PROGRAM - 1) FORK SYSTEM CALL PROGRAM:

```c
#include<stdio.h>
#include<stdlib.h>
int main( )
{
int pid;
pid=fork( );
printf("\n THIS LINE EXECUTED TWICE");
if(pid==-1) {
    printf("\n CHILD PROCESS NOT CREATED\n");
    exit(0);
}
else if(pid==0) {
    printf("\n I AM CHILD PROCESS AND MY ID IS %d \n",getpid( ));
    printf("\n THE CHILD PARENT PROCESS ID IS:%d \n",getppid( ));

}else{
printf("\n I AM PARENT PROCESS AND MY ID IS:%d\n",getpid( ));
printf("\n THE PARENTS PARENT PROCESS ID IS:%d\n",getppid( ));
}
printf("\n IT CAN BE EXECUTED TWICE");
printf("\n");
}
```

OUTPUT:

```
THIS LINE EXECUTED TWICE
I AM PARENT PROCESS AND MY ID IS:50414

THE PARENTS PARENT PROCESS ID IS:50407

IT CAN BE EXECUTED TWICE

THIS LINE EXECUTED TWICE
I AM CHILD PROCESS AND MY ID IS 50415

THE CHILD PARENT PROCESS ID IS:50414

IT CAN BE EXECUTED TWICE
```

PROGRAM - 2) SYSTEM CALL WITH ARGUMENT PROGRAM:

```c
#include<stdio.h>
#include<stdlib.h>
int main(int argc,int *argv[])
{
int pid,i;
pid=fork( );
printf("\n THIS LINE EXECUTED TWICE");
if(pid==-1) {
printf("\n CHILD PROCESS NOT CREATED\n");
exit(0);
}
if(pid==0) {
    printf("\n CHILD PROCESS IS IN PROGRESS\n");
    for(i=0;i<5;i++){
        printf("\n THE CHILD PROCESSING VALUE IS:%d \n",i);
    }
    execvp("ls",argv);
}
else {
    printf("\n PARENT PROCESS IS IN WAITING\n");
    printf("\n CHILD PROCESS COMPLETED ITS TASK\n");
}
exit(0);
}
```

OUTPUT – 2:

```
THIS LINE EXECUTED TWICE
PARENT PROCESS IS IN WAITING


CHILD PROCESS COMPLETED ITS TASK
THIS LINE EXECUTED TWICE
CHILD PROCESS IS IN PROGRESS

THE CHILD PROCESSING VALUE IS:0

THE CHILD PROCESSING VALUE IS:1

THE CHILD PROCESSING VALUE IS:2

THE CHILD PROCESSING VALUE IS:3

THE CHILD PROCESSING VALUE IS:4
```

PROGRAM - 3)WORKING WITH DIRECTORY PROGRAM:

```c
#include<stdio.h>
#include<sys/types.h>
#include<sys/dir.h>
#include<stdlib.h> // Include stdlib.h for exit()

int main(int argc,char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "Usage: %s <directory>\n", argv[0]);
        exit(1); // Exit with error status
    }

    DIR *dir;
    struct dirent *rddir;
    printf("LISTING THE DIRECTORY CONTENT\n");
    dir = opendir(argv[1]);
    if (dir == NULL) {
        perror("opendir");
        exit(1); // Exit with error status
    }

    printf("THE CURRENT DIRECTORY FILES ARE:\n");
    while ((rddir = readdir(dir)) != NULL) {
        printf("%s\n", rddir->d_name);
    }

    closedir(dir);
    return 0; // Exit normally
}
```

Output

```
/tmp/FvmLTlihMC.o
Usage: /tmp/FvmLTlihMC.o <directory>
```

PROGRAM - 4)STAT SYSTEM CALL PROGRAM:

```c
#include<stdio.h>
#include<sys/stat.h>
int main( ){
    struct stat sfile;
    stat("stat.c",&sfile);
    printf("file st_uid:%d\n",sfile.st_uid);
    printf("file st_uid:%d\n",sfile.st_gid);
    printf("file st_size:%ld\n",sfile.st_size);
    printf("file st_blocks:%ld\n",sfile.st_blocks);
    printf("file serialno:%ld\n",sfile.st_ino);
    printf("file recent access time :%ld\n",sfile.st_atime);
    printf("file permission change time:%ld\n",sfile.st_ctime);
    printf("file recent modified time:%ld\n",sfile.st_mtime);
}
```

