VIMAL JYOTHI
ENGINEERING COLLEGE
Affiliated to APJ Abdul Kalam Technological University &
Kannur University | Approved by AICTE
Under the Archdiocese of Thalassery

LAB MANUAL

CSL204 –OPERATING SYSTEMS LAB

Name: .....
PRN.....
Registration No.....

Certified that this is the bonafide record of the work done in 'Operating Systems Lab ' of Semester IV, B Tech- Computer Science and Engineering
under the
Department of CSE , Vimal Jyothi Engineering College – Chemperi, Kannur
affiliated to
APJ Abdul Kalam Technological University
by
Mr./ Ms. .....

Place : **Staff in-charge** **HoD**
Date :

DEPARTMENT OF
COMPUTER SCIENCE AND ENGINEERING

VISION OF THE DEPARTMENT

To contribute to the society through excellence in scientific and knowledge-based education utilizing the potential of computer science and engineering with a deep passion for wisdom, culture and values.

MISSION OF THE DEPARTMENT

- To promote all-round growth of an individual by creating futuristic environment that fosters critical thinking, dynamism and innovation to transform them into globally competitive professionals.

- To undertake collaborative projects which offer opportunities for long-term interaction with academia and industry.

- To develop human potential to its fullest extent so that intellectually capable and optimistic leaders can emerge in a range of professions.

PROGRAM EDUCATIONAL OBJECTIVES (PEOs)

I. Graduates will achieve broad and in-depth knowledge of Computer Science and Engineering relating to industrial practices and research to analyze the practical problems and think creatively to generate innovative solutions using appropriate technologies.

II. Graduates will make valid judgment, synthesize information from a range of sources and communicate them in sound ways appropriate to their discipline.

III. Graduates will sustain intellectual curiosity and pursue life-long learning not only in areas that are relevant to Computer Science and Engineering, but also that are important to society.

IV. Graduates will adapt to different roles and demonstrate leaderships in global working environment by respecting diversity, professionalism and ethical practices.

PROGRAMME OUTCOMES (POs)

Engineering Graduates will be able to:

1. **Engineering Knowledge**: Apply the knowledge of mathematics, science, engineering Fundamentals, and an engineering specialization to the solution of complex engineering problems.

2. **Problem Analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

3. **Design/ Development of Solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

4. **Conduct Investigations of Complex Problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

5. **Modern Tool Usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

6. **The Engineer and Society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

7. **Environment and Sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9. **Individual and Team Work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

11. **Project Management and Finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

12. Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.-A4

PROGRAM SPECIFIC OUTCOMES (PSOs)

1. Apply the knowledge of electrical fundamentals, circuit design, control engineering, analog & digital electronics to the field of electrical & electronics systems in industry.

2. Develop technical knowledge, skill, and competence to identify comprehend and solve problems in research and academic related to power system engineering, industrial drives & control.

SYLLABUS

Course No.	Course Name	L-T-P Credits	Year of Introduction
CSL204	OPERATING SYSTEMS LAB	0-0-3	2019

List of Experiments:* mandatory

1. Basic Linux commands
2. Shell programming -Command syntax -Write simple functions with basic tests, loops, patterns
3. System calls of Linux operating system:* fork, exec, getpid, exit, wait, close, stat, opendir, readdir
4. Write programs using the I/O system calls of Linux operating system (open, read, write)
5. Implement programs for Inter Process Communication using Shared Memory *
6. Implement Semaphores*
7. Implementation of CPU scheduling algorithms.
    a. Round Robin
    b. SJF
    c. FCFS
    d. Priority *
8. Implementation of the Memory Allocation Methods for fixed partition*
    a. First Fit
    b. Worst Fit
    c. Best Fit
9. Implement l page replacement algorithms
    a. FIFO
    b. LRU
    c. LFU*
10. Implement the banker's algorithm for deadlock avoidance. *
11. Implementation of Deadlock detection algorithm
12. Simulate file allocation strategies. b) Sequential b) Indexed c) Linked
13. Simulate disk scheduling algorithms. *
    a. FCFS
    b. SCAN
    c. C-SCAN

Course Outcomes:

At the end of the course, the student should be able to

CO1	Illustrate the use of systems calls in Operating Systems. (Cognitive knowledge: Understand)
CO2	Implement Process Creation and Inter Process Communication in Operating Systems. (Cognitive knowledge: Apply)
CO3	Implement First Come First Served, Shortest Job First, Round Robin and Prioritybased CPU Scheduling Algorithms. (Cognitive knowledge: Apply)
CO4	Illustrate the performance of First In First Out, Least Recently Used and Least Frequently Used Page Replacement Algorithms. (Cognitive knowledge: Apply)
CO5	Implement modules for Deadlock Detection and Deadlock Avoidance in Operating Systems. (Cognitive knowledge: Apply) CO6 Implement modules for Storage Management and Disk Scheduling in Operating Systems. (Cognitive knowledge: Apply)

