R20 OS LAB MANUAL

1. Practicing of Basic UNIX Commands.

Files (command -- description)

ls --- lists your files

ls -l --- lists your files in 'long format', which contains lots of useful information, e.g. the exact size of the file, who owns the file and who has the right to look at it, and when it was last modified.

ls -a --- lists all files, including the ones whose filenames begin in a dot, which you do not always want to see.

There are many more options, for example to list files by size, by date, recursively etc.

cp *filename1 filename2* --- copies a file

rm *filename* --- removes a file. It is wise to use the option rm -i, which will ask you for confirmation before actually deleting anything. You can make this your default by making an alias in your .cshrc file.

diff *filename1 filename2* --- compares files, and shows where they differ

wc *filename* --- tells you how many lines, words, and characters there are in a file

chmod *options filename* --- lets you change the read, write, and execute permissions on your files. The default is that only you can look at them and change them, but you may sometimes want to change these permissions. For example, **chmod o+r *filename*** will make the file readable for everyone, and **chmod o-r *filename*** will make it unreadable for others again. Note that for someone to be able to actually look at the file the directories it is in need to be at least executable. See help protection for more details.

Directories (command -- description)

Directories, like folders on a Macintosh, are used to group files together in a hierarchical structure.

mkdir *dirname* --- make a new directory

cd *dirname* --- change directory. You basically 'go' to another directory, and you will see the files in that directory when you do 'ls'. You always start out in your 'home directory', and you can get back there by typing 'cd' without arguments. 'cd ..' will get you one level up from your current position. You don't have to walk along step by step - you can make big leaps or avoid walking around by specifying pathnames.

pwd --- tells you where you currently are.

Finding things (command -- description)

ff --- find files anywhere on the system. This can be extremely useful if you've forgotten in which directory you put a file, but do remember the name. In fact, if you use **ff -p** you don't even need the full name, just the beginning. This can also be useful for finding other things on the system, e.g. documentation.

grep *string filename(s)* --- looks for the string in the files. This can be useful a lot of purposes, e.g. finding the right file among many, figuring out which is the right version of something, and even doing serious corpus work. grep comes in several varieties (**grep**, **egrep**, and **fgrep**) and has a lot of very flexible options. Check out the man pages if this sounds good to you.

About other people (command -- description)

w --- tells you who's logged in, and what they're doing. Especially useful: the 'idle' part. This allows you to see whether they're actually sitting there typing away at their keyboards right at the moment.

who --- tells you who's logged on, and where they're coming from. Useful if you're looking for someone who's actually physically in the same building as you, or in some other particular location.

finger *username* --- gives you lots of information about that user, e.g. when they last read their mail and whether they're logged in. Often people put other practical information, such as phone numbers and addresses, in a file called **.plan**. This information is also displayed by 'finger'.

•**last -1** ***username*** --- tells you when the user last logged on and off and from where. Without any options, **last** will give you a list of everyone's logins.

•**talk** ***username*** --- lets you have a (typed) conversation with another user

write ***username*** --- lets you exchange one-line messages with another user

elm --- lets you send e-mail messages to people around the world (and, of course, read them). It's not the only mailer you can use, but the one we recommend. See the elm page, and find out about the departmental mailing lists (which you can also find in /user/linguistics/helpfile).

About your (electronic) self

whoami --- returns your username. Sounds useless, but isn't. You may need to find out who it is who forgot to log out somewhere, and make sure *you* have logged out.

passwd --- lets you change your password, which you should do regularly (at least once a year). See the LRB guide and/or look at help password.

ps -u ***yourusername*** --- lists your processes. Contains lots of information about them, including the process ID, which you need if you have to kill a process. Normally, when you have been kicked out of a dialin session or have otherwise managed to get yourself disconnected abruptly, this list will contain the processes you need to kill. Those may include the shell (tcsh or whatever you're using), and anything you were running, for example emacs or elm. Be careful not to kill your current shell - the one with the number closer to the one of the ps command you're currently running. But if it happens, don't panic. Just try again :) If you're using an X-display you may have to kill some X processes before you can start them again. These will show only when you use **ps -efl**, because they're root processes.

kill ***PID*** --- kills (ends) the processes with the ID you gave. This works only for your own processes, of course. Get the ID by using **ps**. If the process doesn't 'die' properly, use the option -9. But attempt without that option first, because it doesn't give the process

a chance to finish possibly important business before dying. You may need to kill processes for example if your modem connection was interrupted and you didn't get logged out properly, which sometimes happens.

2. Write programs using the following UNIX operating system calls

Fork

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
        fork();
        fork();
        fork();
        printf("hello\n");
        return 0;
}
```

output:

```
hello
hello
hello
hello
hello
hello
hello
hello
```

exec

file name : EXEC.c

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

int main()
{
        int i;
        printf("I am EXEC.c called by execvp() ");
        printf("\n");
```

```
        return 0;
}

// compile but don't run

file name : execDemo.c

#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
int main()
{
        char *args[]={"./EXEC",NULL};
        execvp(args[0],args);

        return 0;
}
```

output:

```
I AM EXEC.c called by execvp()
```

getpid

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

int main(void){

    pid_t thisPID = getpid();
    pid_t parentPID = getppid();

    printf("PID of current process: %d\n", thisPID);
    printf("PID of parent process: %d\n", parentPID);

    return 0;

}
```

output:

```
PID of current process: 156636
PID of parent process: 152284
```

exit

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

```
FILE *stream;

int main(void)
{
  if ((stream = fopen("myfile.dat", "r")) == NULL)
  {
    printf("Could not open data file\n");
    exit(EXIT_FAILURE);
    }
  printf("End of program -- not executable print statement if exit
runs");
}
```

output:

```
Could not open data file
```

wait

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

int main()
{
        pid_t cpid;
        if (fork()== 0)
                exit(0);
        else
                cpid = wait(NULL);
        printf("Parent pid = %d\n", getpid());
        printf("Child pid = %d\n", cpid);

        return 0;
}
```

output:

```
Parent pid = 157483
Child pid = 157484
```

stat

```
#include<stdio.h>
#include<sys/stat.h>
int main()
{
```

```
  //pointer to stat struct
  struct stat sfile;

  //stat system call
  stat("stat.c", &sfile);

  //accessing st_mode (data member of stat struct)
  printf("st_mode = %o", sfile.st_mode);
  return 0;
}
```

output:

