared memory concurrently? If not which techniqueprovides such a benefit?

ASSIGNMENT QUESTIONS

Write a C program to simulate producer-consumer problem using message-passing system

PROGRAM 5

PRE-LAB QUESTIONS

1. Define resource. Give examples.
2. What is deadlock?
3. What are the conditions to be satisfied for the deadlock to occur?

POST-LAB QUESTIONS

4. How can be the resource allocation graph used to identify a deadlock situation?
5. How is Banker's algorithm useful over resource allocation graph technique?
6. Differentiate between deadlock avoidance and deadlock prevention?

ASSIGNMENT QUESTIONS

7. Write a C program to implement deadlock detection technique for the following scenarios?
   a. Single instance of each resource type
   b. Multiple instances of each resource type

PROGRAM 6

PRE-LAB QUESTIONS

1. What are the advantages of noncontiguous memory allocation schemes?
2. What is the process of mapping a logical address to physical address with respect to the paging memorymanagement technique?
3. Define the terms – base address, offset?

POST-LAB QUESTIONS

4. Differentiate between paging and segmentation memory allocation techniques?
5. What is the purpose of page table?
6. Whether the paging memory management technique suffers with internal or external fragmentationproblem. Why?
7. What is the effect of paging on the overall context-switching time?

ASSIGNMENT QUESTIONS

8. Write a C program to simulate two-level paging technique.

PROGRAM 7

PRE-LAB QUESTIONS

1. Define the concept of virtual memory?
2. What is the purpose of page replacement?
3. Define the general process of page replacement?
4. List out the various page replacement techniques?
5. What is page fault?

POST-LAB QUESTIONS

6. Which page replacement algorithm suffers with the problem of Belady's anomaly?
7. Define the concept of thrashing? What is the scenario that leads to the situation of thrashing?

ASSIGNMENT QUESTIONS

8. Write a C program to simulate LRU-approximation page replacement algorithm?
   a. Additional-Reference bits algorithm
   b. Second-chance algorithm

PROGRAM 8

PRE-LAB QUESTIONS

1. Define directory?
2. Describe the general directory structure?
3. List the different types of directory structures?

POST-LAB QUESTIONS

4. Which of the directory structures is efficient? Why?
5. Which directory structure does not provide user-level isolation and protection?
6. What is the advantage of hierarchical directory structure?

ASSI7GNMENT QUESTIONS

7. Write a C to simulate acyclic graph directory structure?
8. Write a C to simulate general graph directory structure?

PROGRAM 9

PRE-LAB QUESTIONS

1. Define file?
2. What are the different kinds of files?
3. What is the purpose of file allocation strategies?

POST-LAB QUESTIONS

4. Identify ideal scenarios where sequential, indexed and linked file allocation strategiesare most appropriate?
5. What are the disadvantages of sequential file allocation strategy?
6. What is an index block?
7. What is the file allocation strategy used in UNIX?

ASSIGNMENT QUESTIONS

8. Write a C program to simulate a two-level index scheme for file allocation?

PROGRAM 10

PRE-LAB QUESTIONS

1. What is disk scheduling?
2. List the different disk scheduling algorithms?
3. Define the terms – disk seek time, disk access time and rotational latency?

POST-LAB QUESTIONS

4. What is the advantage of C-SCAN algorithm over SCAN algorithm?
5. Which disk scheduling algorithm has highest rotational latency? Why?

ASSIGNMENT QUESTIONS

Write a C program to implement SSTF disk scheduling algorithm