LIST OF EXPERIMENTS

1. Basic Linux commands
2. Shell programming -Command syntax -Write simple functions with basic tests, loops, patterns
3. System calls of Linux operating system:* fork, exec, getpid, exit, wait, close, stat, opendir, readdir
4. Write programs using the I/O system calls of Linux operating system (open, read, write)
5. Implement programs for Inter Process Communication using Shared Memory *
6. Implement Semaphores*
7. Implementation of CPU scheduling algorithms.
    a. Round Robin
    b. SJF
    c. FCFS
    d. Priority *
8. Implementation of the Memory Allocation Methods for fixed partition*
    a. First Fit
    b. Worst Fit
    c. Best Fit
9. Implement l page replacement algorithms
    a. FIFO
    b. LRU
    c. LFU*
10. Implement the banker's algorithm for deadlock avoidance. *
11. Simulate disk scheduling algorithms. *
    a. FCFS
    b. SCAN
    c. C-SCAN

INDEX					
Sl #	**Experiment**	**Date**	**Page#**	**Mark**	**Remarks**
1	Basic Linux commands				
2	Shell programming -Command syntax -Write simple functions with basic tests, loops, patterns				
3	System calls of Linux operating system:* fork, exec, getpid, exit, wait, close, stat, opendir, readdir				
4	Write programs using the I/O system calls of Linux operating system (open, read, write)				
5	Implement programs for Inter Process Communication using Shared Memory *				
6	Implement Semaphores*				
7	Implementation of CPU scheduling algorithms. a. Round Robin b. SJF c. FCFS d. Priority *				
8	Implementation of the Memory Allocation Methods for fixed partition* a. First Fit b. Worst Fit c. Best Fit				
9	Implement l page replacement algorithms a. FIFO b. LRU c. LFU*				
10	Implement the banker's algorithm for deadlock avoidance. *				

11	Simulate disk scheduling algorithms. * a. FCFS b. SCAN c. C-SCAN				

Program #: 1
Date:

Linux commands

AIM

Write basic linux commands.

PROGRAM

File Commands		
1.	ls	Directory listing
2.	ls -al	Formatted listing with hidden files
3.	ls -lt	Sorting the Formatted listing by time modification
4.	cd dir	Change directory to dir
5.	cd	Change to home directory
6.	pwd	Show current working directory
7.	mkdir dir	Creating a directory dir
8.	cat >file	Places the standard input into the file
9.	more file	Output the contents of the file
10.	head file	Output the first 10 lines of the file
11.	tail file	Output the last 10 lines of the file
12.	tail -f file	Output the contents of file as it grows,starting with the last 10 lines
13.	touch file	Create or update file
14.	rm file	Deleting the file
15.	rm -r dir	Deleting the directory
16.	rm -f file	Force to remove the file
17.	rm -rf dir	Force to remove the directory dir
18.	cp file1 file2	Copy the contents of file1 to file2
19.	cp -r dir1 dir2	Copy dir1 to dir2;create dir2 if not present
20.	mv file1 file2	Rename or move file1 to file2,if file2 is an existing directory

RESULT

Output is successfully obtained and verified.

Program #: 2
Date:

SHELL PROGRAMMING

OBJECTIVE

Write a program to implement Shell programming -Command syntax -Write simple functions with basic tests, loops, patterns

PROGRAM

test.sh

```
$ type [
[ is a shell builtin
$ which [
/usr/bin/[
$ ls -l /usr/bin/[
lrwxrwxrwx 1 root root 4 Mar 27 2000 /usr/bin/[ -> test
$ ls -l /usr/bin/test
-rwxr-xr-x 1 root root 35368 Mar 27  2000 /usr/bin/test
```

while.sh

```
#!/bin/sh
INPUT_STRING=hello
while [ "$INPUT_STRING" != "bye" ]
do
  echo "Please type something in (bye to quit)"
  read INPUT_STRING
  echo "You typed: $INPUT_STRING"
done
```

for.sh

```
#!/bin/sh
for i in 1 2 3 4 5
do
  echo "Looping ... number $i"
done
```

pattern

```
# Static input for N
N=5

# variable used for
# while loop
i=0
j=0

while [ $i -le `expr $N - 1` ]
do
    j=0
```

```
    while [ $j -le `expr $N - 1` ]
    do
       if [ `expr $N - 1` -le `expr $i + $j` ]
       then
         # Print the pattern
         echo -ne "#"
       else
         # Print the spaces required
         echo -ne " "
       fi
       j=`expr $j + 1`
    done
    # For next line
    echo

    i=`expr $i + 1`
done
```

OUTPUT

```
     #
    ##
   ###
  ####
#####
```

RESULT

Output is successfully obtained and verified.

Program #: 3
Date:

SYSTEM CALLS

OBJECTIVE

Write a program to simulate System calls of Linux operating system:* fork, exec, getpid, exit, wait, close, stat, opendir, readdir

PROGRAM

getpid and fork

```
//
 process1.c
#include<stdio.h>
void main()
{
int pid1,pid2,pid3;
 printf("Parent id is %d and root id is
%d\n",getpid(),getppid());
 pid1=fork();
if(pid1==0)
{
 printf("Process 1 id is %d and its parent id is %d\n",getpid(),getppid());
 pid2=fork();
}
if(pid2==0)
{
 printf("Process 2 id is %d and its parent id is %d\n",getpid(),getppid());
 pid3=fork();
}
if(pid1==0&&pid2==0&&pid3==0)
```

```
{
 printf("Process 3 id is %d and its parent id is %d\n",getpid(),getppid());
}
}
```