Output

```
/tmp/kVY97ztLzx.o
file st_uid:0
file st_uid:0
file st_size:0
file st_blocks:0
file serialno:0
file recent access time :286052357
file permission change time:0
file recent modified time:4198997
```

PROGRAM – 3A

```c
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[]) {
    if (argc != 3) {
        fprintf(stderr, "Usage: %s <source_file> <destination_file>\n", argv[0]);
        return 1;
    }
    FILE *fp1 = fopen(argv[1], "r");
    FILE *fp2 = fopen(argv[2], "w");
    if (!fp1 || !fp2) {
        perror("Error");
        return 1;
    }
    char ln[80];
    while (fgets(ln, sizeof(ln), fp1)) {
        fputs(ln, fp2);
    }
    printf("FILE HAS BEEN COPIED SUCCESSFULLY\n");
    fclose(fp1);
    fclose(fp2);
    return 0;
}
```

Output

```
/tmp/HJXyZ81Ko4.o
Usage: /tmp/HJXyZ81Ko4.o <source_file> <destination_file>
```

PROGRAM – 3B - GREP COMMAND

```c
#include<stdio.h>
#include<stdlib.h>
int strmat(char ln[], char pa[]) {
    int i, j, k;
    for (i = 0; ln[i] != '\0'; i++) {
        for (j = i, k = 0; ln[j] == pa[k] && pa[k] != '\0'; j++, k++);
        if (k > 0 && pa[k] == '\0')
            return 1;
    }
    return 0;
}
int main(int argc, char *argv[]) {
    FILE *fp1;
    char ln[80];
    if (argc < 3) {
        printf("USAGE: %s SOURCE_FILE SEARCH_PATTERN\n", argv[0]);
        return 1;
    }
    fp1 = fopen(argv[1], "r");
    if (fp1 == NULL) {
        printf("Error: Unable to open source file.\n");
        return 1;
    }
    while (fgets(ln, 80, fp1)) {
        if (strmat(ln, argv[2]))
            printf("%s", ln);
    }
    fclose(fp1);
    return 0;
}
```

Output

```
/tmp/U7XfrPDpO7.o
USAGE: /tmp/U7XfrPDpO7.o SOURCE_FILE SEARCH_PATTERN
```

EXP - 4 - PROGRAM - 1. GREATEST AMONG THREE NUMBER:

```
#!/bin/bash

echo "ENTER THREE NUMBERS"
read a b c

if [ $a -gt $b ] && [ $a -gt $c ]; then
    echo "$a is greater"
elif [ $b -gt $c ]; then
    echo "$b is greater"
else
    echo "$c is greater"
fi
```

OUPUT:

```
ENTER THREE NUMBERS
5 9 3
5 is greater
```

PROGRAM - 2.FACTORIAL OF A GIVEN NUMBER:

```
echo "ENTER THE NUMBER:"
read n
fact=1
while [ $n -
gt 1 ] do
fact=`expr $fact \* $n |
bc` n=`expr $n - 1`
done
echo "FACTORIAL OF THE GIVEN NUMBER IS $fact"
```

OUTPUT:

```
ENTER THE NUMBER:
5
FACTORIAL OF THE GIVEN NUMBER IS 120
```

PROGRAM - 3.SUM OF ODD NUMBERS UPTO N:

```
echo "Enter the range"
read n
x=1
sum=0
while [ $x -le $n ]; do
    sum=`expr $sum + $x`
    x=`expr $x + 2`
done
echo "Sum of odd numbers in the range is: $sum"
```

OUTPUT:

```
Enter the range
10
Sum of odd numbers in the range is: 25
```

PROGRAM - 4.GENERATION OF FIBONACCI NUMBERS:

```
echo "ENTER THE LIMIT:"
read n
p=0
q=1
i=1
while [ $i -le $n ]; do
    r=`expr $p + $q`
    p=$q
    q=$r
    echo "$r"
    i=`expr $i + 1`
done
```

OUPUT:

```
ENTER THE LIMIT:
5
1
1
2
3
5
```

PROGRAM – 5A - FCFS ALGORITHM:

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

int main() {
    int n, pid[10], at[10], bt[10], ft[10], wt[10], ta[10], i, j, t, stt = 0, totta = 0, totwt = 0;
    float avgta, avgwt;
    printf("ENTER THE NUMBER OF PROCESSES: ");
    scanf("%d", &n);
    for(i = 1; i <= n; i++) {
        pid[i] = i;
        printf("\nENTER THE ARRIVAL TIME FOR PROCESS %d: ", i);
        scanf("%d", &at[i]);
        printf("ENTER THE BURST TIME FOR PROCESS %d: ", i);
        scanf("%d", &bt[i]);
    }
    for(i = 1; i <= n; i++) {
        for(j = i + 1; j <= n; j++) {
            if(at[i] > at[j]) {
                t = pid[i]; pid[i] = pid[j]; pid[j] = t;
                t = at[i]; at[i] = at[j]; at[j] = t;
                t = bt[i]; bt[i] = bt[j]; bt[j] = t;
            }
        }stt = at[1];
    }
    printf("\nTHE VALUE OF THE FIRST ARRIVAL TIME IS %d", stt);

    for(i = 1; i <= n; i++) {
        ft[i] = stt + bt[i];
        wt[i] = stt - at[i];
        ta[i] = ft[i] - at[i];
        totta += ta[i];
        totwt += wt[i];
        stt = ft[i];
    }
    avgta = (float)totta / n;
    avgwt = (float)totwt / n;
    printf("\nPNO\tARRIVAL TIME\tBURST TIME\tCOMPLETION TIME\tWAITING TIME\tTURNAROUND TIME\n");
    for(i = 1; i <= n; i++)
        printf("%d\t\t%d\t\t\t%d\t\t\t%d\t\t\t%d\t\t\t%d\n", pid[i], at[i], bt[i], ft[i], wt[i], ta[i]);
    printf("\nAVERAGE TURNAROUND TIME = %f", avgta);
    printf("\nAVERAGE WAITING TIME = %f\n", avgwt);

    return 0;
}
```

OUTPUT:

Output

```
/tmp/c2iFEcoGZh.o
ENTER THE NUMBER OF PROCESSES: 4

ENTER THE ARRIVAL TIME FOR PROCESS 1: 1
ENTER THE BURST TIME FOR PROCESS 1: 2

ENTER THE ARRIVAL TIME FOR PROCESS 2: 3
ENTER THE BURST TIME FOR PROCESS 2: 4

ENTER THE ARRIVAL TIME FOR PROCESS 3: 3
ENTER THE BURST TIME FOR PROCESS 3: 2

ENTER THE ARRIVAL TIME FOR PROCESS 4: 1
ENTER THE BURST TIME FOR PROCESS 4: 3

THE VALUE OF THE FIRST ARRIVAL TIME IS 1
PNO ARRIVAL TIME    BURST TIME  COMPLETION TIME WAITING TIME    TURNAROUND TIME
1       1               2               3               0               2
4       1               3               6               2               5
3       3               2               8               3               5
2       3               4               12              5               9

AVERAGE TURNAROUND TIME = 5.250000
AVERAGE WAITING TIME = 2.500000
```

PROGRAM – 5B - Shortest Job First(NON-PRE-EMPTIVE SCHEDULING)

```c
#include<stdio.h>
int main() {
    int i, j, t, n, stt = 0, pid[10], at[10], bt[10], ft[10],
    att, wt[10], ta[10], totwt = 0, totta = 0;
    float avgwt, avgta;
    printf("ENTER THE NUMBER OF PROCESSES: ");
    scanf("%d", &n);
    printf("\nENTER THE ARRIVAL TIME: ");
    scanf("%d", &att);
    for(i = 1; i <= n; i++) {
        pid[i] = i;
        at[i] = att;
        printf("\nENTER THE BURST TIME FOR PROCESS %d: ", i);
        scanf("%d", &bt[i]);
    }
    for(i = 1; i <= n; i++) {
        for(j = i + 1; j <= n; j++) {
            if(bt[i] > bt[j]) {
                t = pid[i]; pid[i] = pid[j]; pid[j] = t;
                t = bt[i]; bt[i] = bt[j]; bt[j] = t;
    }}}
    stt = att;
    for(i = 1; i <= n; i++) {
        ft[i] = stt + bt[i];
        wt[i] = stt - at[i];
        ta[i] = ft[i] - at[i];
        totta += ta[i];
        totwt += wt[i];
        stt = ft[i];
    }
    avgwt = (float)totwt / n;
    avgta = (float)totta / n;
    printf("\nPNO\tARRIVAL TIME\tBURST TIME\tCOMPLETION TIME\tWAIT TIME\tTAT\n");
    for(i = 1; i <= n; i++) {
        printf("%d\t\t%d\t\t%d\t\t%d\t\t\t%d\t\t\t%d\n", pid[i], at[i], bt[i],
               ft[i], wt[i], ta[i]);
    }
    printf("\nAVERAGE TURNAROUND TIME = %f", avgta);
    printf("\nAVERAGE WAITING TIME = %f\n", avgwt);
    return 0;
}
```

OUTPUT:

Output

