Sri Indu
College of Engineering & Technology
UGC Autonomous Institution
Recognized under 2(f) & 12(B) of UGC Act 1956,
NAAC, Approved by AICTE &
Permanently Affiliated to JNTUH
Estd.2001

NAAC
NATIONAL ASSESSMENT AND ACCREDITATION COUNCIL

(R22CSE2226) OPERATING SYSTEMS LAB

LAB MANUAL

II Year II Semester

DEPARTMENT OF INFORMATION TECHNOLOGY

SRI INDU COLLEGE OF ENGINEERING & TECHNOLOGY
B. TECH –INFORMATION TECHNOLOGY

INSTITUTION VISION

To be a premier Institution in Engineering & Technology and Management with competency, valuesand social consciousness.

INSTITUTION MISSION

IM1 Provide high quality academic programs, training activities and research facilities.

IM2 Promote Continuous Industry-Institute Interaction for Employability, Entrepreneurship, Leadership and Research aptitude among stakeholders.

IM3 Contribute to the Economical and technological development of the region, state and nation.

DEPARTMENT VISION

To be a recognized knowledge center in the field of Information Technology with self - motivated, employable engineers to society.

DEPARTMENT MISSION

The Department has following Missions:

DM1 To offer high quality student centric education in Information Technology.

DM2 To provide a conducive environment towards innovation and skills.

DM3 To involve in activities that provide social and professional solutions.

DM4 To impart training on emerging technologies namely cloud computing and IOT with involvement of stake holders.

PROGRAM EDUCATIONAL OBJECTIVES (PEOs)

PEO1: **Higher Studies:** Graduates with an ability to apply knowledge of Basic sciences and programming skills in their career and higher education.

PEO2: **Lifelong Learning:** Graduates with an ability to adopt new technologies for ever changing IT industry needs through Self-Study, Critical thinking and Problem solving skills.

PEO3: **Professional skills:** Graduates will be ready to work in projects related to complex problems involving multi-disciplinary projects with effective analytical skills.

PEO4: **Engineering Citizenship:** Graduates with an ability to communicate well and exhibitsocial, technical and ethical responsibility in process or product.

PROGRAM OUTCOMES (POs) & PROGRAM SPECIFIC OUTCOMES (PSOs)

PO	Description
PO 1	**Engineering Knowledge:** Apply knowledge of mathematics, natural science, computing, engineering fundamentals and an engineering specialization as specified in WK1 to WK4 respectively to develop to the solution of complex engineering problems.
PO 2	**Problem Analysis:** Identify, formulate, review research literature and analyze complex engineering problems reaching substantiated conclusions with consideration for sustainable development. (WK1 to WK4)
PO 3	**Design/Development of Solutions:** Design creative solutions for complex engineering problems and design/develop systems/components/processes to meet identified needs with consideration for the public health and safety, whole-life cost, net zero carbon, culture, society and environment as required. (WK5)
PO 4	**Conduct Investigations of Complex Problems:** Conduct investigations of complex engineering problems using research-based knowledge including design of experiments, modelling, analysis & interpretation of data to provide valid conclusions. (WK8).
PO 5	**Engineering Tool Usage:** Create, select and apply appropriate techniques, resources and modern engineering & IT tools, including prediction and modelling recognizing their limitations to solve complex engineering problems. (WK2 and WK6)
PO 6	**The Engineer and The World:** Analyze and evaluate societal and environmental aspects while solving complex engineering problems for its impact on sustainability with reference to economy, health, safety, legal framework, culture and environment. (WK1, WK5, and WK7).
PO 7	**Ethics:** Apply ethical principles and commit to professional ethics, human values, diversity and inclusion; adhere to national & international laws. (WK9)
PO 8	**Individual and Collaborative Team work:** Function effectively as an individual, and as a member or leader in diverse/multi-disciplinary teams.
PO 10	**Project Management and Finance:** Apply knowledge and understanding of engineering management principles and economic decision-making and apply these to one's own work, as a member and leader in a team, and to manage projects and in multidisciplinary environments.
PO 11	**Life-Long Learning:** Recognize the need for, and have the preparation and ability for i) independent and life-long learning ii) adaptability to new and emerging technologies and iii) critical thinking in the broadest context of technological change. (WK8)
Program Specific Outcomes	
PSO 1	**Software Development:** To apply the knowledge of Software Engineering, Data Communication, Web Technology and Operating Systems for building IOT and Cloud Computing applications.
PSO 2	**Industrial Skills Ability:** Design, develop and test software systems for world-wide network of computers toprovide solutions to real world problems.
PSO 3	**Project implementation:** Analyze and recommend the appropriate IT Infrastructure required for the implementationof a project.

COURSE NAME: OPERATING SYSTEMS

LAB COURSE CODE: (R22CSE2226)

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

Course Outcomes	Statements
C22L1.1	Simulate and implement operating system concepts such as scheduling, deadlock management, file management and memory management.
C22L1.2	Able to implement C programs using Unix system calls

CO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	P10	P11	PSO1	PSO2	PSO3
C22L1.1	3	1	2	1	3	-	-	-	-	-	-	-	1	-
C22L1.2	3	3	3	3	3	-	-	-	-	-	-	-	1	-
C22L1	3	2	2.5	2	3	-	-	-	-	-	-	-	1	-

OPERATING SYSTEMS LAB MANUAL
(R22CSE2226)

II YEAR II SEMESTER

GENERAL LABORATORY INSTRUCTIONS

1. Students are advised to come to the laboratory at least 5 minutes before (to the starting time),those who come after 5 minutes will not be allowed into the lab.
2. Plan your task properly much before to the commencement, come prepared to the lab with thesynopsis / program / experiment details.
3. Student should enter into the laboratory with:
   a. Laboratory observation notes with all the details (Problem statement, Aim, Algorithm,Procedure, Program, Expected Output, etc.,) filled in for the lab session.
   b. Laboratory Record updated up to the last session experiments and other utensils (if any)needed in the lab.
   c. Proper Dress code and Identity card.
4. Sign in the laboratory login register, write the TIME-IN, and occupy the computer systemallotted to you by the faculty.
5. Execute your task in the laboratory, and record the results / output in the lab observation notebook, and get certified by the concerned faculty.
6. All the students should be polite and cooperative with the laboratory staff, must maintain thediscipline and decency in the laboratory.
7. Computer labs are established with sophisticated and high end branded systems, whichshould be utilized properly.
8. Students / Faculty must keep their mobile phones in SWITCHED OFF mode during the lab sessions. Misuse of the equipment, misbehaviors with the staff and systems etc., will attract severe punishment.
9. Students must take the permission of the faculty in case of any urgency to go out ; if anybody found loitering outside the lab / class without permission during working hours will be treated seriously and punished appropriately.
10. Students should LOG OFF/ SHUT DOWN the computer system before he/she leaves the lab after completing the task (experiment) in all aspects. He/she must ensure the system / seat is kept properly.

Head of the Department **Principal**

SRI INDU COLLEGE OF ENGINEERING & TECHNOLOGY

(An Autonomous Institution under UGC, New Delhi)

B.Tech. - II Year – II Semester

L	T	P	C
0	0	2	1

(R22CSE2226) OPERATING SYSTEMS LAB

Course Objectives:

- To provide an understanding of the design aspects of operating system concepts throughsimulation
- Introduce basic Unix commands, system call interface for process management, inter process communication and I/O in Unix

Course Outcomes:

- Simulate and implement operating system concepts such as scheduling, deadlockmanagement, file management and memory management.
- Able to implement C programs using Unix system calls

List of Experiments:

1. Write C programs to simulate the following CPU Scheduling algorithms a) FCFS b) SJF c) Round Robin d) priority

2. Write programs using the I/O system calls of UNIX/LINUX operating system (open, read, write, close, fcntl, seek, stat, opendir, readdir)

3. Write a C program to simulate Bankers Algorithm for Deadlock Avoidance and Prevention.

4. Write a C program to implement the Producer – Consumer problem using semaphores using UNIX/LINUX system calls.

5. Write C programs to illustrate the following IPC mechanisms a) Pipes b) FIFOs c) Message Queues d) Shared Memory

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

7. Write C programs to simulate Page replacement policies a) FCFS b) LRU c) Optimal

TEXT BOOKS:

8. Operating System Principles- Abraham Silberchatz, Peter B. Galvin, Greg Gagne 7th Edition,John Wiley
9. Advanced programming in the Unix environment, W.R.Stevens, Pearson education.