Output

```
$ cc process1.c
$ a.out
Parent id is 3553 and root id is 2495
Process 1 id is 3554 and its parent id is 3553
Process 2 id is 3555 and its parent id is 3554
Process 3 id is 3556 and its parent id is 3555
```

exec

```
#include<stdio.h>
#include<sys/types.h>
#include<unistd.h>
void main()
{
int pid1;
 pid1=fork();
if(pid1==0)
{
 printf("Process id is %d ",getpid());
 printf("and its parent id is %d",getppid());
execl("/bin/who","who",0);
}
}
```

Output

```
$ cc program2.c
$ a.out
Process id is 3553 and parent id is 2495
Root ttyp2 jun25 03.30
sit ttyp1 jun25  03.30
```

Opendir and Readdir

```
#include<stdio.h>
#include<dirent.h>
main()
{
DIR *p;
struct dirent *dp;
 p=opendir("."); //p=opendir("./shantha");
if(p==NULL)
{
 perror("opendir");
exit(0);
}
dp=readdir(p);
while(p!=NULL)
{
```

```
 printf("%d%s\n",dp->d_ino,dp->d_name);
dp=readdir(p);
}
}
```

Output:

```
"di.c" [New] 21L, 239C written
[test1@localhost test1]$ cc di.c
[test1@localhost test1]$ ./a.out
1049278.
442373..
1049279.kde
1049364.aa.c.swp
1049285.balan.sh.swp
```

RESULT

Output is successfully obtained and verified.

Program #: 4
Date:

IO SYSTEM CALLS

OBJECTIVE

Write programs using the I/O system calls of Linux operating system (open, read, write)

PROGRAM

```
#include<unistd.h>

#include<fcntl.h>

#include<sys/stat.h>

#include<sys/types.h>

#include<stdio.h>


int main()

{

        int n,fd;

        char buff[50];  // declaring buffer
```

```
        //message printing on the display
        printf("Enter text to write in the file:\n");
        //read from keyboard, specifying 0 as fd for std input device
        //Here, n stores the number of characters
        n= read(0, buff, 50);

        // creating a new file using open.
        fd=open("file",O_CREAT | O_RDWR, 0777);

        //writting input data to file (fd)
        write(fd, buff, n);
        //Write to display (1 is standard fd for output device)
        write(1, buff, n);

        //closing the file
        int close(int fd);

        return 0;
}
```

OUTPUT

```
Enter text to write in the file:
Hello world, welcome @ IncludeHelp
Hello world, welcome @ IncludeHelp
```

RESULT

Output is successfully obtained and verified.

Program #: 5
Date:

INTER PROCESS COMMUNICATION USING SHARED MEMORY

OBJECTIVE

Write a program to implement for Inter Process Communication using Shared Memory.

PROGRAM

Writing into Shared Memory – File: shm_write.c

```
#include<stdio.h>
#include<sys/ipc.h>
#include<sys/shm.h>
#include<sys/types.h>
#include<string.h>
#include<errno.h>
#include<stdlib.h>
#include<unistd.h>
#include<string.h>

#define BUF_SIZE 1024
#define SHM_KEY 0x1234

struct shmseg {
  int cnt;
  int complete;
  char buf[BUF_SIZE];
};
int fill_buffer(char * bufptr, int size);

int main(int argc, char *argv[]) {
  int shmid, numtimes;
  struct shmseg *shmp;
  char *bufptr;
  int spaceavailable;
  shmid = shmget(SHM_KEY, sizeof(struct shmseg), 0644|IPC_CREAT);
  if (shmid == -1) {
    perror("Shared memory");
    return 1;
  }

  // Attach to the segment to get a pointer to it.
```

```
  shmp = shmat(shmid, NULL, 0);
  if (shmp == (void *) -1) {
    perror("Shared memory attach");
    return 1;
  }

  /* Transfer blocks of data from buffer to shared memory */
  bufptr = shmp->buf;
  spaceavailable = BUF_SIZE;
  for (numtimes = 0; numtimes < 5; numtimes++) {
    shmp->cnt = fill_buffer(bufptr, spaceavailable);
    shmp->complete = 0;
    printf("Writing Process: Shared Memory Write: Wrote %d bytes\n", shmp->cnt);
    bufptr = shmp->buf;
    spaceavailable = BUF_SIZE;
    sleep(3);
  }
  printf("Writing Process: Wrote %d times\n", numtimes);
  shmp->complete = 1;

  if (shmdt(shmp) == -1) {
    perror("shmdt");
    return 1;
  }

  if (shmctl(shmid, IPC_RMID, 0) == -1) {
    perror("shmctl");
    return 1;
  }
  printf("Writing Process: Complete\n");
  return 0;
}

int fill_buffer(char * bufptr, int size) {
  static char ch = 'A';
  int filled_count;

  //printf("size is %d\n", size);
  memset(bufptr, ch, size - 1);
  bufptr[size-1] = '\0';
  if (ch > 122)
  ch = 65;
  if ( (ch >= 65) && (ch <= 122) ) {
    if ( (ch >= 91) && (ch <= 96) ) {
```

```
        ch = 65;
      }
    }
    filled_count = strlen(bufptr);

    //printf("buffer count is: %d\n", filled_count);
    //printf("buffer filled is:%s\n", bufptr);
    ch++;
    return filled_count;
}
```