```
/tmp/xjr4n55Kl5.o
ENTER THE NUMBER OF PROCESSES: 4

ENTER THE ARRIVAL TIME: 0

ENTER THE BURST TIME FOR PROCESS 1: 3

ENTER THE BURST TIME FOR PROCESS 2: 4

ENTER THE BURST TIME FOR PROCESS 3: 5

ENTER THE BURST TIME FOR PROCESS 4: 6

PNO ARRIVAL TIME    BURST TIME  COMPLETION TIME     WAIT TIME   TURN AROUND TIME
1       0               3           3                   0               3
2       0               4           7                   3               7
3       0               5           12                  7               12
4       0               6           18                  12              18

AVERAGE TURNAROUND TIME = 10.000000
AVERAGE WAITING TIME = 5.500000
```

PROGRAM – 5C - Priority Scheduling

```c
#include<stdio.h>
int main() {
    int i, j, t, n, stt = 0, pid[10], pr[10], at[10], bt[10], ft[10], att;
    int wt[10], ta[10], totwt = 0, totta = 0;
    float avgwt, avgta;
    printf("ENTER THE NUMBER OF PROCESSES: ");
    scanf("%d", &n);
    printf("\nENTER THE ARRIVAL TIME: ");
    scanf("%d", &att);
    for(i = 1; i <= n; i++) {
        pid[i] = i;
        at[i] = att;
    }
    printf("\nENTER THE BURST TIME AND PRIORITY OF EACH PROCESS:\n");
    for(i = 1; i <= n; i++) {
        printf("PROCESS %d:\n", i);
        printf("BURST TIME: ");
        scanf("%d", &bt[i]);
        printf("PRIORITY: ");
        scanf("%d", &pr[i]);
    }
    for(i = 1; i <= n; i++) {
        for(j = i + 1; j <= n; j++) {
            if(pr[i] > pr[j]) {
                t = pid[i]; pid[i] = pid[j]; pid[j] = t;
                t = bt[i]; bt[i] = bt[j]; bt[j] = t;
                t = pr[i]; pr[i] = pr[j]; pr[j] = t;
        }}stt = att;
    }
    for(i = 1; i <= n; i++) {
        ft[i] = stt + bt[i];
        wt[i] = stt - at[i];
        ta[i] = ft[i] - at[i];
        if(wt[i] < 0)
            wt[i] = 0;
        totwt += wt[i];
        totta += ta[i];
        stt = ft[i];
    }
    avgwt = (float)totwt / n;
    avgta = (float)totta / n;
    printf("\nPNO\tARRIVAL TIME\tBURST TIME\tFINISH TIME\tWAIT TIME\tTURNAROUND TIME\n");
    for(i = 1; i <= n; i++) {
        printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n", pid[i], at[i], bt[i],
               ft[i], wt[i], ta[i]);
    }
    printf("\nTHE AVERAGE WAITING TIME IS: %f\n", avgwt);
    printf("THE AVERAGE TURNAROUND TIME IS: %f\n", avgta);
    return 0;
}
```

Output

```
/tmp/S1MAF9cmTr.o
ENTER THE NUMBER OF PROCESSES: 3

ENTER THE ARRIVAL TIME: 0

ENTER THE BURST TIME AND PRIORITY OF EACH PROCESS:
PROCESS 1:
BURST TIME: 3
PRIORITY: 2
PROCESS 2:
BURST TIME: 4
PRIORITY: 1
PROCESS 3:
BURST TIME: 2
PRIORITY: 3