REFERENCE BOOKS:

1. Operating Systems – Internals and Design Principles, William Stallings, Fifth Edition–2005,Pearson Education/PHI
2. Operating System - A Design Approach-Crowley, TMH.
3. Modern Operating Systems, Andrew S Tanenbaum, 2nd edition, Pearson/PHI
4. UNIX Programming Environment, Kernighan and Pike, PHI/Pearson Education
5. UNIX Internals: The New Frontiers, U. Vahalia, Pearson Education

WEEK-1

Write C programs to simulate the following CPU Scheduling algorithms.

a) FCFS
b) SJF
c) Round Robin
d) Priority

a) FCFS (First Come First Serve)

Aim: Write a C program to implement the various process scheduling mechanisms such as FCFS scheduling.

Algorithm:

1: Start the process
2: Accept the number of processes in the ready Queue
3: For each process in the ready Q, assign the process id and accept the CPU burst time
4: Set the waiting of the first process as '0' and its burst time as its turn around time
5: for each process in the Ready Q calculate
a. Waiting time for process(n)= waiting time of process (n-1) + Burst time of process(n-1)
b. Turnaround time for Process(n)= waiting time of Process(n)+ Burst time for process(n)
6: Calculate
a. Average waiting time = Total waiting Time / Number of process
b. Average Turnaround time = Total Turnaround Time / Number of process
7: Stop the process

Program:

```
#include<stdio.h>
int main()
{
   int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,pos,temp;
   float avg_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
   }
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\t\t  %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("\n\nAverage Waiting Time=%f",avg_wt);
   printf("\nAverage Turnaround Time=%f\n",avg_tat);
}
```

Output:

```
[188r1a0501@localhost ~]$ vi w1.c
[188r1a0501@localhost ~]$ gcc  w1.c
[188r1a0501@localhost ~]$ ./a.out
Enter number of process:3

Enter Burst Time:
p1:3
p2:4
p3:2

Process     Burst Time          Waiting Time    Turnaround Time
p1                3                 0                 3
p2                4                 3                 7
p3                2                 7                 9

Average Waiting Time=3.333333
Average Turnaround Time=6.333333
[188r1a0501@localhost ~]$
```

b) SJF (Shortest Job First)

Aim: Write a C program to implement the various process scheduling mechanisms such as SJF Scheduling.

Algorithm:

1: Start the process

2: Accept the number of processes in the ready Queue

3: For each process in the ready Q, assign the process id and accept the CPU burst time 4: Start the Ready Q according the shortest Burst time by sorting according to lowest to highest burst time.

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

6: For each process in the ready queue, calculate

(a) Waiting time for process(n)= waiting time of process (n-1) + Burst time of process(n-1)
(b) Turn around time for Process(n)= waiting time of Process(n)+ Burst time for process(n)
7: Calculate

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

• Average Turnaround time = Total Turnaround Time / Number of process
8: Stop the process

Program:

```
#include<stdio.h>
int main()
{
   int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,pos,temp;
   float avg_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\t\t  %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("\n\nAverage Waiting Time=%f",avg_wt);
  printf("\nAverage Turnaround Time=%f\n",avg_tat);
}
```

Output:

```
[188r1a0501@localhost ~]$ vi w1a.c
[188r1a0501@localhost ~]$ gcc  w1a.c
[188r1a0501@localhost ~]$ ./a.out
Enter number of process:3

Enter Burst Time:
p1:5
p2:3
p3:7

Process     Burst Time          Waiting Time    Turnaround Time
p2                3                 0                 3
p1                5                 3                 8
p3                7                 8                 15

Average Waiting Time=3.666667
Average Turnaround Time=8.666667
[188r1a0501@localhost ~]$
```

c) Round Robin

Aim: Write a C program to implement the various process scheduling mechanisms such as Round Robin Scheduling.

Algorithm

1: Start the process

2: Accept the number of processes in the ready Queue and time quantum (or) time slice

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

4: Calculate the no. of time slices for each process where

No. of time slice for process(n) = burst time process(n)/time slice

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

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

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

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

7: Calculate

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

(b) Average Turnaround time = Total Turnaround Time / Number of process Step 8: Stop the process

Program:

```
#include<stdio.h>
main()
{
    int st[10],bt[10],wt[10],tat[10],n,tq;
    int i,count=0,swt=0,stat=0,temp,sq=0;
    float awt,atat;
    printf("enter the number of processes");
    scanf("%d",&n);
    printf("enter the burst time of each process /n");
    for(i=0;i<n;i++)
    {
printf(("p%d",i+1);
scanf("%d",&bt[i]);
        st[i]=bt[i];
    }
    printf("enter the time quantum");
    scanf("%d",&tq);
    while(1)
    {
        for(i=0,count=0;i<n;i++)
        {
            temp=tq;
            if(st[i]==0)
            {
                count++;
                continue;
            }
            if(st[i]>tq)
            st[i]=st[i]-tq;
```

```
                else
                if(st[i]>=0)
                {
                        temp=st[i];
                        st[i]=0;
                }
                sq=sq+temp;
                tat[i]=sq;
        }
        if(n==count)
        break;
    }
    for(i=0;i<n;i++)
    {
        wt[i]=tat[i]-bt[i];
        swt=swt+wt[i];
        stat=stat+tat[i];
    }
    awt=(float)swt/n;
    atat=(float)stat/n;
    printf("process no\t burst time\t waiting time\t turnaround time\n");
    for(i=0;i<n;i++)
    printf("%d\t\t %d\t\t %d\t\t %d\n",i+1,bt[i],wt[i],tat[i]);
    printf("avg wt time=%f,avg turn around time=%f",awt,atat);
}
```

Output:

```
[188r1a0501@localhost ~]$ vi w1d.c
[188r1a0501@localhost ~]$ gcc  w1d.c
[188r1a0501@localhost ~]$ ./a.out
enter the number of processes  3
enter the burst time of each process
p1  7
p2  2
p3  8
enter the time quantum  2
process no       burst time      waiting time    turnaround time
1                7               8               15
2                2               2               4
3                8               9               17
avg wt time=6.333333,avg turn around time=12.000000[188r1a0501@localhost ~]$
```

d) Priority

Aim: Write a C program to implement the various process scheduling mechanisms such as Priority Scheduling.

Algorithm:

1: Start the process

2: Accept the number of processes in the ready Queue

3: For each process in the ready Q, assign the process id and accept the CPU burst time 4: Sort the ready queue according to the priority number.
5: Set the waiting of the first process as '0' and its burst time as its turn around time

6: For each process in the Ready Q calculate

(e) Waiting time for process(n)= waiting time of process (n-1) + Burst time of process(n-1)
(f) Turn around time for Process(n)= waiting time of Process(n)+ Burst time for process(n)
7: Calculate

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

(h) Average Turnaround time = Total Turnaround Time / Number of process Step 8: Stop the process

Program:

```
#include<stdio.h>
int main()
{
   int bt[20],p[20],wt[20],tat[20],pri[20],i,j,k,n,total=0,pos,temp;
   float avg_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
   }
printf(" enter priority of the process ");
for(i=0;i<n;i++)
{
p[i] = i;
//printf("Priority of Process");
printf("p%d ",i+1);
scanf("%d",&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;
 }
```

```
    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  \tPriority  \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\t\t  %d\t\t  %d\t\t  %d\t\t\t%d",p[i],bt[i],pri[i],wt[i],tat[i]);
    }

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

Output:

```
[188r1a0501@localhost ~]$ vi w1b.c
[188r1a0501@localhost ~]$ gcc  w1b.c
[188r1a0501@localhost ~]$ ./a.out
Enter number of process:3

Enter Burst Time:
p1:5
p2:6
p3:7
 enter priority of the process p1 1
p2 3
p3 2

Process     Burst Time          Priority        Waiting Time    Turnaround Time
p0                5               1               0                       5
p2                7               2               5                       12
p1                6               3               12                      18