OUTPUT

```
Writing Process: Shared Memory Write: Wrote 1023 bytes
Writing Process: Shared Memory Write: Wrote 1023 bytes
Writing Process: Shared Memory Write: Wrote 1023 bytes
Writing Process: Shared Memory Write: Wrote 1023 bytes
Writing Process: Shared Memory Write: Wrote 1023 bytes
Writing Process: Wrote 5 times
Writing Process: Complete
```

Reading from the Shared Memory and writing to the standard output – File: shm_read.c

```
#include<stdio.h>
#include<sys/ipc.h>
#include<sys/shm.h>
#include<sys/types.h>
#include<string.h>
#include<errno.h>
#include<stdlib.h>

#define BUF_SIZE 1024
#define SHM_KEY 0x1234

struct shmseg {
   int cnt;
   int complete;
   char buf[BUF_SIZE];
};

int main(int argc, char *argv[]) {
   int shmid;
   struct shmseg *shmp;
   shmid = shmget(SHM_KEY, sizeof(struct shmseg), 0644|IPC_CREAT);
   if (shmid == -1) {
      perror("Shared memory");
      return 1;
```

```
  }

  // Attach to the segment to get a pointer to it.
  shmp = shmat(shmid, NULL, 0);
  if (shmp == (void *) -1) {
    perror("Shared memory attach");
    return 1;
  }

  /* Transfer blocks of data from shared memory to stdout*/
  while (shmp->complete != 1) {
    printf("segment contains : \n\"%s\"\n", shmp->buf);
    if (shmp->cnt == -1) {
      perror("read");
      return 1;
    }
    printf("Reading Process: Shared Memory: Read %d bytes\n", shmp->cnt);
    sleep(3);
  }
  printf("Reading Process: Reading Done, Detaching Shared Memory\n");
  if (shmdt(shmp) == -1) {
    perror("shmdt");
    return 1;
  }
  printf("Reading Process: Complete\n");
  return 0;
}
```

RESULT

Output is successfully obtained and verified.

Program #: 6
Date:

SEMAPHORE

OBJECTIVE

Write a program to Implement Semaphores*

PROGRAM

```
#include<stdio.h>
#include<pthread.h>
#include<semaphore.h>
int buf[5],f,r;
sem_t mutex,full,empty;
void *produce(void *arg)
{
   int i;
   for(i=0;i<10;i++)
   {
      sem_wait(&empty);
      sem_wait(&mutex);
      printf("produced item is %d\n",i);
      buf[(++r)%5]=i;
      sleep(1);
      sem_post(&mutex);
      sem_post(&full);
   }
}
void *consume(void *arg)
{
      int item,i;
      for(i=0;i<10;i++)
      {
            sem_wait(&full);
            sem_wait(&mutex);
            item=buf[(++f)%5];
            printf("consumed item is %d\n",item);
            sleep(1);
            sem_post(&mutex);
            sem_post(&empty);
      }
}
main()
{
   pthread_t tid1,tid2;
   sem_init(&mutex,0,1);
   sem_init(&full,0,1);
   sem_init(&empty,0,5);
   pthread_create(&tid1,NULL,produce,NULL);
   pthread_create(&tid2,NULL,consume,NULL);
   pthread_join(tid1,NULL);
   pthread_join(tid2,NULL);
}
/*
OUTPUT:
produced item is 0
consumed item is 0
```

```
produced item is 1
consumed item is 0
produced item is 2
consumed item is 0
consumed item is 0
produced item is 3
produced item is 4
consumed item is 0
produced item is 5
consumed item is 0
produced item is 6
consumed item is 1
produced item is 7
consumed item is 2
produced item is 8
consumed item is 3
produced item is 9
consumed item is 4
*/
```

RESULT

Output is successfully obtained and verified.

Program #: 7
Date:

CPU SCHEDULING

OBJECTIVE

Write a program to implement CPU scheduling algorithms.

d. Round Robin
e. SJF
f. FCFS
g. Priority *

PROGRAM

FCFS CPU SCHEDULING

```
#include<stdio.h>
int main()
{
 int n,i,j;
 //Read the number of processes
 printf("Enter the number of process : ");
 scanf("%d",&n);
 int a[n],bt[n],tat[n],wt[n],ct[n];
 //Read the burst time of the processes
 printf("Enter the Burst time of process : ");
 for(i=0;i<n;i++)
  scanf("%d",&bt[i]);
 //Calculate process number and completion time of first process
 for(i=0;i<n;i++)
  a[i]=i+1;
  ct[0]=bt[0];
 //Calculate completion time
 for(i=1;i<n;i++)
  ct[i]=ct[i-1]+bt[i];
 //Calculate turn around time
 for(i=0;i<n;i++)
  tat[i]=ct[i];
 //Calculate wait time
 for(i=0,j=0;i<n;i++,j++)
  wt[i]=tat[i]-bt[j];
 //Display
 printf("\nProcess\tBT\tCT\tTAT\tWT");
 for(i=0;i<n;i++)
 {
  printf("\nP%d\t%d\t%d\t%d\t%d",a[i],bt[i],ct[i],tat[i],wt[i]);
 }
 printf("\n");
 return 0;
}
/*
OUTPUT:
Enter the number of process : 4
Enter the Burst time of process : 3 4 2 1
```

```
Process BT        CT TAT   WT
 P1     3 3          3           0
 P2     4 7   7      3
 P3     2 9   9      7
 P4     1 10       10         9
*/
```