PNO ARRIVAL TIME    BURST TIME  FINISH TIME     WAIT TIME   TURNAROUND TIME
2       0               4           4               0           4
1       0               3           7               4           7
3       0               2           9               7           9

THE AVERAGE WAITING TIME IS: 3.666667
THE AVERAGE TURNAROUND TIME IS: 6.666667
```

PROGRAM – 5D ROUND ROBIN SCHEDULING

```c
#include<stdio.h>
int main() {
    int i, j, n, wt[10], ta[10], at[10], bt[10], tot_wt = 0, tot_ta = 0, ft[10], t;
    int s[10], prid[10], p[10], max = 0, temp, stt = 0, ts = 0, x = 0;
    float avg_wt, avg_ta;
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    for(i = 1; i <= n; i++) {
        prid[i] = i;
        printf("\nEnter the arrival time of process %d: ", i);
        scanf("%d", &at[i]);
        printf("Enter the burst time of process %d: ", i);
        scanf("%d", &bt[i]);
        wt[i] = 0;
        p[i] = 0;
        if(at[i] > max) {
            max = at[i];
        }}
    max = at[i];
    printf("\nEnter the time slice: ");
    scanf("%d", &ts);
    for(i = 1; i < n; i++) {
        for(j = i + 1; j <= n; j++) {
            if(at[i] > at[j]) {
                t = at[i];
                at[i] = at[j];
                at[j] = t;

                t = bt[i];
                bt[j] = bt[i];
                bt[i] = t;
    }}}
    for(i = 1; i <= n; i++) {
        i = 1;
        x = 0;
        while(x < n) {
            s[i] = bt[i];
            if(p[i] == 1)
                continue;
            if(at[i] > stt) {
                temp = max;
                for(i = 1; i <= n; i++) {
                    if(p[i] == 0 && at[i] <= temp) {
                        temp = at[i];
                }}
                if(temp > stt) {
                    stt = temp;
                }
           if(s[i] > ts) {
               s[i] = s[i] - ts;
               stt = stt + ts;
               stt = stt + s[i];
               ft[i] = stt;
               s[i] = 0;
           } else {
               s[i] = s[i] - ts;
           }
           con:
               i++;
           if(i > n)
               p[i] = 1;
           x++;
           i = 1;
    }}
    for(i = 1; i <= n; i++) {
        ta[i] = ft[i] - at[i];
        wt[i] = ta[i] - bt[i];
        tot_ta += ta[i];
        tot_wt += wt[i];
    }
    avg_wt = (float)tot_wt / n;
    avg_ta = (float)tot_ta / n;
    printf("\nPNO\tARR TIME\tBURST TIME\tWAIT TIME\tTURN TIME\tFINISH TIME\n");
    for(i = 1; i <= n; i++) {
        printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n", i, at[i], bt[i], wt[i], ta[i], ft[i]);
    }
    printf("\nThe Average Waiting Time is: %0.2f\n", avg_wt);
    printf("The Average Turnaround Time is: %0.2f\n", avg_ta);
    return 0;
}
```

OUTPUT:

```
Enter the number of processes: 3

Enter the arrival time of process 1: 0
Enter the burst time of process 1: 5

Enter the arrival time of process 2: 1
Enter the burst time of process 2: 3

Enter the arrival time of process 3: 2
Enter the burst time of process 3: 7

Enter the time slice: 2

PNO     ARR TIME    BURST TIME  WAIT TIME   TURN TIME   FINISH TIME
1       0           5           2           7           10
2       1           3           6           8           9
3       2           7           8           18          20