Average Waiting Time=5.666667
Average Turnaround Time=11.666667
```

WEEK-2

Write programs using the I/O system calls of UNIX/LINUX operating system (open, read, write, close, fcntl, seek, stat, opendir, readdir)

Aim: C program using open, read, write, close system calls

Theory:

There are 5 basic system calls that Unix provides for file I/O.

1. Create: Used to Create a new empty file

Syntax :int creat(char *filename, mode_t mode)
filename : name of the file which you want to create
mode : indicates permissions of new file.

2. **open**: Used to Open the file for reading, writing or both.

Syntax: int open(char *path, int flags [, int mode]);
Path : path to file which you want to use
flags : How you like to use
O_RDONLY: read only, O_WRONLY: write only, O_RDWR: read and write, O_CREAT: create file if it doesn't exist, O_EXCL: prevent creation if it already exists

3. **close**: Tells the operating system you are done with a file descriptor and Close the file which pointed by fd.

Syntax: int close(int fd);
fd :file descriptor

4. **read:** From the file indicated by the file descriptor fd, the read() function reads cnt bytes of input into the memory area indicated by buf. A successful read() updates the access time for the file.

Syntax: int read(int fd, char *buf, int size);
fd: file descripter
buf: buffer to read data from
cnt: length of buffer

5. **write**: Writes cnt bytes from buf to the file or socket associated with fd. cnt should not be greater than INT_MAX (defined in the limits.h header file). If cnt is zero, write() simply returns 0 without attempting any other action.

Syntax: int write(int fd, char *buf, int size);
fd: file descripter
buf: buffer to write data to
cnt: length of buffer

*__File descriptor__ is integer that uniquely identifies an open file of the process.

Algorithm

1. Star the program.
2. Open a file for O_RDWR for R/W,O_CREATE for creating a file ,O_TRUNC for truncate a file.
3. Using getchar(), read the character and stored in the string[] array.
4. The string [] array is write into a file close it.
5. Then the first is opened for read only mode and read the characters and displayed it and close the file.
6. Stop the program.

Program

```
#include<sys/stat.h>
#include<stdio.h>
#include<fcntl.h>
#include<sys/types.h>
int main()
{
   int n,i=0;
   int f1,f2;
   char c,strin[100];
   f1=open("data",O_RDWR|O_CREAT|O_TRUNC);
   while((c=getchar())!='\n')
   {
      strin[i++]=c;

   }
   strin[i]='\0';
   write(f1,strin,i);
   close(f1);
   f2=open("data",O_RDONLY);
   read(f2,strin,0);
   printf("\n%s\n",strin);
   close(f2);
  return 0;

}
```

Output:

Hai
Hai

b) Aim: C program using lseek

Theory:

lseek is a system call that is used to change the location of the read/write pointer of a file descriptor. The location can be set either in absolute or relative terms.

Syntax : off_t lseek(int fildes, off_t offset, int whence);

int fildes : The file descriptor of the pointer that is going to be moved.

off_t offset : The offset of the pointer (measured in bytes).

int whence : Legal values for this variable are provided at the end which are SEEK_SET (Offset is to be measured in absolute terms), SEEK_CUR (Offset is to be measured relative to the current location of the pointer), SEEK_END (Offset is to be measured relative to the end of the file)

Algorithm:

1. Start the program
2. Open a file in read mode
3. Read the contents of the file
4. Use lseek to change the position of pointer in the read process
5. Stop

Program:

```
#include<stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>

int main()
{
	int file=0;
	if((file=open("testfile.txt",O_RDONLY)) < -1)
		return 1;

	char buffer[19];
	if(read(file,buffer,19) != 19)  return 1;
	printf("%s\n",buffer);

	if(lseek(file,10,SEEK_SET) < 0) return 1;

	if(read(file,buffer,19) != 19)  return 1;
	printf("%s\n",buffer);

	return 0;
}
```

Output:

```
[188r1a0501@localhost ~]$ vi testfile.txt
[188r1a0501@localhost ~]$ cat testfile.txt
sdfghjkkl;mnbbvrtyuijnnb
ggtyujjg

[188r1a0501@localhost ~]$ gcc  w2c.c
[188r1a0501@localhost ~]$ ./a.out
sdfghjkkl;mnbbvrtyu
mnbbvrtyuijnnb
ggty
```

c) Aim: C program using opendir(), closedir(), readdir()

Theory:

The following are the various operations using directories

1. Creating directories.
   Syntax : int mkdir(const char *pathname, mode_t mode);
2. The 'pathname' argument is used for the name of the directory.
3. Opening directories
   Syntax : DIR *opendir(const char *name);
4. Reading directories.
   Syntax: struct dirent *readdir(DIR *dirp);
5. Removing directories.
   Syntax: int rmdir(const char *pathname);
6. Closing the directory.
   Syntax: int closedir(DIR *dirp);
7. Getting the current working directory.
   Syntax: char *getcwd(char *buf, size_t size);

Algorithm:

1. Start the program
2. Print a menu to choose the different directory operations
3. To create and remove a directory ask the user for name and create and remove the same respectively.
4. To open a directory check whether directory exists or not. If yes open the directory .If it does not exists print an error message.
5. Finally close the opened directory.
6. Stop

Program:

```
#include<stdio.h>
#include<fcntl.h>
#include<dirent.h>
main()
{
char d[10]; int c,op; DIR *e;
struct dirent *sd;
printf("**menu**\n1.create dir\n2.remove dir\n 3.read dir\n enter ur choice");
scanf("%d",&op);
switch(op)
{
case 1: printf("enter dir name\n"); scanf("%s",&d);
c=mkdir(d,777);
if(c==1)
printf("dir is not created");
 else
printf("dir is created"); break;
case 2: printf("enter dir name\n"); scanf("%s",&d);
c=rmdir(d);
if(c==1)
printf("dir is not removed");
 else
printf("dir is removed"); break;
case 3: printf("enter dir name to open");
scanf("%s",&d);
e=opendir(d);
if(e==NULL)
printf("dir does not exist"); else
{
printf("dir exist\n"); while((sd=readdir(e))!=NULL) printf("%s\t",sd->d_name);
}
closedir(e);
break;
}
}
```

Output:

```
[188r1a0501@localhost f]$ gcc  w2e.c
[188r1a0501@localhost f]$ ./a.out
**menu**
1.create dir
2.remove dir
 3.read dir
 enter ur choice1
enter dir name
d
dir is created[188r1a0501@localhost f]$ ls
a.out  a.txt  d  w2d.c  w2e.c
```

WEEK -3

Write a C program to simulate Bankers Algorithm for Deadlock Avoidance and Prevention

a) Aim

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

Data structures

1. n- Number of process, m-number of resource types.
2. Available: Available[j]=k, k – instance of resource type Rj is available.
3. Max: If max [i, j]=k, Pi may request at most k instances resource Rj.
4. Allocation: If Allocation [i, j]=k, Pi allocated to k instances of resource Rj
5. Need: If Need[I, j]=k, Pi may need k more instances of resource type Rj,
6. Need [I, j] =Max [I, j]-Allocation [I, j];

Safety Algorithm

1. Work and Finish be the vector of length m and n respectively, Work=Available and Finish[i] =False.
2. Find an i such that both
3. Finish[i] =False
4. Need<=Work
5. If no such I exist go to step 4.
6. work=work+Allocation, Finish[i] =True;
7. If Finish [1] =True for all I, then the system is in safe state.

Resource request algorithm

1. Let Request i be request vector for the process Pi, If request i=[j]=k, then process Pi wants k instances of resource type Rj.
2. If Request<=Need I go to step 2. Otherwise raise an error condition.
3. If Request<=Available go to step 3. Otherwise Pi must since the resources are available.
4. Have the system pretend to have allocated the requested resources to process Pi by modifying the state as follows;
5. Available=Available-Request I;
6. Allocation I =Allocation+Request I;
7. Need i=Need i-Request I;

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

Algorithm:

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

Program:

```
#include<stdio.h>

int main ()
{
  int allocated[15][15], max[15][15], need[15][15], avail[15], tres[15],
    work[15], flag[15];
  int pno, rno, i, j, prc, count, t, total;
  count = 0;
  //clrscr ();

  printf ("\n Enter number of process:");
  scanf ("%d", &pno);
  printf ("\n Enter number of resources:");
  scanf ("%d", &rno);
  for (i = 1; i <= pno; i++)
    {
      flag[i] = 0;
    }
  printf ("\n Enter total numbers of each resources:");
  for (i = 1; i <= rno; i++)
    scanf ("%d", &tres[i]);


  printf ("\n Enter Max resources for each process:");
  for (i = 1; i <= pno; i++)
    {
```

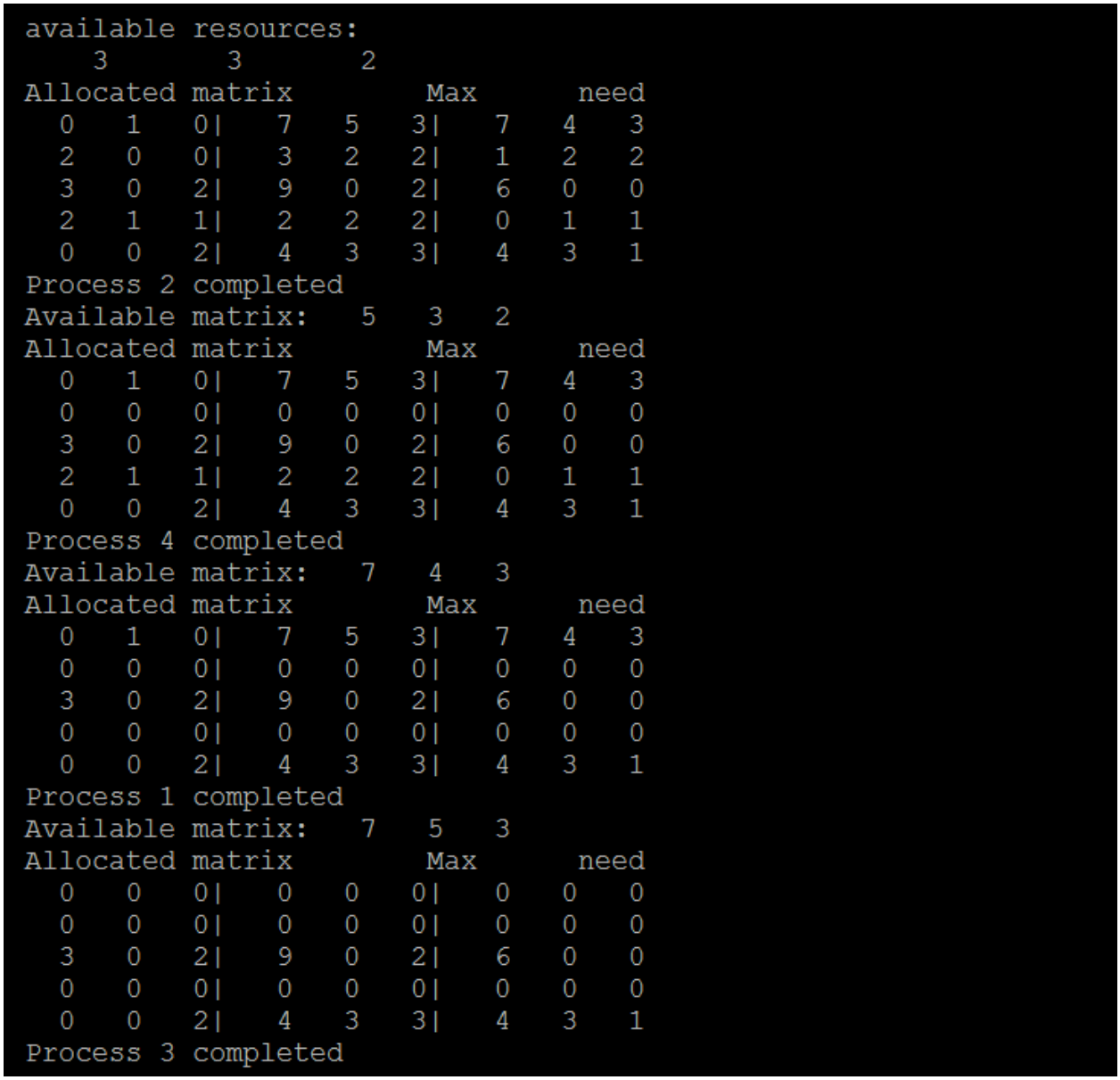
```
    printf ("\n for process %d:", i);
    for (j = 1; j <= rno; j++)
      scanf ("%d", &max[i][j]);
   }


  printf ("\n Enter allocated resources for each process:");
  for (i = 1; i <= pno; i++)
   {
    printf ("\n for process %d:", i);
    for (j = 1; j <= rno; j++)
      scanf ("%d", &allocated[i][j]);


   }
printf ("\n available resources:\n");
  for (j = 1; j <= rno; j++)
   {
    avail[j] = 0;
    total = 0;
    for (i = 1; i <= pno; i++)
      {
       total += allocated[i][j];
      }
    avail[j] = tres[j] - total;
    work[j] = avail[j];
    printf ("     %d \t", work[j]);
   }


  do
   {

    for (i = 1; i <= pno; i++)
      {
       for (j = 1; j <= rno; j++)
         {
          need[i][j] = max[i][j] - allocated[i][j];
         }
      }
```

```
printf ("\n Allocated matrix        Max      need");
     for (i = 1; i <= pno; i++)
       {
         printf ("\n");
         for (j = 1; j <= rno; j++)
           {
             printf ("%4d", allocated[i][j]);
           }
         printf ("|");
         for (j = 1; j <= rno; j++)
           {
             printf ("%4d", max[i][j]);
           }
         printf ("|");
         for (j = 1; j <= rno; j++)
           {
             printf ("%4d", need[i][j]);
           }
       }

     prc = 0;

     for (i = 1; i <= pno; i++)
       {
         if (flag[i] == 0)
           {
             prc = i;

             for (j = 1; j <= rno; j++)
               {
                 if (work[j] < need[i][j])
                   {
                     prc = 0;
                     break;
                   }
               }
           }
if (prc != 0)
           break;
       }
```

```
    if (prc != 0)
      {
        printf ("\n Process %d completed", i);
        count++;
        printf ("\n Available matrix:");
        for (j = 1; j <= rno; j++)
          {
            work[j] += allocated[prc][j];
            allocated[prc][j] = 0;
            max[prc][j] = 0;
            flag[prc] = 1;
            printf ("  %d", work[j]);
          }
      }

   }
  while (count != pno && prc != 0);

  if (count == pno)
    printf ("\nThe system is in a safe state!!");
  else
    printf ("\nThe system is in an unsafe state!!");
return 0;

}
```

Output:

```
[188r1a0501@localhost ~]$ vi dp.c
[188r1a0501@localhost ~]$ gcc  dp.c
[188r1a0501@localhost ~]$ ./a.out

 Enter number of process:5

 Enter number of resources:3

 Enter total numbers of each resources:10       5       7

 Enter Max resources for each process:
 for process 1:7        5       3

 for process 2:3        2       2

 for process 3:9        0       2

 for process 4:2        2       2

 for process 5:4        3       3

 Enter allocated resources for each process:
 for process 1:0        1       0

 for process 2:2        0       0

 for process 3:3        0       2

 for process 4:2        1       1

 for process 5:0        0       2
```


available resources:
3 3 2
Allocated matrix Max need
0 1 0| 7 5 3| 7 4 3
2 0 0| 3 2 2| 1 2 2
3 0 2| 9 0 2| 6 0 0
2 1 1| 2 2 2| 0 1 1
0 0 2| 4 3 3| 4 3 1
Process 2 completed
Available matrix: 5 3 2
Allocated matrix Max need
0 1 0| 7 5 3| 7 4 3
0 0 0| 0 0 0| 0 0 0
3 0 2| 9 0 2| 6 0 0
2 1 1| 2 2 2| 0 1 1
0 0 2| 4 3 3| 4 3 1
Process 4 completed
Available matrix: 7 4 3
Allocated matrix Max need
0 1 0| 7 5 3| 7 4 3
0 0 0| 0 0 0| 0 0 0
3 0 2| 9 0 2| 6 0 0
0 0 0| 0 0 0| 0 0 0
0 0 2| 4 3 3| 4 3 1
Process 1 completed
Available matrix: 7 5 3
Allocated matrix Max need
0 0 0| 0 0 0| 0 0 0
0 0 0| 0 0 0| 0 0 0
3 0 2| 9 0 2| 6 0 0
0 0 0| 0 0 0| 0 0 0
0 0 2| 4 3 3| 4 3 1
Process 3 completed
Available matrix: 10 5 5
Allocated matrix Max need
0 0 0| 0 0 0| 0 0 0
0 0 0| 0 0 0| 0 0 0
0 0 0| 0 0 0| 0 0 0
0 0 0| 0 0 0| 0 0 0
0 0 2| 4 3 3| 4 3 1
Process 5 completed
Available matrix: 10 5 7

b) Aim

Write a C program to simulate Bankers Algorithm for Deadlock Prevention

Algorithm:

1. Start
2. Attacking Mutex condition : never grant exclusive access. but this may not be possible for several resources.
3. Attacking preemption: not something you want to do.
4. Attacking hold and wait condition : make a process hold at the most 1 resource at a time.make all the requests at the beginning. All or nothing policy. If you feel,retry. eg. 2-phase locking 34
5. Attacking circular wait: Order all the resources. Make sure that the requests are issued in the correct order so that there are no cycles present in the resource graph. Resources numbered 1 ... n. Resources can be requested only in increasing order. ie. you cannot request a resource whose no is less than any you may be holding.
6. Stop

Program:

```
#include<stdio.h>

int max[10][10],alloc[10][10],need[10][10],avail[10],i,j,p,r,finish[10]={0},flag=0;
main( )
{

printf("\n SIMULATION OF DEADLOCK PREVENTION \n ");
printf("Enter no. of processes, resources\n ");
scanf("%d%d",&p,&r);
printf("Enter allocation matrix");
for(i=0;i<p;i++)
for(j=0;j<r;j++)
scanf("%d",&alloc[i][j]);
printf("\n  enter max matrix");
for(i=0;i<p;i++) /*reading the maximum matrix and availale matrix*/
for(j=0;j<r;j++)
scanf("%d",&max[i][j]);
printf(" \n enter available matrix");
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];
fun(); /*calling function*/
if(flag==0)
{if(finish[i]!=1)
{
printf("\n Failing :Mutual exclusion");
for(j=0;j<r;j++)
{ /*checking for mutual exclusion*/
if(avail[j]<need[i][j])
avail[j]=need[i][j];
}fun();
printf("\n By allocating required resources to process %d dead lock is prevented ",i);
printf("\n lack of preemption");
for(j=0;j<r;j++)
{
if(avail[j]<need[i][j])
avail[j]=need[i][j];
alloc[i][j]=0;
}
fun( );
printf("\n dead lock is prevented by allocating needed resources");

printf(" \n failing:Hold and Wait condition ");
for(j=0;j<r;j++)
{ /*checking hold and wait condition*/
if(avail[j]<need[i][j])
avail[j]=need[i][j];
}
fun( );
printf("\n AVOIDING ANY ONE OF THE CONDITION, U CAN PREVENT DEADLOCK");
}
}
}
fun()
{
while(1)
{
for(flag=0,i=0;i<p;i++)
```

```
{
if(finish[i]==0)
{
for(j=0;j<r;j++)
{
if(need[i][j]<=avail[j])
continue;
else
break;
}
if(j==r)
{
for(j=0;j<r;j++)
avail[j]+=alloc[i][j];
flag=1;
finish[i]=1;
}
}
}
```

Output:

```
[188r1a0501@localhost ~]$ vi dp1.c
[188r1a0501@localhost ~]$ gcc dp1.c
[188r1a0501@localhost ~]$ ./a.out

 SIMULATION OF DEADLOCK PREVENTION
 Enter no. of processes, resources
 3
2
Enter allocation matrix4
5
3
4
5
2

  enter max matrix4
3
4
5
6
1

 enter available matrix2
5

 Failing :Mutual exclusion
 By allocating required resources to process 3 dead lock is prevented
 lack of preemption
 dead lock is prevented by allocating needed resources
  failing:Hold and Wait condition
 AVOIDING ANY ONE OF THE CONDITION, U CAN PREVENT DEADLOCK[188r1a0501@localhost ~]$
```

WEEK-4

Write a C program to implement the Producer – Consumer problem using semaphores using UNIX/LINUX system calls.

Aim:

Write a C program to implement the Producer – Consumer problem using semaphores using UNIX/LINUX system calls.

Algorithm:

1. The Semaphore mutex, full & empty are initialized.
2. In the case of producer process
3. Produce an item in to temporary variable.
   If there is empty space in the buffer check the mutex value for enter into the critical section.
   If the mutex value is 0, allow the producer to add value in the temporary variable to the buffer.
4. In the case of consumer process
   i) It should wait if the buffer is empty
   ii) If there is any item in the buffer check for mutex value, if the mutex==0, remove item from buffer
   iii) Signal the mutex value and reduce the empty value by 1.
   iv) Consume the item.
5. Print the result

Program:

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

int mutex = 1, full = 0, empty = 3, x = 0;

int main ()
{
 int n;
 void producer ();
 void consumer ();
 int wait (int);
 int signal (int);
 printf ("\n1.Producer\n2.Consumer\n3.Exit");
 while (1)
  {
   printf ("\nEnter your choice:");
   scanf ("%d", &n);
   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;
        }
  }

 return 0;
```

```
}

int wait (int s)
{
  return (--s);
}

int signal (int s)
{
  return (++s);
}

void producer ()
{
  mutex = wait (mutex);
  full = signal (full);
  empty = wait (empty);
  x++;
  printf ("\nProducer produces the item %d", x);
  mutex = signal (mutex);
}

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

Output:

```
[188r1a0501@localhost ~]$ vi pc.c
[188r1a0501@localhost ~]$ gcc  pc.c
[188r1a0501@localhost ~]$ ./a.out

1.Producer
2.Consumer
3.Exit
Enter your choice:1

Producer produces the item 1
Enter your choice:1

Producer produces the item 2
Enter your choice:1

Producer produces the item 3
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
[188r1a0501@localhost ~]$
```

Week: 5

Write C programs to illustrate the following IPC mechanisms

Aim: Write C programs to illustrate the following IPC mechanisms

ALGORITHM:

1. Start the program.
2. Declare the variables.
3. Read the choice.
4. Create a piping processing using IPC.
5. Assign the variable lengths
6. "*strcpy"* the message lengths.
7. To join the operation using IPC .
8. Stop the program

Program : (PIPE PROCESSING)

```
#include <unistd.h>
#include <stdlib.h> #include <stdio.h>
#include <string.h> #define MSG_LEN 64 int main(){
Int result;
int fd[2];
char  message[MSG_LEN];
 char recvd_msg[MSG_LE]; result = pipe (fd);
//Creating a pipe//fd[0] is for reading and fd[1] is for writing if (result < 0){
perror("pipe "); exit(1);
}
strncpy(message,"Linux  World!! ",MSG_LEN);  result=write(fd[1],message,strlen(message));  if  (result
< 0){
perror("write"); exit(2);
}
strncpy(message,"Understanding ",MSG_LEN); result=write(fd[1],message,strlen(message)); if
(result < 0){
perror("write"); exit(2);
}
strncpy(message,"Concepts of ",MSG_LEN); result=write(fd[1],message,strlen(message)); if (result <
0){

perror("write");  exit(2);
}

strncpy(message,"Piping ",  MSG_LEN);  result=write(fd[1],message,strlen(message));  if  (result  <  0){
perror("write"); exit(2);
}

result=read(fd[0],recvd_msg,MSG_LEN); if (result < 0){
perror("read"); exit(3);
}

printf("%s\n",recvd_msg); return 0;
}
```

a) FIFO

Program:

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

#include <linux/stat.h>

#define FIFO_FILE       "MYFIFO"

int main(void)
{
      FILE *fp;
      char readbuf[80];

      /* Create the FIFO if it does not exist */
      umask(0);
      mknod(FIFO_FILE, S_IFIFO|0666, 0);

      while(1)
      {
            fp = fopen(FIFO_FILE, "r");
            fgets(readbuf, 80, fp);
            printf("Received string: %s\n", readbuf);
            fclose(fp);
      }

      return(0);
}
```

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

#define FIFO_FILE       "MYFIFO"

int main(int argc, char *argv[])
{
        FILE *fp;

        if ( argc != 2 ) {
                printf("USAGE: fifoclient [string]\n");
                exit(1);
        }

        if((fp = fopen(FIFO_FILE, "w")) == NULL) {
                perror("fopen");
                exit(1);
        }
        fputs(argv[1], fp);

        fclose(fp);
        return(0);
}
```

C Program for Message Queue (Writer Process)

```
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/msg.h>

// structure for message queue
struct mesg_buffer {
    long  msg_type;
    char msg_text[100];
} message;

int main()
{
    key_t key;
    int msgid;
   // ftok to generate unique key
    key = ftok("progfile", 65);
  // msgget creates a message queue
    // and returns identifier
    msgid = msgget(key, 0666 | IPC_CREAT);
    message.mesg_type = 1;

    printf("Write Data : ");
    gets(message.mesg_text);

    // msgsnd to send message
    msgsnd(msgid, &message, sizeof(message), 0);