SJF CPU SCHEDULING

```
#include<stdio.h>
int main()
{
 int n,i,temp,j,k=0;
 //Read the number of processes
 printf("Enter the number of process : ");
 scanf("%d",&n);
 int bt[n],tat[n],wt[n],a[n],ct[n];
 //Read the burst time of the processes
 printf("Enter the Burst time of process : ");
 for(i=0;i<n;i++)
 {
  scanf("%d",&bt[i]);
 }
 //Calculate process number
 for(i=0;i<n;i++)
  a[i]=i+1;
 //Sort burst time and process number
 for(i=0;i<n;i++)
 {
  for(j=i+1;j<n;j++)
  {
   if(bt[i]>bt[j])
    {
     temp=bt[i];
     bt[i]=bt[j];
     bt[j]=temp;
     temp=a[i];
     a[i]=a[j];
     a[j]=temp;
    }
  }
 }
 //Assign completion time of first process
 ct[0]=bt[0];
 //Calculate completion time
 for(i=1;i<n;i++)
  ct[i]=ct[i-1]+bt[i];
 //Calculate turn around time
 for(i=0;i<n;i++)
  tat[i]=ct[i];
 //Calculate wait time
 for(i=0,j=0;i<n;i++,j++)
  wt[i]=tat[i]-bt[j];
 printf("\nProcess\tBT\tCT\tTAT\tWT");
 for(i=0;i<n;i++)
 {
  printf("\nP%d\t%d\t%d\t%d\t%d",a[i],bt[i],ct[i],tat[i],wt[i]);
 }
 printf("\n");
```

```
 return 0;
}
/*
OUTPUT:
Enter the number of process : 4
Enter the Burst time of process : 3 4 2 1
Process BT  CT  TAT  WT
  P4      1   1       1       0
  P3    2   3         3       1
  P1    3   6         6       3
  P2    4   10       10       6
*/
```

ROUND ROBIN CPU SCHEDULING

```
#include<stdio.h>
int main()
{
 int n,x=0,i,j,t;
 //Read the number of processes
 printf("Enter the number of process : ");
 scanf("%d",&n);
 int bta[n],bt[n],tat[n],wt[n],ct[n],a[n];
 //Read the burst time of the processes
 printf("Enter the Burst time of process : ");
 for(i=0;i<n;i++)
  scanf("%d",&bt[i]);
 //Read the time slice
 printf("Enter the Time slice : ");
  scanf("%d",&t);
 //Calculate process number
 for(i=0;i<n;i++)
  a[i]=i+1;
 //Initialize completion time to 0
 for(i=0;i<n;i++)
  ct[0]=0;
 //Duplicate burst time
 for(i=0;i<n;i++)
  bta[i]=bt[i];
 //Computation
 for(j=0;j<n;j++)
 {
  for(i=0;i<n;i++)
  {
   if(bta[i]>t) //Burst time is more than time slice
   {
    bta[i]=bta[i]-t;
    x=x+t;
    ct[i]=x;
   }
   else if(bta[i]==0)
    continue;
   else if(bta[i]<=t)
   {
    x=x+bta[i];
    ct[i]=x;
    bta[i]=0;
   }
  }
```

```
 }
 //Completion time is TAT
 for(i=0;i<n;i++)
  tat[i]=ct[i];
 //Calculate Waiting time
 for(i=0,j=0;i<n;i++,j++)
  wt[i]=tat[i]-bt[j];
 //Display
 printf("\nProcess\tBT\tCT\tTAT\tWT");
 for(i=0;i<n;i++)
 {
  printf("\nP%d\t%d\t%d\t%d\t%d",a[i],bt[i],ct[i],tat[i],wt[i]);
 }
 printf("\n");
 return 0;
}
/*
OUTPUT:
Enter the number of process : 4
Enter the Burst time of process : 3 4 2 1
Enter the Time slice : 2
Process  BT        CT  TAT  WT
  P1          3         8   8       5
  P2     4 10          10   6
  P3          2         6          6          4
  P4          1         7          7          6
*/
```

PRIORITY CPU SCHEDULING

```
#include<stdio.h>
int main()
{
 int n,i,temp,j,k=0;
 //Read the number of processes
 printf("Enter the number of process : ");
 scanf("%d",&n);
 int a[n],bt[n],pt[n],tat[n],wt[n],pta[n],ct[n];
 //Read the burst time of the processes
 printf("Enter the Burst time of process : ");
 for(i=0;i<n;i++)
  scanf("%d",&bt[i]);
 //Read the priority of each process
 printf("Enter the Priority of process : ");
 for(i=0;i<n;i++)
  scanf("%d",&pt[i]);
 //Calculate the process number
 for(i=0;i<n;i++)
  a[i]=i+1;
 //sort priority,burst time and process number
 for(i=0;i<n;i++)
 {
  for(j=i+1;j<n;j++)
  {
   if(pt[i]>pt[j])
    {
     temp=pt[i];
     pt[i]=pt[j];
     pt[j]=temp;
```

```
    temp=bt[i];
    bt[i]=bt[j];
    bt[j]=temp;
    temp=a[i];
    a[i]=a[j];
    a[j]=temp;
   }
  }
 }
 //Assign completion time of first process
 ct[0]=bt[0];
 //Calculate completion time
 for(i=1;i<n;i++)
  ct[i]=ct[i-1]+bt[i];
 //Calculate turn around time
 for(i=0;i<n;i++)
  tat[i]=ct[i];
 //Calculate wait time
 for(i=0,j=0;i<n;i++,j++)
  wt[i]=tat[i]-bt[j];
 printf("\nProcess\tBT\tCT\tTAT\tWT");
 for(i=0;i<n;i++)
 {
  printf("\nP%d\t%d\t%d\t%d\t%d",a[i],bt[i],ct[i],tat[i],wt[i]);
 }
 printf("\n");
 return 0;
}
/*
OUTPUT:
Enter the number of process : 4
Enter the Burst time of process : 3 4 2 1
Enter the Priority of process : 4 2 1 3
Process BT  CT  TAT WT
  P3      2       2       2       0
  P2      4       6       6       2
  P4      1       7       7       6
  P1      3       10      10      7
*/
```