st_mode = 100664

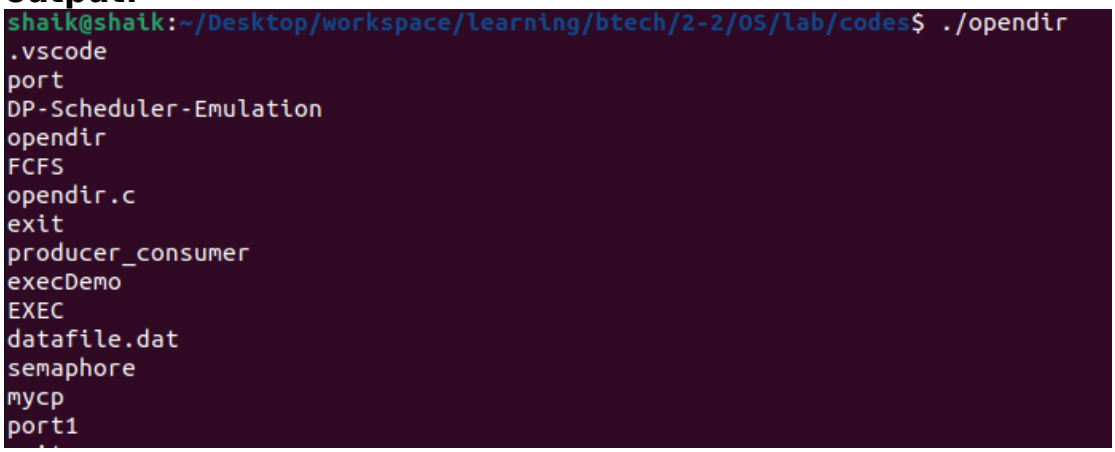
opendir and readdir

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

int main(void)
{
        struct dirent *de;
        DIR *dr = opendir(".");
        if (dr == NULL)
        {
                printf("Could not open current directory" );
                return 0;
        }
        while ((de = readdir(dr)) != NULL)
                printf("%s\n", de->d_name);

        closedir(dr);
        return 0;
}
```

output:



3. Simulate UNIX commands like cp, ls, grep, etc.,

cp:

```
#include <stdio.h>
#include<fcntl.h>
#include<unistd.h>
#include<sys/stat.h>
#include <string.h>
#define BUF_SIZE 32
#define FILE_NAME_LEN 200
int main(int argc, char *argv[])
{
FILE * file_to_read;
FILE * file_to_write;
char name_of_file_to_read[FILE_NAME_LEN+1];
char name_of_file_to_write[FILE_NAME_LEN+1];
char buf[BUF_SIZE];
size_t num_rec;
if(argc>3 || argc<3)
{
printf("Please Provide two arugments \n");
}
else{
if(access(argv[1],F_OK)<0)
{
printf("%s not found \n ",argv[1]);
}
/* Prepare the source file name */
strcpy(name_of_file_to_read, argv[1]);
/* Prepare the target file name */
if ( argc == 3 )
strcpy(name_of_file_to_write, argv[2]);
else
strcat(strcpy(name_of_file_to_write, name_of_file_to_read), ".fread");
/* Open source file in read-only mode */
if ( (file_to_read = fopen(name_of_file_to_read, "r")) == NULL )
{
fprintf(stderr, "Could not open file '%s' for reading\
n",name_of_file_to_read);
return 3;
}
/* Open target file in write mode */
if ( (file_to_write = fopen(name_of_file_to_write, "w")) == NULL )
{
fprintf(stderr, "Could not open file '%s' for writing\
n",name_of_file_to_write);
fclose(file_to_read);
return 4;
```

```
}
while ( (num_rec = fread(buf, sizeof(char), BUF_SIZE, file_to_read) )
> 0 )
{
fwrite(buf, sizeof(char), num_rec, file_to_write);
if ( ferror(file_to_write) )
{
fprintf(stderr, "Error while writing into file '%s'\n",
name_of_file_to_write);
fclose(file_to_read);
fclose(file_to_write);
return 5;
}
}
if ( ferror(file_to_read) )
{
fprintf(stderr, "Error while reading the file '%s'\n",
name_of_file_to_read);
fclose(file_to_read);
fclose(file_to_write);
return 6;
}
/* Close the files */
fclose(file_to_read);
fclose(file_to_write);
printf("File '%s' successfully copied to file '%s'\n",
name_of_file_to_read,
name_of_file_to_write);
return 1;
}
}
```

output:

```
shaik@shaik:~/Desktop/workspace/learning/btech/2-2/OS/lab/codes$ nano mycp.c
shaik@shaik:~/Desktop/workspace/learning/btech/2-2/OS/lab/codes$ gcc mycp.c -o mycp
shaik@shaik:~/Desktop/workspace/learning/btech/2-2/OS/lab/codes$ ./mycp f1 file
File 'f1' successfully copied to file 'file'
```

ls

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

void _ls(const char *dir,int op_a,int op_l)
{
        struct dirent *d;
```

```
        DIR *dh = opendir(dir);
        if (!dh)
        {
                if (errno = ENOENT)
                {
                        perror("Directory doesn't exist");
                }
                else
                {
                        perror("Unable to read directory");
                }
                exit(EXIT_FAILURE);
        }
        while ((d = readdir(dh)) != NULL)
        {
                if (!op_a && d->d_name[0] == '.')
                        continue;
                printf("%s  ", d->d_name);
                if(op_l) printf("\n");
        }
        if(!op_l)
        printf("\n");
}
int main(int argc, const char *argv[])
{
        if (argc == 1)
        {
                _ls(".",0,0);
        }
        else if (argc == 2)
        {
                if (argv[1][0] == '-')
                {
                        int op_a = 0, op_l = 0;
                        char *p = (char*)(argv[1] + 1);
                        while(*p){
                                if(*p == 'a') op_a = 1;
                                else if(*p == 'l') op_l = 1;
                                else{
                                        perror("Option not available");
                                        exit(EXIT_FAILURE);
                                }
                                p++;
                        }
                        _ls(".",op_a,op_l);
                }
        }
        return 0;
}
```

output:

```
shaik@shaik:~/Desktop/workspace/learning/btech/2-2/OS/lab/codes$ nano ls.c
shaik@shaik:~/Desktop/workspace/learning/btech/2-2/OS/lab/codes$ gcc ls.c -o ls
shaik@shaik:~/Desktop/workspace/learning/btech/2-2/OS/lab/codes$ ./ls
DP-Scheduler-Emulation  opendir  FCFS  opendir.c  exit  execDemo  EXEC  datafile.dat  mycp  exit.c  Main.c  close.c  wait.c  pid  wait  SJF  sta
t  ls.c  file  f6  fork.c  cp  rr.c  SJF.c  fork  cp.c  rr  close1.c  execDemo.c  ls  pid.c  EXEC.c  mycp.c  FCFS.c  RoundRobin.c  Main  RoundRo
bin  f1  stat.c
shaik@shaik:~/Desktop/workspace/learning/btech/2-2/OS/lab/codes$
```

grep:

```
#include<stdio.h>
#include<dirent.h>
#include <string.h>
int main()
{
   char fn[10], pat[10], temp[200];
   FILE *fp;
   printf("\n Enter file name : ");
   scanf("%s", fn);
   printf("Enter the pattern: ");
   scanf("%s", pat);
   fp = fopen(fn, "r");
   while (!feof(fp))
   {
      fgets(temp, sizeof(fp), fp);
      if (strcmp(temp, pat))
         printf("%s", temp);
   }
   fclose(fp);
   return 1;
}
```

output:

```
shaik@shaik:~/Desktop/workspace/learning/btech/2-2/OS/lab/codes$ nano grep.c
shaik@shaik:~/Desktop/workspace/learning/btech/2-2/OS/lab/codes$ gcc grep.c -o grep
shaik@shaik:~/Desktop/workspace/learning/btech/2-2/OS/lab/codes$ ./grep

 Enter file name : f1
Enter the pattern: *
Hello world
orld
shaik@shaik:~/Desktop/workspace/learning/btech/2-2/OS/lab/codes$
```