    // display the message
    printf("Data send is : %s \n", message.mesg_text);

    return 0;
}
```

C Program for Message Queue (Reader Process)

```
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/msg.h>

// structure for message queue
struct mesg_buffer {
    long mesg_type;
```

```
    char mesg_text[100];
} message;

int main()
{
    key_t key;
    int msgid;

    // ftok to generate unique key
key = ftok("progfile", 65);



// msgget creates a message queue  and returns identifier
    msgid = msgget(key, 0666 | IPC_CREAT);
    // msgrcv to receive message
    msgrcv(msgid, &message, sizeof(message), 1, 0);

    // display the message
    printf("Data Received is : %s \n",
                    message.mesg_text);

    // to destroy the message queue
    msgctl(msgid, IPC_RMID, NULL);

    return 0;
}
```

C Program for Message Queue (Reader Process)

```
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/msg.h>
  // structure for message queue
struct mesg_buffer {
    long  mesg_type;
    char mesg_text[100];
} message;

int main()
{
    key_t key;
    int msgid;
```

```
    // ftok to generate unique key
    key = ftok("progfile", 65);

    // msgget creates a message queue
    // and returns identifier
    msgid = msgget(key, 0666 | IPC_CREAT);

    // msgrcv to receive message
    msgrcv(msgid, &message, sizeof(message), 1, 0);

    // display the message
    printf("Data Received is : %s \n",
                message.mesg_text);

    // to destroy the message queue
    msgctl(msgid, IPC_RMID, NULL);

    return 0;
}
```

OUTPUT: Thus the Piping process using IPC program was executed and verified successfully

Week: 6

Aim: Write C programs to simulate the following memory management techniques

a) Paging

AIM: To write a C program to implement memory management using paging technique.

ALGORITHM:

Step1 : Start the program.
Step2 : Read the base address, page size, number of pages and memory unit.
Step3 : If the memory limit is less than the base address display the memory limit is less than limit.
Step4 : Create the page table with the number of pages and page address.
Step5 : Read the page number and displacement value.
Step6 : If the page number and displacement value is valid, add the displacement value with the address corresponding to the page number and display the result.
Step7 : Display the page is not found or displacement should be less than page size.
Step8 : Stop the program.

Program:

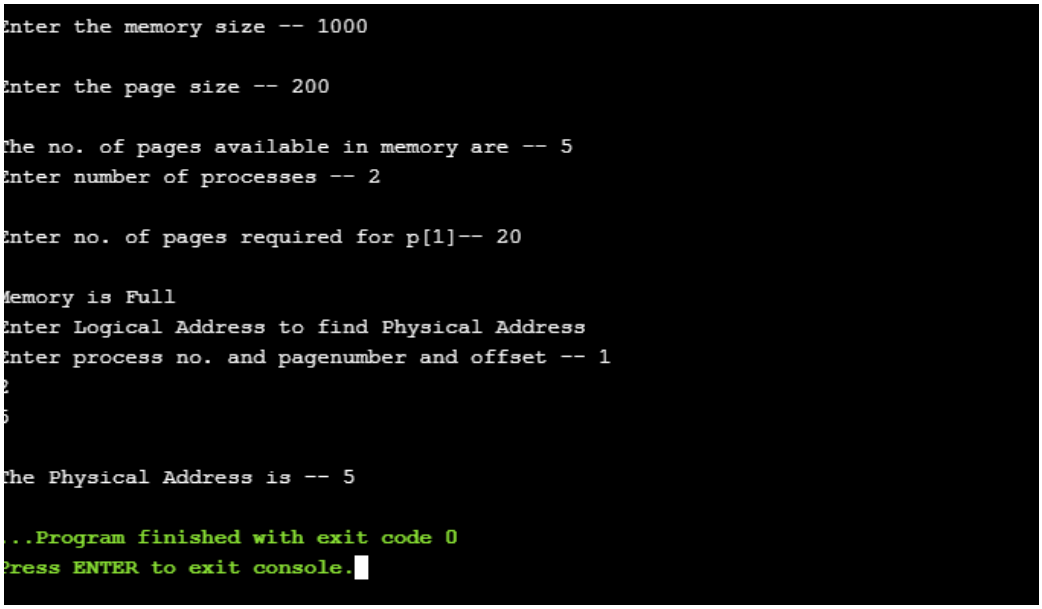
```
#include<stdio.h>

#include<conio.h>

main()

{

int ms, ps, nop, np, rempages, i, j, x, y, pa, offset; int s[10], fno[10][20];

printf("\nEnter the memory size -- ");

scanf("%d",&ms);

printf("\nEnter the page size -- ");

scanf("%d",&ps);

nop = ms/ps;

printf("\nThe no. of pages available in memory are -- %d ",nop);

printf("\nEnter number of processes -- ");

scanf("%d",&np);
```

```
rempages = nop; for(i=1;i<=np;i++)
{

printf("\nEnter no. of pages required for p[%d]-- ",i);
scanf("%d",&s[i]);
if(s[i] >rempages)
{
printf("\nMemory is Full");
break;
}
rempages = rempages - s[i];
printf("\nEnter pagetable for p[%d] --- ",i);
for(j=0;j<s[i];j++)
scanf("%d",&fno[i][j]);
}

printf("\nEnter Logical Address to find Physical Address ");
printf("\nEnter process no. and pagenumber and offset -- ");
scanf("%d %d %d",&x,&y, &offset);
if(x>np || y>=s[i] || offset>=ps)
printf("\nInvalid Process or Page Number or offset");
else
{
pa=fno[x][y]*ps+offset;
printf("\nThe Physical Address is -- %d",pa);
}
```

getch();

}

OUTPUT:



b) Segmentation

Aim: To write a C program to implement memory management using segmentation

Algorithm:

Step1 : Start the program.
Step2 : Read the base address, number of segments, size of each segment, memory limit.
Step3 : If memory address is less than the base address display "invalid memory limit".
Step4 : Create the segment table with the segment number and segment address and display it.
Step5 : Read the segment number and displacement.
Step6 : If the segment number and displacement is valid compute the real address and display the same. Step7 : Stop the program.

Program:

```
#include<stdio.h>
#include<conio.h>
struct list
{
int seg;
int base;
int limit;
struct list *next;
} *p;
void insert(struct list *q,int base,int limit,int seg)
{
if(p==NULL)
{
p=malloc(sizeof(Struct list));
p->limit=limit;
p->base=base;
p->seg=seg;
p->next=NULL;
}
else
{
while(q->next!=NULL)
{
Q=q->next;
Printf("yes")
}
q->next=malloc(sizeof(Struct list));
q->next ->limit=limit;
q->next ->base=base;
q->next ->seg=seg;
```

```
q->next ->next=NULL;
}
}
int find(struct list *q,int seg)
{
while(q->seg!=seg)
{
q=q->next;
}
return q->limit;
}
int search(struct list *q,int seg)
{
while(q->seg!=seg)
{
q=q->next;
}
return q->base;
}
main()
{


p=NULL;
int seg,offset,limit,base,c,s,physical;
printf("Enter segment table/n");
printf("Enter -1 as segment value for termination\n");
do
{
printf("Enter segment number");
scanf("%d",&seg);
if(seg!=-1)
{
printf("Enter base value:");
scanf("%d",&base);

printf("Enter value for limit:");
scanf("%d",&limit);
insert(p,base,lmit,seg);
}
}
while(seg!=-1)
printf("Enter offset:");
```

```
scanf(“%d”,&offset);
printf(“Enter bsegmentation number:”);
scanf(“%d”,&seg);
c=find(p,seg);
s=search(p,seg);
if(offset<c)
{
physical=s+offset;
printf(“Address in physical memory %d\n”,physical);
}
else
{
printf(“error”);
}
```

OUTPUT:

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

Enter segment table
Enter -1 as segmentation value for termination
Enter segment number:1
Enter base value:2000
Enter value for limit:100
Enter segment number:2
Enter base value:2500
Enter value for limit:100
Enter segmentation number:-1
Enter offset:90
Enter segment number:1
Address in physical memory 20
```

Week-7 PAGE REPLACEMENT ALGORITHMS

AIM: To implement FIFO page replacement technique.

a) FIFO b) LRU c) OPTIMAL

DESCRIPTION:

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

FIFO-This is the simplest page replacement algorithm. In this algorithm, the operating system keeps track of all pages in the memory in a queue, the oldest page is in the front of the queue. When a page needs to be replaced page in the front of the queue is selected for removal.