RESULT

Output is successfully obtained and verified.

Program #: 8
Date:

MEMORY ALLOCATION METHODS

OBJECTIVE

Write a program to Implement of the Memory Allocation Methods for fixed partition*

h. First Fit
i. Worst Fit
j. Best Fit

PROGRAM

a)WORST-FIT

```
#include<stdio.h>
#include<conio.h>
#define max 25
void main()
{
int frag[max],b[max],f[max],i,j,nb,nf,temp;
static int bf[max],ff[max];
clrscr();
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]],frag[i]);
getch();
}
```

INPUT

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

OUTPUT

File No	File Size	Block No	Block Size	Fragment
1	1	1	5	4
2	4	3	7	3

b) Best-fit

```
#include<stdio.h>
#include<conio.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];
clrscr();
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]],frag[i]);
getch();
}
```

INPUT
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

OUTPUT

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

c) First-fit

```
#include<stdio.h>
#include<conio.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];
clrscr();
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]],frag[i]);
getch();
}
```

INPUT

```
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
```

OUTPUT

File No	File Size	Block No	Block Size	Fragment
1	1	3	7	6
2	4	1	5	1

RESULT

Output is successfully obtained and verified.

Program #:9
Date:

PAGE REPLACEMENT ALGORITHM

OBJECTIVE

Write a program to implement Implement l page replacement algorithms

a. FIFO
b. LRU
c. LFU*

PROGRAM

FIFO PAGE REPLACEMENT TECHNIQUE

```
#include<stdio.h>
int main()
{
 int n,m,i,j,k=0,pf=0,f=0;
 //Read the number of pages
 printf("Enter the number of pages : ");
 scanf("%d",&n);
 //Read the size of the cache
 printf("Enter the size of cache memory : ");
 scanf("%d",&m);
 int a[n],b[m];
 //Read page number
 printf("Enter the page number of each page : ");
 for(i=0;i<n;i++)
  scanf("%d",&a[i]);
 for(i=0;i<n;i++)
 {
  for(j=0;j<m;j++)
  {
   if(a[i]==b[j])
    {
     f=0;
     break;
    }
   else
    {
     f=1;
     continue;
    }
  }
 if(k>=m)
  k = 0;
 if(f==1)
  {
    b[k]=a[i];
    k++;
    pf++;
    continue;
  }
 }
```

```
 for(i=0;i<m;i++)
  printf("%d\t",b[i]);
 printf("\nNumber of page faults : ");
 printf("%d",pf);
 printf("\n");
 return 0;
}
/*
OUTPUT:
Case 1:
Enter the number of pages : 4
Enter the size of cache memory : 3
Enter the page number of each page : 3 3 1 2
3           1           2
Number of page faults : 3
Case 2:
Enter the number of pages : 4
Enter the size of cache memory : 3
Enter the page number of each page : 1 2 3 4
4           2           3
Number of page faults : 4
Case 3:
Enter the number of pages : 3
Enter the size of cache memory : 3
Enter the page number of each page : 1 2 2
1           2           0
Number of page faults : 2
*/
```

LRU PAGE REPLACEMENT TECHNIQUE

```
#include<stdio.h>
int main()
{
 int index,s,n,m,i,j,k,pf=0,f=0;
 //Read the number of pages
 printf("Enter the number of pages : ");
 scanf("%d",&n);
 //Read the size of the cache
 printf("Enter the size of cache memory : ");
 scanf("%d",&m);
 int a[n],b[m],c[m];
 //Read page number
 printf("Enter the page number of each page : ");
 for(i=0;i<n;i++)
  scanf("%d",&a[i]);
 //Initialization
 for(i=0;i<m;i++)
  {
   c[i]=0;
   b[i]=0;
  }
 //Computation
 for(i=0;i<n;i++)
 {
  f=1; //Only when page number is not in cache
  for(j=0;j<m;j++)
  {
   if(a[i]==b[j])
```

```
   {
    f=0; //Only when page number is in cache
    c[j]=i+1;
    break;
   }
  }
  if(f==1)
  {
   s=c[0];
   index=0;
   for(k=0;k<m;k++)
   {
    if(c[k]<s)
    {
     s=c[k];
     index=k;
    }
   }
   b[index]=a[i];
   c[index]=i+1;
   pf++;
  }
 }
 for(i=0;i<m;i++)
  printf("%d\t",b[i]);
 printf("\nNumber of page faults : ");
 printf("%d",pf);
 printf("\n");
 return 0;
}
/*
OUTPUT:
Case 1:
Enter the number of pages : 6
Enter the size of cache memory : 3
Enter the page number of each page : 1 2 2 1 3 4
1       4       3
Number of page faults : 4
Case 2:
Enter the number of pages : 4
Enter the size of cache memory : 3
Enter the page number of each page : 3 3 1 2
3       1       2
Number of page faults : 3
*/
```