4) Simulate the following CPU scheduling algorithms

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

A) Round Robin :

```
#include<stdio.h>
void main ()
{
int i, j, n, bu[10], wa[10], tat[10], t, ct[10], max;
float awt = 0, att = 0, temp = 0;
printf ("Enter the no of processes -- ");
scanf ("%d", &n);
for (i = 0; i < n; i++)
{
printf ("\nEnter Burst Time for process %d -- ", i + 1);
scanf ("%d", &bu[i]);
ct[i] = bu[i];
}
printf ("\nEnter the size of time slice -- ");
scanf ("%d", &t);
max = bu[0];
for (i = 1; i < n; i++)
if (max < bu[i])
max = bu[i];
for (j = 0; j < (max / t) + 1; j++)
for (i = 0; i < n; i++)
if (bu[i] != 0)
if (bu[i] <= t)
        {
tat[i] = temp + bu[i];
temp = temp + bu[i];
bu[i] = 0;
}
        else
        {
bu[i] = bu[i] - t;
temp = temp + t;
}
for (i = 0; i < n; i++)
{
wa[i] = tat[i] - ct[i];
att += tat[i];
awt += wa[i];
}
printf ("\nThe Average Turnaround time is -- %f", att / n);
printf ("\nThe Average Waiting time is -- %f ", awt / n);
printf ("\n\tPROCESS\t BURST TIME \t WAITING TIME\tTURNAROUND
TIME\n");
```

```
for (i = 0; i < n; i++)
printf ("\t%d \t %d \t\t %d \t\t %d \n", i + 1, ct[i], wa[i], tat[i]);
}
```

INPUT :

```
Enter the no of processes – 3
Enter Burst Time for process 1 –      24
Enter Burst Time for process 2 --     3
Enter Burst Time for process 3 --     3
Enter the size of time slice – 3
```

OUTPUT :

```
The Average Turnaround time is – 15.666667
The Average Waiting time is --       5.666667
PROCESS    BURST TIME        WAITING TIME     TURNAROUND TIME
1                  24                6                30
2                  3                 4                7
3                  3                 7                10
```

B) SJF

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

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

INPUT :

```
Enter the number of processes --     4
Enter Burst Time for Process 0 --     6
Enter Burst Time for Process 1 --     8
Enter Burst Time for Process 2 --     7
Enter Burst Time for Process 3 --     3
```

OUTPUT

PROCESS	BURST TIME	WAITING TIME	TURNAROUND TIME
P3	3	0	3
P0	6	3	9
P2	7	9	16
P1	8	16	24

Average Waiting Time -- 7.000000
Average Turnaround Time -- 13.000000

C) FCFS

```
#include<stdio.h>
#include<conio.h>
main()
{
int bt[20], wt[20], tat[20], i, n;
float wtavg, tatavg;
clrscr();
printf("\nEnter the number of processes -- ");
scanf("%d", &n);
for(i=0;i<n;i++)
{
```

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

INPUT :

```
Enter the number of processes --     3
Enter Burst Time for Process 0 --     24
Enter Burst Time for Process 1 --     3
Enter Burst Time for Process 2 --     3
```

OUTPUT:

```
PROCESS    BURST TIME        WAITING TIME      TURNAROUND TIME
P0                  24                  0                  24
P1                  3                   24                 27
P2                  3                   27                 30
Average Waiting Time--   17.000000
Average Turnaround Time --   27.000000
```

D) Priority

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

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

INPUT :

```
Enter the number of processes --  5
Enter the Burst Time & Priority of Process 0 --- 10      3
Enter the Burst Time & Priority of Process 1 --- 1       1
Enter the Burst Time & Priority of Process 2 --- 2       4
Enter the Burst Time & Priority of Process 3 --- 1       5
Enter the Burst Time & Priority of Process 4 --- 5       2
```

OUTPUT :

```
PROCESS    PRIORITY   BURST TIME  WAITING TIME      TURNAROUND
TIME
1                  1           1           0                 1
4                  2           5           1                 6
0                  3           10          6                 16
2                  4           2           16                18
3                  5           1           18                19
Average Waiting Time is ---    8.200000
```

Average Turnaround Time is --- 12.000000

5. Assume that there are five jobs with different weights ranging from 1 to 5. Implement round robin algorithm with time slice equivalent to weight.

```
#include<stdio.h>

void main()
{
   int i, NOP, sum=0,count=0, y, quant, wt=0, tat=0, at[10], bt[10],
temp[10];
   float avg_wt, avg_tat;
   printf(" Total number of process in the system: ");
   scanf("%d", &NOP);
   y = NOP;

for(i=0; i<NOP; i++)
{
printf("\n Enter the Arrival and Burst time of the Process[%d]\n",
i+1);
printf(" Arrival time is: \t");
scanf("%d", &at[i]);
printf(" \nBurst time is: \t");
scanf("%d", &bt[i]);
temp[i] = bt[i];
}
printf("Enter the Time Quantum for the process: \t");
scanf("%d", &quant);
printf("\n Process No \t\t Burst Time \t\t TAT \t\t Waiting Time ");
for(sum=0, i = 0; y!=0; )
{
if(temp[i] <= quant && temp[i] > 0)
{
   sum = sum + temp[i];
   temp[i] = 0;
   count=1;
   }
   else if(temp[i] > 0)
   {
      temp[i] = temp[i] - quant;
      sum = sum + quant;
   }
   if(temp[i]==0 && count==1)
   {
      y--;
```

```
        printf("\nProcess No[%d] \t\t %d\t\t\t\t %d\t\t\t %d", i+1, bt[i],
sum-at[i], sum-at[i]-bt[i]);
        wt = wt+sum-at[i]-bt[i];
        tat = tat+sum-at[i];
        count =0;
    }
    if(i==NOP-1)
    {
        i=0;
    }
    else if(at[i+1]<=sum)
    {
        i++;
    }
    else
    {
        i=0;
    }
}
avg_wt = wt * 1.0/NOP;
avg_tat = tat * 1.0/NOP;
printf("\n Average Turn Around Time: \t%f", avg_wt);
printf("\n Average Waiting Time: \t%f", avg_tat);
}
```

INPUT :

Total number of process in the system: 5

Enter the Arrival and Burst time of the Process[1]
Arrival time is: 0

Burst time is: 1

Enter the Arrival and Burst time of the Process[2]
Arrival time is: 0

Burst time is: 2

Enter the Arrival and Burst time of the Process[3]
Arrival time is: 0

Burst time is: 3

Enter the Arrival and Burst time of the Process[4]
Arrival time is: 0

Burst time is: 4

```
 Enter the Arrival and Burst time of the Process[5]
 Arrival time is:    0

Burst time is:      5
Enter the Time Quantum for the process: 5
```

OUTPUT :

```
 Process No              Burst Time              TAT          Waiting
Time
Process No[1]           1                       1                 0
Process No[2]           2                       3                 1
Process No[3]           3                       6                 3
Process No[4]           4                       10                6
Process No[5]           5                       15                10
 Average Turn Around Time:  4.000000
 Average Waiting Time:      7.000000
```