LRU-In this algorithm page will be replaced which is least recently used

OPTIMAL- In this algorithm, pages are replaced which would not be used for the longest duration of time in the future. This algorithm will give us less page fault when compared to other page replacement algorithms.

ALGORITHM:

1. Start the process
2. Read number of pages n
3. Read number of Pages no
4. Read page numbers into an array a[i]
5. Initialize avail[i] =0 to check page hit
6. Replace the pages with circular queue, while re-placing check page availability in the frame place avail[i]=1, if page is placed in the frame Count page faults
7. Print the results.
8. Stop the process.

A) FIRST IN FIRST OUT

SOURCE CODE :

```
#include<stdio.h>
#include<conio.h> int fr[3];
void main()
{
void display();
int i,j,page[12]={2,3,2,1,5,2,4,5,3,2,5,2};
int
flag1=0,flag2=0,pf=0,frsize=3,top=0;
clrscr();
for(i=0;i<3;i++)
{
fr[i]=-1;
}
for(j=0;j<12;j++)
{
flag1=0; flag2=0; for(i=0;i<12;i++)
{
if(fr[i]==page[j])
{
flag1=1; flag2=1; break;
}
}
if(flag1==0)
{
for(i=0;i<frsize;i++)
{
if(fr[i]==-1)
{
fr[i]=page[j]; flag2=1; break;
}
}
}
if(flag2==0)
{
fr[top]=page[j];
top++;
pf++;
if(top>=frsize)
top=0;
}
display();
}
```

```
printf("Number of page faults : %d ",pf+frsize);
 getch();
}
void display()
{
int     i;     printf("\n");
for(i=0;i<3;i++)
printf("%d\t",fr[i]);
}
```

OUTPUT:

```
2 -1 -1
2  3 -1
2  3 -1
2  3  1
5  3  1
5  2  1
5  2  4
5  2  4
3  2  4
3  2  4
3  5  4
3  5  2
```

Number of page faults: 9

B) LEAST RECENTLY USED

AIM: To implement LRU page replacement technique.

ALGORITHM:

1. Start the process
2. Declare the size
3. Get the number of pages to be inserted
4. Get the value
5. Declare counter and stack
6. Select the least recently used page by counter value
7. Stack them according the selection.
8. Display the values
9. Stop the process

SOURCE CODE :

```
#include<stdio.h>
#include<conio.h>
int fr[3];
void main()
{
void display();
int p[12]={2,3,2,1,5,2,4,5,3,2,5,2},i,j,fs[3];
int index,k,l,flag1=0,flag2=0,pf=0,frsize=3;
 clrscr();
for(i=0;i<3;i++)
{
fr[i]=-1;
}
for(j=0;j<12;j++)
{
flag1=0,flag2=0;
for(i=0;i<3;i++)
{
if(fr[i]==p[j])
{
flag1=1;
flag2=1; break;
}
}
if(flag1==0)
```

```
{
for(i=0;i<3;i++)
{
if(fr[i]==-1)
{
fr[i]=p[j];       flag2=1;
break;
}
}
}
if(flag2==0)
{
for(i=0;i<3;i++)
fs[i]=0;
for(k=j-1,l=1;l<=frsize-1;l++,k--)
{
for(i=0;i<3;i++)
{
if(fr[i]==p[k]) fs[i]=1;
}}
for(i=0;i<3;i++)
{
if(fs[i]==0)
index=i;
}
fr[index]=p[j];
pf++;
}
display();
}
printf("\n no of page faults :%d",pf+frsize);
getch();
}
void display()
{
int i; printf("\n");
for(i=0;i<3;i++)
printf("\t%d",fr[i]);
}
```

OUTPUT:

2 -1 -1
2 3 -1
2 3 -1
2 3 1
2 5 1
2 5 1
2 5 4
2 5 4
3 5 4
3 5 2
3 5 2
3 5 2

No of page faults: 7

C) OPTIMAL

AIM: To implement optimal page replacement technique.

ALGORTHIM:

1. Start Program
2. Read Number Of Pages And Frames
3.Read Each Page Value
4. Search For Page In The Frames
5.If Not Available Allocate Free Frame
6. If No Frames Is Free Repalce The Page With The Page That Is Leastly Used
7.Print Page Number Of Page Faults
8.Stop process.

SOURCE CODE:

```
/* Program to simulate optimal page replacement */
#include<stdio.h>
#include<conio.h>
int fr[3], n, m;
void
display();
void main()
{
 int i,j,page[20],fs[10];
 int
 max,found=0,lg[3],index,k,l,flag1=0,flag2=0,pf=0;
 float pr;
 clrscr();
 printf("Enter length of the reference string: ");
 scanf("%d",&n);
 printf("Enter the reference string: ");
 for(i=0;i<n;i++)
 scanf("%d",&page[i]);
 printf("Enter no of frames: ");
 scanf("%d",&m);
 for(i=0;i<m;i++)
 fr[i]=-1; pf=m;
```

```
for(j=0;j<n;j++)
 {
 flag1=0;          flag2=0;
 for(i=0;i<m;i++)
 {
 if(fr[i]==page[j])
 {
 flag1=1;  flag2=1;
 break;
 }
 }
 if(flag1==0)
 {
 for(i=0;i<m;i++)
 {
 if(fr[i]==-1)
 {
fr[i]=page[j];  flag2=1;
break;
 }
 }
 }
 if(flag2==0)
 {
 for(i=0;i<m;i++)
 lg[i]=0;
 for(i=0;i<m;i++)
 {
 for(k=j+1;k<=n;k++)
 {
 if(fr[i]==page[k])
 {
 lg[i]=k-j;
 break;
 }
 }
 }
 found=0;
 for(i=0;i<m;i++)
 {
 if(lg[i]==0)
 {
index=i;
found = 1;
```

```
 break;
 }
 }
if(found==0)
 {
 max=lg[0];    index=0;
 for(i=0;i<m;i++)
 {
 if(max<lg[i])
{
 max=lg[i];
 index=i;
}
 }
 }
 fr[index]=page[j];
 pf++;
}
display();
}
printf("Number of page faults : %d\n", pf);
pr=(float)pf/n*100;
printf("Page fault rate = %f \n", pr); getch();
}
void display()
{
 int  i;  for(i=0;i<m;i++)
 printf("%d\t",fr[i]);
 printf("\n");
}
```

OUTPUT:

```
Enter length of the reference string: 12
Enter the reference string:
1 2 3 4 1 2 5 1 2 3 4 5
Enter no of frames: 3
1 -1 -1
1 2 -1
1 2 3
1 2 4
1 2 4
1 2 4
1 2 5
1 2 5
1 2 5
3 2 5
4 2 5
4 2 5
Number of page faults : 7 Page fault rate = 58.333332
```

VIVA QUESTIONS

1) What is meant by page fault?
2) What is meant by paging?
3) What is page hit and page fault rate?
4) List the various page replacement algorithm
5) Which one is the best replacement algorithm?

OPERATING SYSTEMS – ADDITIONAL EXPERIMENTS

(R22CSE2226)

II YEAR II SEMESTER

Ex.No:1.a	**BASICS OF UNIX COMMANDS**
	INTRODUCTION TO UNIX

AIM:

To study about the basics of UNIX

UNIX:

It is a multi-user operating system. Developed at AT & T Bell Industries, USA in 1969.

Ken Thomson along with Dennis Ritchie developed it from MULTICS (Multiplexed Information and Computing Service) OS.

By1980, UNIX had been completely rewritten using C language.

LINUX:

It is similar to UNIX, which is created by Linus Torualds. All UNIX commands works in Linux. Linux is a open source software. The main feature of Linux is coexisting with other OS such as windows and UNIX.

STRUCTURE OF A LINUXSYSTEM:

It consists of three parts.

a)UNIX kernel
b) Shells
c) Tools and Applications

UNIX KERNEL:

Kernel is the core of the UNIX OS. It controls all tasks, schedule all Processes and carries out all the functions of OS.

Decides when one programs tops and another starts.

SHELL:

Shell is the command interpreter in the UNIX OS. It accepts command from the user and analyses and interprets them

Ex.No:1.b	BASICS OF UNIX COMMANDS
	BASIC UNIX COMMANDS

AIM:

To study of Basic UNIX Commands and various UNIX editors such as vi, ed, ex and EMACS.