LFU PAGE REPLACEMENT TECHNIQUE

```
#include<stdio.h>
int main()
{
 int s,n,m,i,j,k=0,l,pf=0,f=0,index;
 //Read the number of pages
 printf("Enter the number of pages : ");
 scanf("%d",&n);
 //Read the size of the cache
 printf("Enter the size of cache memory : ");
 scanf("%d",&m);
 int a[n],b[m],c[m];
 //Read page number
 printf("Enter the page number of each page : ");
 for(i=0;i<n;i++)
  scanf("%d",&a[i]);
 //Initialization
 for(i=0;i<m;i++)
 {
  c[i]=0;
  b[i]=0;
 }
 for(i=0;i<m;i++)
 {
  c[i]=1;
 }
 for(i=0;i<n;i++)
 {
  f=1; //Only when page number is not in cache
  for(j=0;j<m;j++)
  {
   if(a[i]==b[j])
   {
    f=0; //Only when page number is in cache
    c[j]=c[j]+1;
    break;
   }
  }

  if(f==1)
   {
    l=c[0];
    printf("Value of l = %d",l);
    index=0;
    for(k=0;k<m;k++)
    {
    if(c[k]<=l)
    {
      l=c[k];
      index=k;
    }
```

```
    }
    printf("Value of l = %d",l);
    b[index]=a[i];
    c[index]=1;
    pf++;
    }
    for(k=0;k<m;k++)
     printf("Value of k=%d c[k] = %d\n",k,c[k]);
   }
 for(i=0;i<m;i++)
  printf("%d\t",b[i]);
 printf("\nNumber of page faults : ");
 printf("%d",pf);
 printf("\n");
 return 0;
}
```

RESULT

Output is successfully obtained and verified.

Program #: 10
Date:

BANKERS ALGORITHM

OBJECTIVE

Write a program to implement the banker's algorithm for deadlock avoidance.

PROGRAM

```
#include<stdio.h>

int max[100][100];
int alloc[100][100];
int need[100][100];
int avail[100];
int n,r;
void input();
void show();
void cal();
int main()
{
 int i,j;
 printf("********** Banker's Algorithm ***********\n");
 input();
 show();
 cal();

 return 0;
}

void input()
{
 int i,j;
 printf("Enter the no of Processes\t");
 scanf("%d",&n);
 printf("Enter the no of resources instances\t");
 scanf("%d",&r);
 printf("Enter the Max Matrix\n");
 for(i=0;i<n;i++)
 {
  for(j=0;j<r;j++)
  {
   scanf("%d",&max[i][j]);
  }
 }
 printf("Enter the Allocation Matrix\n");
 for(i=0;i<n;i++)
 {
  for(j=0;j<r;j++)
  {
   scanf("%d",&alloc[i][j]);
  }
 }
```

```
 printf("Enter the available Resources\n");
 for(j=0;j<r;j++)
 {
  scanf("%d",&avail[j]);
 }
}


void show()
{
 int i,j;
 printf("\nProcess\t Allocation\t Max\t Available\t");
 for(i=0;i<n;i++)
 {
   printf("\nP%d\t   ",i+1);
   for(j=0;j<r;j++)
   {
    printf("%d ",alloc[i][j]);
   }
   printf("\t");

   for(j=0;j<r;j++)
   {
    printf("%d ",max[i][j]);
   }
   printf("\t");


   if(i==0)
   {
    for(j=0;j<r;j++)
    printf("%d ",avail[j]);
   }
   printf("\n");
 }
}


void cal()
{
 int finish[100],temp,flag=1,k,c1=0;
 int safe[100];
 int i,j;
 for(i=0;i<n;i++)
 {
  finish[i]=0;
 }

 //find need matrix
 for(i=0;i<n;i++)
 {
  for(j=0;j<r;j++)
  {
   need[i][j]=max[i][j]-alloc[i][j];
  }
 }
 printf("\n");
```

```
while(flag)
{
 flag=0;
  for(i=0;i<n;i++)
    {
      int c=0;
      for(j=0;j<r;j++)
        {
         if((finish[i]==0)&&(need[i][j]<=avail[j]))
           {
            c++;
            if(c==r)
              {
               for(k=0;k<r;k++)
                 {
                   avail[k]+=alloc[i][j];
                   finish[i]=1;
                   flag=1;
                 }
               printf("P%d->",i+1);
               if(finish[i]==1)
                 {
                  i=n;
                 }
              }
           }
        }
    }
 }
 for(i=0;i<n;i++)
   {
    if(finish[i]==1)
      {
       c1++;
      }
  //  else
  //    {
  //      printf("P%d->",i+1);
  //    }
   }
if(c1==n)
{
 printf("\n The system is in safe state");
}
else
{
 printf("\n Process are in dead lock");
 printf("\n System is in unsafe state");
}
}
/*
OUTPUT:
Case 1:
********** Banker's Algorithm ************
Enter the no of Processes 4
Enter the no of resources instances            3
```

```
Enter the Max Matrix 5 4 3 6 5 4 7 6 5 8 7 6
Enter the Allocation Matrix 1 1 1 1 1 1 1 1 1 1 1 1
Enter the available Resources 6 5 4
Process  Allocation      Max      Available
P1          1 1 1       5 4 3    6 5 4
P2          1 1 1       6 5 4
P3          1 1 1       7 6 5
P4          1 1 1       8 7 6
P1->P2->P3->P4->
The system is in safe state
Case 2:
********** Banker's Algorithm ************
Enter the no of Processes 4
Enter the no of resources instances          3
Enter the Max Matrix 5 4 3 6 5 4 7 6 5 8 7 6
Enter the Allocation Matrix 1 1 1 1 1 1 1 1 1 1 1 1
Enter the available Resources 2 3 1
Process  Allocation      Max      Available
P1          1 1 1       5 4 3    2 3 1
P2          1 1 1       6 5 4
P3          1 1 1       7 6 5
P4          1 1 1       8 7 6
Process are in dead lock
System is in unsafe state
*/
```