6. Simulate how parent and child processes use shared memory and address space.

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

int globalVar;

int main(void)
{
        int localVar = 0;
        int* p = (int*) malloc(2);
        pid_t childPID = fork();
        *p = 0;
        if (childPID >= 0)
        {
                if (childPID == 0)
                {
                        printf("\n Child Process Initial Value :: localVar"
                                " = %d, globalVar = %d", localVar,
                                globalVar);
                        localVar++;
                        globalVar++;
```

```
                int c = 500;
                printf("\n Child Process :: localVar = %d, "
                        "globalVar = %d", localVar, globalVar);
                printf("\n Address of malloced mem child = %p "
                        "and value is %d", p, *p);
                printf("\n lets change the value pointed my
malloc");

                *p = 50;
                printf("\n Address of malloced mem child = %p "
                        "and value is %d", p, *p);
                printf("\n lets change the value pointed my "
                        "malloc in child");

                *p = 200;
                printf("\n Address of malloced mem child = %p "
                        "and value is %d\n\n\n", p, *p);
        }
        else
        {
                printf("\n Parent process Initial Value :: "
                        "localVar = %d, globalVar = %d",
                        localVar, globalVar);

                localVar = 10;
                globalVar = 20;
                printf("\n Parent process :: localVar = %d,"
                        " globalVar = %d", localVar, globalVar);
                printf("\n Address of malloced mem parent= %p "
                        "and value is %d", p, *p);

                *p = 100;
                printf("\n Address of malloced mem parent= %p "
                        "and value is %d", p, *p);
                printf("\n lets change the value pointed my"
                                " malloc in child");
                *p = 400;
                printf("\n Address of malloced mem child = %p"
                        " and value is %d \n", p, *p);
        }
    }
    else
    {
        printf("\n Fork failed, quitting!!!!!!\n");
        return 1;
    }

    return 0;
}
```

output:

7) Simulate sleeping barbaer problem

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h> // The maximum number of customer
threads.
#define MAX_CUSTOMERS 25 // Function prototypes...
void *customer(void *num);
void *barber(void *);
void randwait(int secs);
sem_t waitingRoom;
sem_t barberPillow;
sem_t seatBelt;
int allDone = 0;
int main(int argc, char *argv[])
{
 pthread_t btid;
 pthread_t tid[MAX_CUSTOMERS];
 int i, x, numCustomers, numChairs; int Number[MAX_CUSTOMERS];
 printf("Maximum number of customers can only be 25. Enter
number of customers and chairs.\n");
 scanf("%d",&x);
 numCustomers = x;
 scanf("%d",&x);
 numChairs = x;
 if (numCustomers > MAX_CUSTOMERS) {
printf("The maximum number of Customers is %d.\n",
MAX_CUSTOMERS);
 system("PAUSE");
```

```
 return 0;
 }
 printf("A solution to the sleeping barber problem using
semaphores.\n");
 for (i = 0; i < MAX_CUSTOMERS; i++) {
 Number[i] = i;
 }
 sem_init(&waitingRoom, 0, numChairs);
 sem_init(&barberChair, 0, 1);
 sem_init(&barberPillow, 0, 0);
 sem_init(&seatBelt, 0, 0);
 pthread_create(&btid, NULL, barber, NULL);
 for (i = 0; i < numCustomers; i++) {
 pthread_create(&tid[i], NULL, customer, (void *)&Number[i]);
 }
 for (i = 0; i < numCustomers; i++) {
 pthread_join(tid[i],NULL);
 }
 sem_post(&barberPillow); // Wake the barber so he will exit.
 pthread_join(btid,NULL);
 system("PAUSE");
 return 0;
}
void *customer(void *number) {
 int num = *(int *)number;
 printf("Customer %d leaving for barber shop.\n", num);
 randwait(5);
 printf("Customer %d arrived at barber shop.\n", num);
sem_wait(&waitingRoom);
 printf("Customer %d entering waiting room.\n", num);
sem_wait(&barberChair);
 sem_post(&waitingRoom);
 printf("Customer %d waking the barber.\n", num);
 sem_post(&barberPillow);
 sem_wait(&seatBelt); // Give up the chair.
 sem_post(&barberChair);
 printf("Customer %d leaving barber shop.\n", num);
}
void *barber(void *junk)
{
 while (!allDone) {
 printf("The barber is sleeping\n");
 sem_wait(&barberPillow);
 if (!allDone)
 {
 printf("The barber is cutting hair\n");
 randwait(3);
 printf("The barber has finished cutting hair.\n");
sem_post(&seatBelt);
```

```
 }
 else {
 printf("The barber is going home for the day.\n");
 }
 }
}

void randwait(int secs) {
 int len = 1; // Generate an arbit number...
 sleep(len);
}
```

Output:

```
A solution to the sleeping barber problem using semaphores.
Maximum number of customers can only be 25.
Enter number of customers and chairs.
4
4
The barber is sleeping
The barber is cutting hair
The barber has finished cutting hair.
Customer 1 leaving for barber shop.
Customer 2 arrived at barber shop.
Customer 2 entering waiting room.
The barber is sleeping
Customer 2 wakingup the barber
The barber is cutting hair
The barber has finished cutting hair.
Customer 2 leaving barber shop.
Customer 3 arrived at barber shop.
Customer 3 entering waiting room.
The barber is sleeping
Customer 3 wakingup the barber
The barber is cutting hair
The barber has finished cutting hair.
```

8) Simulate dining philosopher’s problem

```
#include<stdio.h>
#include<stdlib.h>
int tph, philname[20], status[20], howhung, hu[20], cho;

void one()
{
  int pos = 0, x, i;
  printf ("\nAllow one philosopher to eat at any time\n");
  for (i = 0; i < howhung; i++, pos++)
    {
      printf ("\nP %d is granted to eat", philname[hu[pos]]);
      for (x = pos; x < howhung; x++)
         printf ("\nP %d is waiting", philname[hu[x]]);
    }
}

void two()
{
  int i, j, s = 0, t, r, x;
  printf ("\n Allow two philosophers to eat at same time\n");
  for (i = 0; i < howhung; i++)
    {
      for (j = i + 1; j < howhung; j++)
         {
           if (abs(hu[i] - hu[j]) >= 1 && abs(hu[i] - hu[j]) != 4)
             {
               printf ("\n\ncombination %d \n", (s + 1));
               t = hu[i];
               r = hu[j];
               s++;
            printf ("\nP %d and P %d are granted to eat",
                          philname[hu[i]], philname[hu[j]]);
               for (x = 0; x < howhung; x++)
                  {
                    if ((hu[x] != t) && (hu[x] != r))
                      printf ("\nP %d is waiting", philname[hu[x]]);
                  }
             }
         }
    }
}

void main()
{
  int i;
  printf("\n\nDINING PHILOSOPHER PROBLEM");
  printf("\nEnter the total no. of philosophers: ");
```

```
 scanf("%d", &tph);
 for(i = 0; i < tph; i++)
   {
    philname[i] = (i + 1);
    status[i] = 1;
   }
 printf("How many are hungry : ");
 scanf("%d", &howhung);
 if (howhung == tph)
   {
    printf ("\nAll are hungry..\nDead lock stage will occur");
    printf ("\nExiting..");
   }
 else
   {
    for (i = 0; i < howhung; i++)
       {
        printf ("Enter philosopher %d position: ", (i + 1));
        scanf ("%d", &hu[i]);
        status[hu[i]] = 2;
       }
    do
       {
        printf
          ("1.One can eat at a time\t2.Two can eat at a time\t3.Exit\
nEnter your choice:");
        scanf ("%d", &cho);
        switch (cho)
          {
          case 1:
             one();
             break;
          case 2:
         two();
         break;
          case 3:
              exit(0);
          default:
              printf ("\nInvalid option..");
          }
       }
    while(1);
   }
}
```