CONTENT:

Note: Syn->Syntax

a) **date**

–used to check the date and time

Syn:$date

Format	Purpose	Example	Result
+%m	To display only month	$date+%m	06
+%h	To display month name	$date+%h	June
+%d	To display day of month	$date+%d	O1
+%y	To display last two digits of years	$date+%y	09
+%H	To display hours	$date+%H	10
+%M	To display minutes	$date+%M	45
+%S	To display seconds	$date+%S	55

b) **cal**

–used to display the calendar

Syn:$cal 2 2009

c)**echo**

–used to print the message on the screen.

Syn:$echo "text"

d)**ls**

–used to list the files. Your files are kept in a directory.

Syn:$lsls–s

All files (include files with prefix)

ls–l Lodetai (provide file statistics)

ls–t Order by creation time

ls– u Sort by access time (or show when last accessed together with –l)

ls–s Order by size

ls–r Reverse order

ls–f Mark directories with /,executable with* , symbolic links with @, local sockets with =, named pipes(FIFOs)with

ls–s Show file size

ls– h" Human Readable", show file size in Kilo Bytes & Mega Bytes (h can be used together with –l or)

ls[a-m]*List all the files whose name begin with alphabets From ‘a’ to ‘m’
ls[a]*List all the files whose name begins with ‘a’ or ‘A’
Eg:$ls>my list Output of ‘ls’ command is stored to disk file named ‘my list’

e)**lp**
–used to take printouts
Syn:$lp filename
f)**man**
–used to provide manual help on every UNIX commands.
Syn:$man unix command
$man cat

g)**who** & **whoami**
–it displays data about all users who have logged into the system currently. The next command displays about current user only.
Syn:whowhoami

h)**uptime**
–tells you how long the computer has been running since its last reboot or power-off.
Syn:$uptime

i)**uname**
–it displays the system information such as hardware platform, system name and processor, OS type
Syn:$uname–a

j)**hostname**
–displays and set system host name
Syn:$ hostname

k)**bc**
–stands for ‘best calculator’

```
$bc
10/2*3
15
```

```
$ bc
scale =1
2.25+1
3.35
quit
```

```
$ bc
ibase=2
obase=16
11010011
89275
1010
Ā
Quit
```

```
$ bc
sqrt(196)
14 quit
```

```
$bc
for(i=1;i<3;i=i+1)I
1
2
3 quit
```

```
$ bc-l
scale=2
s(3.14)
0
```

FILE MANIPULATION COMMANDS

a)**cat**–this create, view and concatenate files.

Creation:

Syn:$cat>filename

Viewing:

Syn:$cat filename

Add text to an existing file:

Syn:$cat>>filename

Concatenate:

Syn:$catfile1file2>file3

$catfile1file2>>file3 (no over writing of file3)

b)**grep**–used to search a particular word or pattern related to that word from the file.

Syn:$grep search word filename

Eg:$grep anu student

c)**rm**–deletes a file from the file system

Syn:$rm filename

d)**touch**–used to create a blank file.

Syn:$touch file names

e)**cp**–copies the files or directories

Syn:$cpsource file destination file

Eg:$cp student stud

f)**mv**–to rename the file or directory

syn:$mv old file new file

Eg:$mv–i student student list(-i prompt when overwrite)

g)**cut**–it cuts or pickup a given number of character or fields of the file.

Syn:$cut<option><filename>

Eg: $cut –c filename

$cut–c1-10emp

$cut–f 3,6emp

$ cut –f 3-6 emp

-c cutting columns

-f cutting fields

h)**head**–displays10 lines from the head(top)of a given file

Syn:$head filename

Eg:$head student

To display the top two lines:

Syn:$head-2student

i)**tail**–displays last 10 lines of the file
Syn:$tail filename
Eg:$tail student
To display the bottom two lines;
Syn:$ tail -2 student

j)**chmod**–used to change the permissions of a file or directory.
Syn:$ch mod category operation permission file
Where, Category–is the user type
Operation–is used to assign or remove permission
Permission–is the type of permission
File–are used to assign or remove permission all

Examples:
$chmodu-wx student
Removes write and execute permission for users
$ch modu+rw,g+rwstudent
Assigns read and write permission for users and groups
$chmodg=rwx student
Assigns absolute permission for groups of all read, write and execute permissions

k)**wc**–it counts the number of lines, words, character in a specified file(s)
with the options as –l,-w,-c

Category	Operation	Permission
u– users g–group o– others	+assign -remove =assign absolutely	r– read w– write x-execute

Syn: $wc –l filename
$wc –w filename
$wc–c filename

Ex.No:1.c	BASICS OF UNIX COMMANDS
	UNIX EDITORS

AIM:

To study of various UNIX editors such as vi, ed, ex and EMACS.

CONCEPT:

Editor is a program that allows user to see a portions a file on the screen and modify characters and lines by simply typing at the current position. UNIX supports variety of Editors. They are:

ed ex vi
EMACS

Vi- vi is stands for "visual".vi is the most important and powerful editor.vi is a full screen editor that allows user to view and edit entire document at the same time.vi editor was written in the University of California, at Berkley by Bill Joy, who is one of the co-founder of Sun Microsystems.

Features of vi:

It is easy to learn and has more powerful features.
Itworksgreatspeedandiscasesensitive.vihaspowerfulundofunctionsandhas3modes:

1. Command mode
2. Insert mode
3. Escape or ex mode

In command mode, no text is displayed on the screen.
In Insert mode, it permits user to edit insert or replace text.
In escape mode, it displays commands at command line.
Moving the cursor with the help of h, l, k, j, I, etc

EMACS Editor

Motion Commands:
M-> Move to end of file
M-< Move to beginning of file
C-v Move forward a screen M –v Move
backward a screen C –n Move to next line
C-p Move to previous line
C-a Move to the beginning of the line
C-e Move to the end of the line
C-f Move forward a character
C-b Move backward a character
M-f Move forward a word
M-b Move backward a word

Deletion Commands:

DEL	delete the previous character C -d
	delete the current character M -DEL
	delete the previous word
M-d	delete the next word
C-x DEL	deletes the previous sentence
M-k	delete the rest of the current sentence
C-k	deletes the rest of the current line
C-xu	undo the lasted it change

Search and Replace in EMACS:

y	Change the occurrence of the pattern
n	Don't change the occurrence, but look for the other q Don't change. Leave query replace completely
!	Change this occurrence and all others in the file

RESULT:

Ex.No:2	Programs using the following system calls of UNIX operating system fork, exec, getpid, exit, wait, close, stat, opendir, readdir

AIM:

To write C Programs using the following system calls of UNIX operating system fork, exec, getpid, exit, wait, close, stat, opendir, readdir.

1. PROGRAM FOR SYSTEM CALLS OF UNIX OPERATING SYSTEMS (OPENDIR, READDIR, CLOSEDIR)

ALGORITHM:

STEP 1: Start the program.
STEP 2: Create struct dirent.
STEP 3: declare the variable buff and pointer dptr.
STEP 4: Get the directory name.
STEP 5: Open the directory.
STEP 6: Read the contents in directory and print it.
STEP 7: Close the directory.

PROGRAM:

```
#include<stdio.h>
#include<dirent.h>
struct dirent *dptr;
int main(int argc, char *argv[])
{
char buff[100];
DIR *dirp;
printf("\n\n ENTER DIRECTORY NAME");
scanf("%s", buff);
if((dirp=opendir(buff))==NULL)
{
printf("The given directory does not exist");
exit(1);
}
while(dptr=readdir(dirp))
{
printf("%s\n",dptr->d_name);
}
closedir(dirp);
}
```

OUTPUT:

2. PROGRAM FOR SYSTEM CALLS OF UNIX OPERATING SYSTEM (fork, getpid, exit)

ALGORITHM:

STEP 1: Start the program.
STEP 2: Declare the variables pid,pid1,pid2.
STEP 3: Call fork() system call to create process.
STEP 4: If pid==-1, exit.
STEP 5: Ifpid!=-1 , get the process id using getpid().
STEP 6: Print the process id.
STEP 7:Stop the program

PROGRAM:

```
#include<stdio.h>
#include<unistd.h>
main()
{
int pid,pid1,pid2;
pid=fork();
if(pid==-1)
{
printf("ERROR IN PROCESS CREATION \n");
exit(1);
}
if(pid!=0)
{
pid1=getpid();
printf("\n the parent process ID is %d\n", pid1);
}
else
{
pid2=getpid();
printf("\n the child process ID is %d\n", pid2);
```

```
}
}
```

OUTPUT:

RESULT:

.