RESULT

Output is obtained successfully and verified.

Program #: 11
Date:

DISK SCHEDULING ALGORITHM

OBJECTIVE

Write a program to Simulate disk scheduling algorithms. *

a. FCFS
b. SCAN
c. C-SCAN

PROGRAM

FCFS DISK SCHEDULING

```
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
int main()
```

```
{
 int n,a,i,j,r=0,av;
 printf("Enter the size of disk queue  : \n");
 scanf("%d",&n);
 int q[n];
 printf("Enter the disk queue : \n");
 for(i=0;i<n;i++)
  scanf("%d",&q[i]);
 printf("Enter the initial disk head : \n");
 scanf("%d",&a);
 r=q[0]-a;
 for(i=1;i<n;i++)
  r=r+(abs(q[i]-q[i-1]));
 printf("No. of cylinders scanned : %d",r);
 av=r/n;
 printf("\nAverage head move : %d",av);
 printf("\n");
 return 0;
}
/*
OUTPUT:
Enter the size of disk queue  :
8
Enter the disk queue :
98 183 37 122 14 124 65 67
Enter the initial disk head :
53
No. of cylinders scanned : 640
Average head move : 80
*/
```

SCAN DISK SCHEDULING

```
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
int main()
{
 int n,a,i,k,j=0,m=0,temp,x,y=0,p=0;
 float r=0,av;
 printf("Enter the size of disk queue  : \n");
 scanf("%d",&n);
 int q[n],b[n],c[n];
 printf("Enter the disk queue : \n");
 for(i=0;i<n;i++)
 {
  scanf("%d",&q[i]);
  b[i]=0;
  c[i]=0;
 }
 printf("Enter the initial disk head : \n");
 scanf("%d",&a);
```

```
printf("Enter the limit of disk : \n");
scanf("%d",&x);
for(k=0;k<n;k++)
{
 if(a<=q[k])
 {
  b[j]=q[k];
  j++;
 }
 else
 {
  c[m]=q[k];
  m++;
 }
}
for(i=0;i<n;i++)
{
 for(j=i+1;j<n;j++)
 {
  if(b[j]<b[i])
  {
   temp=b[j];
   b[j]=b[i];
   b[i]=temp;
  }
 }
}
for(i=0;i<n;i++)
{
 for(j=i+1;j<n;j++)
 {
  if(c[j]<c[i])
  {
   temp=c[j];
   c[j]=c[i];
   c[i]=temp;
  }
 }
}
for(i=0;i<n;i++)
 printf("%d ",b[i]);
printf("\n");
for(i=0;i<n;i++)
 printf("%d ",c[i]);
for(i=0;i<n;i++)
{
 if(b[i]==0)
  continue;
 else
 {
  p=i;
  //printf("\np=%d\n",p);
  y++;
  if(y==1)//To subtract initial value and next higher value
  r = r + abs(b[p]-a);
  if(i==(n-1))
  r = r + abs(x-b[p]);
```

```
   else
   r = r + abs(b[p+1]-b[p]);
   continue;
  }
 }
y=0;
  for(i=0;i<n;i++)
 {
  if(c[i]==0)
   continue;
  else
  {
   p=i;
   //printf("p=%d\n",p);
   if(c[p+1]==0)
   r = r + abs(x-c[p]);
   else
   r = r + abs(c[p+1]-c[p]);
   continue;
  }
 }
 printf("No. of cylinders scanned : %f",r);
 av=r/n;
 printf("\nAverage head move : %f",av);
 printf("\n");
 return 0;
}
/*
OUTPUT:
Enter the size of disk queue  :
8
Enter the disk queue : 98 183 37 122 14 124 65 67
Enter the initial disk head : 56
Enter the limit of disk : 183
0 0 65 67 98 122 124 183
0 0 0 0 0 0 14 37
No. of cylinders scanned : 296.000000
Average head move : 37.000000
*/
```

RESULT

Output is successfully obtained and verified.