OUTPUT:

```
DINING PHILOSOPHER PROBLEM
Enter the total no. of philosophers: 5
```

How many are hungry : 3
Enter philosopher 1 position: 2
Enter philosopher 2 position: 4
Enter philosopher 3 position: 5

1.One can eat at a time 2.Two can eat at a time 3.Exit
Enter your choice: 1
Allow one philosopher to eat at any time
P 3 is granted to eat
P 3 is waiting
P 5 is waiting
P 0 is waiting
P 5 is granted to eat
P 5 is waiting
P 0 is waiting
P 0 is granted to eat
P 0 is waiting
1. One can eat at a time 2.Two can eat at a time 3.Exit Enter your choice: 2
Allow two philosophers to eat at same time
combination 1
P 3 and P 5 are granted to eat
P 0 is waiting
combination 2
P 3 and P 0 are granted to eat
P 5 is waiting
combination 3
P 5 and P 0 are granted to eat
P 3 is waiting
1.One can eat at a time 2.Two can eat at a time 3.Exit Enter your choice: 3

9) Simulate producer-consumer problem using threads.

```
#include <stdio.h>
#include <stdlib.h>
int mutex = 1;
int full = 0;
int empty = 10, x = 0;
void producer()
{
        --mutex;
        ++full;
        --empty;
        x++;
        printf("\nProducer produces"
                "item %d",
```

```
                x);
        ++mutex;
}

void consumer()
{
        --mutex;
        --full;
        ++empty;
        printf("\nConsumer consumes "
                "item %d",
                x);
        x--;
        ++mutex;
}
int main()
{
        int n, i;
        printf("\n1. Press 1 for Producer"
                "\n2. Press 2 for Consumer"
                "\n3. Press 3 for Exit");

#pragma omp critical

        for (i = 1; i > 0; i++) {

                printf("\nEnter your choice:");
                scanf("%d", &n);
                switch (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;
            }
        }
}
```

output:

```
shaik@shaik:~/Desktop/workspace/learning/btech/2-2/OS/lab/codes$ nano producer_consumer.c
shaik@shaik:~/Desktop/workspace/learning/btech/2-2/OS/lab/codes$ gcc producer_consumer.c -o producer_consumer
shaik@shaik:~/Desktop/workspace/learning/btech/2-2/OS/lab/codes$ ./producer_consumer

1. Press 1 for Producer
2. Press 2 for Consumer
3. Press 3 for Exit
Enter your choice:2
Buffer is empty!
Enter your choice:1

Producer producesitem 1
Enter your choice:1

Producer producesitem 2
Enter your choice:1

Producer producesitem 3
Enter your choice:2

Consumer consumes item 3
Enter your choice:1

Producer producesitem 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
shaik@shaik:~/Desktop/workspace/learning/btech/2-2/OS/lab/codes$
```

10) Implement the following memory allocation methods for fixed partition

a) first fit b) worst fit c) best fit

a) 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;
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) 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,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

c) 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

11) Simulate the following page replacement algorithms a) FIFO b) LRU c) LFU etc.,

A) FIFO

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

```
count=0;
}
printf("\n The number of Page Faults using FIFO are %d",pf);
getch();
}
```

OUTPUT:

```
Enter the length of reference string -- 10
Enter the reference string -- 2 3 4 5 2 3 6 3 1 8
Enter no. of frames -- 3
The Page Replacement Process is --
 2 -1 -1 PF No. 1
 2 3 -1 PF No. 2
 2 3 4 PF No. 3
 5 3 4 PF No. 4
 5 2 4 PF No. 5
 5 2 3 PF No. 6
 6 2 3 PF No. 7
 6 2 3
 6 1 3 PF No. 8
 6 1 8 PF No. 9
The number of Page Faults using FIFO are 9
```

B) LRU

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

```
{
for(j=0;j<f;j++)
{
if(m[j]==rs[i])
{
flag[i]=1;
count[j]=next;
next++;
}
}
if(flag[i]==0)
{
if(i<f)
{
m[i]=rs[i];
count[i]=next;
next++;
}
else
{
min=0;
for(j=1;j<f;j++)
if(count[min] > count[j])
min=j;
m[min]=rs[i];
count[min]=next;
next++;
}
pf++;
}
for(j=0;j<f;j++)
printf("%d\t", m[j]);
if(flag[i]==0)
printf("PF No. -- %d" , pf);
printf("\n");
}
printf("\nThe number of page faults using LRU are %d",pf);
getch();
}
```

OUTPUT:

```
Enter the length of reference string -- 10
Enter the reference string -- 2 3 4 5 2 7 8 1 2 4
Enter the number of frames -- 3
The Page Replacement process is --
2 -1 -1 PF No. -- 1
2 3 -1 PF No. -- 2
2 3 4 PF No. -- 3
```

5 3 4 PF No. -- 4
5 2 4 PF No. -- 5
5 2 7 PF No. -- 6
8 2 7 PF No. -- 7
8 1 7 PF No. -- 8
8 1 2 PF No. -- 9
4 1 2 PF No. -- 10
The number of page faults using LRU are 10

C) LFU

```
#include<stdio.h>
#include<conio.h>
main()
{
 int rs[50], i, j, k, m, f, cntr[20], a[20], min, pf=0;
 clrscr();

 printf("\nEnter number of page references -- ");
 scanf("%d",&m);

 printf("\nEnter the reference string -- ");
 for(i=0;i<m;i++)
   scanf("%d",&rs[i]);

 printf("\nEnter the available no. of frames -- ");
 scanf("%d",&f);

 for(i=0;i<f;i++)
  {
    cntr[i]=0;
    a[i]=-1;
  }
 printf("\nThe Page Replacement Process is – \n");
 for(i=0;i<m;i++)
 {
   for(j=0;j<f;j++)
     if(rs[i]==a[j])
     {
       cntr[j]++;
       break;
       }
      if(j==f)
      {     min = 0;
        for(k=1;k<f;k++)
            if(cntr[k]<cntr[min])
            min=k;
            a[min]=rs[i];
            cntr[min]=1;
```

```
            pf++;
        }
          printf("\n");
          for(j=0;j<f;j++)
          printf("\t%d",a[j]);
          if(j==f)
             printf("\tPF No. %d",pf);
    }
      printf("\n\n Total number of page faults -- %d",pf);
     getch();
}
```

INPUT

```
Enter number of page references -- 10
Enter the reference string -- 1 2 3 4 5 2 5 2 5 1 4 3
 Enter the available no. of frames -- 3
```

OUTPUT

The Page Replacement Process is –

```
1 -1 -1 PF No. 1
1 2 -1 PF No. 2
1 2 3 PF No. 3
4 2 3 PF No. 4
5 2 3 PF No. 5
5 2 3
5 2 3
5 2 1 PF No. 6
5 2 4 PF No. 7
5 2 3 PF No. 8
```

Total number of page faults -- 8

12) Simulate Paging Technique of memory management

```
#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];
clrscr();
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();
}
```

INPUT

Enter the memory size – 1000 Enter the page size -- 100
The no. of pages available in memory are -- 10
Enter number of processes -- 3
Enter no. of pages required for p[1]-- 4
Enter pagetable for p[1] --- 8 6 9 5
Enter no. of pages required for p[2]-- 5
Enter pagetable for p[2] --- 1 4 5 7 3
Enter no. of pages required for p[3]-- 5

OUTPUT

Memory is Full
Enter Logical Address to find Physical Address Enter process no. and pagenumber and offset -- 2
3
60
The Physical Address is -- 760

13) Simulate Bankers Algorithm for Dead Lock avoidance and prevention

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

        printf("\nEnter New Request Details -- ");
        printf("\nEnter pid \t -- \t");
        scanf("%d",&id);
        printf("Enter Request for Resources \t -- \t");
        for(i=0;i<r;i++)
        {
                scanf("%d",&newr);
f[id].all[i] += newr;
                avail[i]=avail[i] - newr;
        }

        for(i=0;i<n;i++)
```

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

        }
```

```
        }
        printf("\nSYSTEM IS IN SAFE STATE");
        printf("\nThe Safe Sequence is -- (");
        for(i=0;i<fl;i++)
                printf("P%d ",seq[i]);
        printf(")");
      y:        printf("\nProcess\t\tAllocation\t\tMax\t\t\tNeed\n");
        for(i=0;i<n;i++)
        {
                printf("P%d\t",i);
                for(j=0;j<r;j++)
                        printf("%6d",f[i].all[j]);
                for(j=0;j<r;j++)
                        printf("%6d",f[i].max[j]);
                for(j=0;j<r;j++)
                        printf("%6d",f[i].need[j]);
                printf("\n");
        }
        getch();
}
```

INPUT

```
Enter number of
processes                 –       5
Enter number of
resources                 --      3
Enter details for
P0
Enter
allocation          --    0       1       0
Enter
Max                 --          7       5       3
Enter details for
P1
Enter
allocation          --    2       0       0
Enter
Max                 --    3       2       2
Enter details for
P2
Enter
allocation          --    3       0       2
Enter
Max                 --    9       0       2
Enter details for
P3
Enter
allocation          --    2       1       1
Enter               --    2       2       2
Max
```

```
Enter details for
P4
Enter
allocation          --    0     0     2
Enter
Max                 --    4     3     3
Enter Available
Resources --          3  3     2
Enter New Request
Details --
Enter
pid       --  1
Enter Request for
Resources           --  1       0     2
```

OUTPUT

```
P1 is
visited(  5 3 2)
P3 is
visited(  7 4 3)
P4 is
visited(  7 4 5)
P0 is
visited(  7 5 5)
P2 is
visited( 10  5 7)
SYSTEM IS IN
SAFE STATE
The Safe Sequence is -- (P1
P3 P4 P0 P2 )
```

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

14) Simulate following file allocation strategies a) **Sequential b) Indexed c) Linked**

A) **Sequential**

```
#include<stdio.h>
#include<conio.h>
struct fileTable
{
char name[20];
```

```
        int sb, nob;
        }ft[30];
        void main()
        {
        int i, j, n;
        char s[20];
        clrscr();
        printf("Enter no of files   :");
        scanf("%d",&n);
        for(i=0;i<n;i++)
        {
printf("\nEnter file name %d   :",i+1);
scanf("%s",ft[i].name);
printf("Enter starting block of file %d :",i+1);
scanf("%d",&ft[i].sb);
printf("Enter no of blocks in file %d :",i+1);
scanf("%d",&ft[i].nob);
}
printf("\nEnter the file name to be searched -- ");
scanf("%s",s);
for(i=0;i<n;i++)
if(strcmp(s, ft[i].name)==0)
break;
if(i==n)
printf("\nFile Not Found");
else
{
printf("\nFILE NAME START BLOCK NO OF BLOCKS BLOCKS
OCCUPIED\n"); printf("\n%s\t\t%d\t\t%d\
t",ft[i].name,ft[i].sb,ft[i].nob); for(j=0;j<ft[i].nob;j++)
printf("%d, ",ft[i].sb+j);
}
getch();
}
```

INPUT:

```
        Enter no of files  :3
        Enter file name 1 :A
        Enter starting block of file 1 :85
        Enter no of blocks in file 1      :6

        Enter file name 2 :B
        Enter starting block of file 2 :102
        Enter no of blocks in file 2      :4

        Enter file name 3 :C
        Enter starting block of file 3 :60
        Enter no of blocks in file 3      :4
        Enter the file name to be searched -- B
```

Output

FILENAME	STARTBLOCK	NO OF BLOCKS	BLOCKS OCUPIED
B	102	4	102,103,104,105

B) ***LINKED***

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

struct fileTable
{
char name[20];
int nob;
struct block *sb;
}ft[30];

struct block
{
int bno;
struct block *next;
};

void main()
{
int i, j, n;
char s[20];
struct block *temp;
clrscr();
printf("Enter no of files:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("\nEnter file name %d:",i+1);
scanf("%s",ft[i].name);
printf("Enter no of blocks in file %d :",i+1);
scanf("%d",&ft[i].nob);
ft[i].sb=(struct block*)malloc(sizeof(struct block));
temp = ft[i].sb;
printf("Enter the blocks of the file    :");
scanf("%d",&temp->bno);
temp->next=NULL;
for(j=1;j<ft[i].nob;j++)
{
temp->next = (struct block*)malloc(sizeof(struct block));
temp = temp->next;
scanf("%d",&temp->bno);
```

```
}
temp->next = NULL;
}
printf("\nEnter the file name to be searched --  ");
scanf("%s",s);
for(i=0;i<n;i++)
if(strcmp(s, ft[i].name)==0)
break;
if(i==n)
printf("\nFile Not Found");
else
{
printf("\nFILE NAME NO OF BLOCKS  BLOCKS OCCUPIED");
printf("\n     %s\t\t%d\t",ft[i].name,ft[i].nob);
temp=ft[i].sb;
for(j=0;j<ft[i].nob;j++)
{
printf("%d □ ",temp->bno);
temp = temp->next;
}
}
getch();
}
```

INPUT:

Enter no of files :
2
Enter file 1 : A
Enter no of blocks in file 1 : 4
Enter the blocks of the file 1 : 12 23 9 4
Enter file 2 : G
Enter no of blocks in file 2 : 5
Enter the blocks of the file 2 : 88 77 66 55 44
Enter the file to be searched : G

OUTPUT :

FILE NAME	NO OF BLOCKS	BLOCKS OCCUPIED
G	5	88 □ 77□ 66□ 55□ 44

C) **INDEXED**

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

struct fileTable
{
char name[20];
int nob, blocks[30];
}ft[30];

void main()
{
        int i, j, n;
        char s[20];
        clrscr();
        printf("Enter no of files  :");
        scanf("%d",&n);
        for(i=0;i<n;i++)
        {
                printf("\nEnter file name %d   :",i+1);
                scanf("%s",ft[i].name);
                printf("Enter no of blocks in file %d :",i+1);
                scanf("%d",&ft[i].nob);
                printf("Enter the blocks of the file    :");
                for(j=0;j<ft[i].nob;j++)
                        scanf("%d",&ft[i].blocks[j]);
        }

        printf("\nEnter the file name to be searched -- ");
        scanf("%s",s);
        for(i=0;i<n;i++)
                if(strcmp(s, ft[i].name)==0)
                        break;
        if(i==n)
                printf("\nFile Not Found");
        else
        {
                printf("\nFILE NAME NO OF
                BLOCKS BLOCKS OCCUPIED");
                printf("\n %s\t\t%d\
                t",ft[i].name,ft[i].nob);
                for(j=0;j<ft[i].nob;j++)
                        printf("%d, ",ft[i].blocks[j]);
        }
        getch();
}
```

INPUT:

Enter no of files   :

```
2
Enter
file 1      : A
Enter no of blocks
in file 1                : 4
Enter the blocks of      : 12 23 9
the file 1               4
Enter
file 2      : G
Enter no of blocks
in file 2                : 5
Enter the blocks of      : 88 77 66
the file 2               55 44
Enter the file to be
searched  : G
OUTPUT
:
FILE          NO OF              BLOCKS
NAME          BLOCKS             OCCUPIED
                                     77      55  4
G                   5             88,    ,66,    ,  4
```

15) Simulate all File Organization Techniques

a) **Single level directory b) Two level c) Hierarchical d)DAG**

a) Single level directory

```
#include<stdlib.h>
#include<string.h>
#include<stdio.h>
struct
{
char dname[10],fname[10][10];
int fcnt;
}dir;
void main()
{
int i,ch;
char f[30];
dir.fcnt = 0;
printf("\nEnter name of directory -- ");
scanf("%s", dir.dname);
while(1)
{
printf("\n\n1. Create File\t2. Delete File\t3. Search File \n 4. Display
Files\t5. Exit\nEnter your choice -- ");
scanf("%d",&ch);
```

```
switch(ch)
{
case 1: printf("\nEnter the name of the file -- ");
scanf("%s",dir.fname[dir.fcnt]);
dir.fcnt++;
break;
case 2: printf("\nEnter the name of the file -- ");
scanf("%s",f);
for(i=0;i<dir.fcnt;i++)
{
if(strcmp(f, dir.fname[i])==0)
{
printf("File %s is deleted ",f);
strcpy(dir.fname[i],dir.fname[dir.fcnt-1]); break; } }
if(i==dir.fcnt) printf("File %s not found",f);
else
dir.fcnt--;
break;
case 3: printf("\nEnter the name of the file -- ");
scanf("%s",f);
for(i=0;i<dir.fcnt;i++)
{
if(strcmp(f, dir.fname[i])==0)
{
printf("File %s is found ", f);
break;
}
}
if(i==dir.fcnt)
printf("File %s not found",f);
break;
case 4: if(dir.fcnt==0)
printf("\nDirectory Empty");
else
{
printf("\nThe Files are -- ");
for(i=0;i<dir.fcnt;i++)
printf("\t%s",dir.fname[i]);
}
break;
default: exit(0);
}
}
}
```

Output:

```
Enter name of directory -- CSE
```

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

4. Display Files 5. Exit

Enter your choice -- 1

Enter the name of the file -- A

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

4. Display Files 5. Exit

Enter your choice -- 1

Enter the name of the file -- B

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

4. Display Files 5. Exit

Enter your choice -- 1

Enter the name of the file -- C

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

4. Display Files 5. Exit

Enter your choice -- 3

Enter the name of the file -- ABC

File ABC not found

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

4. Display Files 5. Exit

Enter your choice -- 2

Enter the name of the file -- B

File B is deleted

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

4. Display Files 5. Exit

Enter your choice -- 5

b) Two level

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

struct
{
   char dname[10],fname[10][10];
   int fcnt;
}dir[10];

void main()
{
   int i,ch,dcnt,k;
   char f[30], d[30];
   clrscr();
   dcnt=0;

   while(1)
   {
      printf("\n\n1. Create Directory\t2. Create File\t3. Delete File");
      printf("\n4. Search File\t\t5. Display\t6. Exit\tEnter your choice
--");
   scanf("%d",&ch);

   switch(ch)
   {
      case 1: printf("\nEnter name of directory -- ");
            scanf("%s", dir[dcnt].dname);
            dir[dcnt].fcnt=0;
            dcnt++;
            printf("Directory created");
         break;

      case 2: printf("\nEnter name of the directory -- ");
            scanf("%s",d);
            for(i=0;i<dcnt;i++)
               if(strcmp(d,dir[i].dname)==0)
               {
                  printf("Enter name of the file -- ");
                  scanf("%s",dir[i].fname[dir[i].fcnt]);
                  dir[i].fcnt++;
                  printf("File created");
                  break;
               }
            if(i==dcnt)
```

```
            printf("Directory %s not found",d);
        break;

case 3: printf("\nEnter name of the directory -- ");
        scanf("%s",d);
        for(i=0;i<dcnt;i++)
        {
            if(strcmp(d,dir[i].dname)==0)
            {
                printf("Enter name of the file -- ");
                scanf("%s",f);
                for(k=0;k<dir[i].fcnt;k++)
                {
                    if(strcmp(f, dir[i].fname[k])==0)
                    {
                        printf("File %s is deleted ",f);
                        dir[i].fcnt--;
                        strcpy(dir[i].fname[k],dir[i].fname[dir[i].fcnt]);
                        goto jmp;
                    }
                }

            printf("File %s not found",f);
            goto jmp;
        }
        }

        printf("Directory %s not found",d);
        jmp : break;

case 4: printf("\nEnter name of the directory -- ");
        scanf("%s",d);
        for(i=0;i<dcnt;i++)
        {
            if(strcmp(d,dir[i].dname)==0)
            {
                printf("Enter the name of the file -- ");
                scanf("%s",f);
                for(k=0;k<dir[i].fcnt;k++)
                {
                    if(strcmp(f, dir[i].fname[k])==0)
                    {
                        printf("File %s is found ",f);
                        goto jmp1;
                    }
                }
                printf("File %s not found",f);
                goto jmp1;
            }
```

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

OUTPUT:

1. Create Directory 2. Create File 3. Delete File

4. Search File 5. Display 6. Exit Enter your choice --1

Enter name of directory -- DIR1

Directory created

1. Create Directory 2. Create File 3. Delete File

4. Search File 5. Display 6. Exit Enter your choice --1

Enter name of directory -- DIR2

Directory created

1. Create Directory 2. Create File 3. Delete File

4. Search File 5. Display 6. Exit Enter your choice --2

Enter name of the directory -- DIR1

Enter name of the file -- A1

File created

1. Create Directory 2. Create File 3. Delete File

4. Search File 5. Display 6. Exit Enter your choice --2

Enter name of the directory -- DIR1

Enter name of the file -- A2

File created

1. Create Directory 2. Create File 3. Delete File

4. Search File 5. Display 6. Exit Enter your choice --2

Enter name of the directory -- DIR2

Enter name of the file -- B1

File created

1. Create Directory 2. Create File 3. Delete File

4. Search File 5. Display 6. Exit Enter your choice --5

Directory Files

DIR1 A1 A2

DIR2 B1

1. Create Directory 2. Create File 3. Delete File

4. Search File 5. Display 6. Exit Enter your choice --4

Enter name of the directory -- DIR

Directory DIR not found

1. Create Directory 2. Create File 3. Delete File

4. Search File 5. Display 6. Exit Enter your choice --3

Enter name of the directory -- DIR1

Enter name of the file -- A2

File A2 is deleted

1. Create Directory 2. Create File 3. Delete File

4. Search File 5. Display 6. Exit Enter your choice --6

c) Herachical

```
#include<stdio.h>
#include<graphics.h>
#include<string.h>
struct tree_element
{
char name[20];
int x, y, ftype, lx, rx, nc, level;
struct tree_element *link[5];
};
typedef struct tree_element node;
void main()
{
int gd=DETECT,gm;
node *root;
root=NULL;
create(&root,0,"root",0,639,320);
clrscr();
initgraph(&gd,&gm,"c:\tc\BGI");
display(root);
closegraph();
}
create(node **root,int lev,char *dname,int lx,int rx,int x)
{
int i, gap;
if(*root==NULL)
{
(*root)=(node *)malloc(sizeof(node));
printf("Enter name of dir/file(under %s) : ",dname);
fflush(stdin);
gets((*root)->name);
printf("enter 1 for Dir/2 for file :");
scanf("%d",&(*root)->ftype);
(*root)->level=lev;
(*root)->y=50+lev*50;
(*root)->x=x;
(*root)->lx=lx;
(*root)->rx=rx;
for(i=0;i<5;i++)
(*root)->link[i]=NULL;
if((*root)->ftype==1)
{
```

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

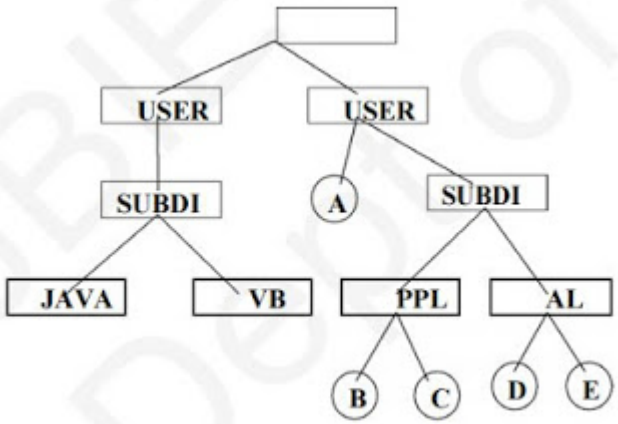
OUTPUT:

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

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

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

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



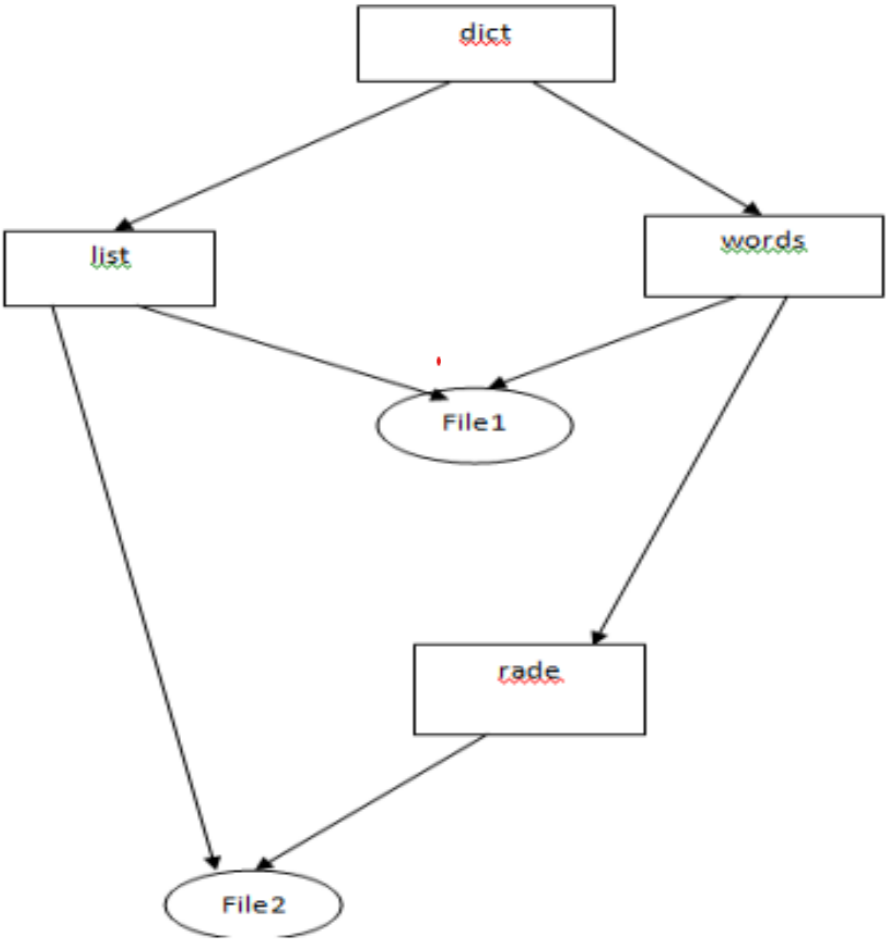
D) DAG:

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<string.h>
struct tree_element
{
char name[20];
int x,y,ftype,lx,rx,nc,level;
struct tree_element *link[5];
};
typedef struct tree_element
node;
typedef struct
{
char from[20];
char to[20];
}link;
Link L[10];
lnt nofl;
node * root;
void main()
{
int gd=DETECT,gm;
root=NULL;
clrscr();
create(&root,0,"root",0,639,320);
read_links();
clrscr();
initgraph(&gd,&gm,"c:\\tc\\BGI");
draw_link_lines();
display(root);
getch();
closegraph();
}
read_links()
{
int i;
printf("how many links");
scanf("%d",&nofl);
for(i=0;i<nofl;i++)
{
printf("File/dir:");
fflush(stdin);
gets(L[i].from);
printf("user name:");
```

```
fflush(stdin);
gets(L[i].to);
}
}
draw_link_lines()
{
int i,x1,y1,x2,y2;
for(i=0;i<nofl;i++)
{
search(root,L[i].from,&x1,&y1);
search(root,L[i].to,&x2,&y2);
setcolor(LIGHTGREEN);
setlinestyle(3,0,1);
line(x1,y1,x2,y2);
setcolor(YELLOW);
setlinestyle(0,0,1);
}
}
search(node *root,char *s,int *x,int *y)
{
int i;
if(root!=NULL)
{
if(strcmpi(root->name,s)==0)
{
*x=root->x;
*y=root->y;
return;
}
else
{
for(i=0;i<root->nc;i++)
search(root->link[i],s,x,y);
}
}
} create(node **root,int lev,char *
dname,int lx,int rx,int x)
{
int i,gap;
if(*root==NULL)
{
(*root)=(node *)malloc(sizeof(node));
printf("enter name of dir/file(under %s):",dname);
fflush(stdin);
gets((*root)->name);
printf("enter 1 for dir/ 2 for file:");
scanf("%d",&(*root)- >ftype);
(*root)->level=lev;
(*root)->y=50+lev*50;
```

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

Input & output:


dict
list
words
File1
rade
File2