The Average Waiting Time is: 5.33
The Average Turnaround Time is: 11.00
```

PROGRAM – 5 - IMPLEMENTATION OF SEMAPHORES

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <time.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#define num_loops 5
union semun {
    int val;
    struct semid_ds *buf;
    unsigned short *array;
};
int main(int argc, char *argv[]) {
    int semset_id;union semun sem_val;
    int child_pid,i,rc;struct sembuf sem_op;
    struct timespec delay;
    semset_id = semget(IPC_PRIVATE, 1, 0600);
    if (semset_id == -1) {
        perror("semget");
        exit(1);
    }
    printf("SEMAPHORE SET CREATED, SEMAPHORE SET ID: %d\n", semset_id);
    sem_val.val = 0;
    rc = semctl(semset_id, 0, SETVAL, sem_val);
    if (rc == -1) {
        perror("semctl");
        exit(1);
    }
    child_pid = fork();
    switch(child_pid) {
        case -1:
            perror("fork");
            exit(1);
        case 0: // Child process (Consumer)
            for(i = 0; i < num_loops; i++) {
                sem_op.sem_num = 0;
                sem_op.sem_op = -1;
                sem_op.sem_flg = 0;
                semop(semset_id, &sem_op, 1);
                printf("Consumer consumed item %d\n", i);
                fflush(stdout);
            }
            break;
        default: // Parent process (Producer)
            for(i = 0; i < num_loops; i++) {
                printf("Producer produced item %d\n", i);
                fflush(stdout);
                sem_op.sem_num = 0;
                sem_op.sem_op = 1;
                sem_op.sem_flg = 0;
                semop(semset_id, &sem_op, 1);

                if (rand() > 3 * (RAND_MAX / 4)) {
                    delay.tv_sec = 0;
                    delay.tv_nsec = 10000000; // 10 milliseconds
                    nanosleep(&delay, NULL);
             }}break;
    }
    return 0;
}
```

OUTPUT:

```
SEMAPHORE SET CREATED, SEMAPHORE SET ID: 123456
Producer produced item 0
Consumer consumed item 0
Producer produced item 1
Consumer consumed item 1
Producer produced item 2
Consumer consumed item 2
Producer produced item 3
Consumer consumed item 3
Producer produced item 4
Consumer consumed item 4
```

PROGRAM – 6 – BANKER ALGO

```c
#include<stdio.h>
int max[100][100],alloc[100][100],need[100][100],avail[100],n, r;
void input();
void show();
void cal();
int main() {
    printf("********** Deadlock Avoidance ***********\n");
    input();show();cal();
    return 0;
}
void input() {
    printf("Enter the number of Processes: ");
    scanf("%d", &n);
    printf("Enter the number of resource instances: ");
    scanf("%d", &r);
    printf("Enter the Max Matrix\n");
    for(int i = 0; i < n; i++) {
        for(int j = 0; j < r; j++) {
            scanf("%d", &max[i][j]);
    }}
    printf("Enter the Allocation Matrix\n");
    for(int i = 0; i < n; i++) {
        for(int j = 0; j < r; j++) {
            scanf("%d", &alloc[i][j]);
    }}
    printf("Enter the available Resources\n");
    for(int j = 0; j < r; j++) {
        scanf("%d", &avail[j]);
    }}
void show() {
    printf("Process\t Allocation\t Max\t Available\t\n");
    for(int i = 0; i < n; i++) {
        printf("P%d\t", i + 1);
        for(int j = 0; j < r; j++) {
            printf("%d ", alloc[i][j]);
        }
        printf("\t");
        for(int j = 0; j < r; j++) {
            printf("%d ", max[i][j]);
        }
        printf("\t");
        if(i == 0) {
            for(int j = 0; j < r; j++) {
                printf("%d ", avail[j]);
        }}
        printf("\n");
    }}
void cal() {
    int finish[100], temp, flag = 1, k, c1 = 0;
    int dead[100],safe[100],i, j;
    for(i = 0; i < n; i++)
        finish[i] = 0;
    for(i = 0; i < n; i++) {
        for(j = 0; j < r; j++)
            need[i][j] = max[i][j] - alloc[i][j];
    }
    while(flag) {
        flag = 0;
        for(i = 0; i < n; i++) {
            int c = 0;
            for(j = 0; j < r; j++) {
                if((finish[i] == 0) && (need[i][j] <= avail[j])) {
                    c++;
                    if(c == r) {
                        for(k = 0; k < r; k++) {
                            avail[k] += alloc[i][j];
                            finish[i] = 1;
                            flag = 1;
                        }
                        printf("P%d->", i);
                        if(finish[i] == 1) {
                            i = n;
        }}}}}}
```

```
    j = 0;
    flag = 0;
    for(i = 0; i < n; i++) {
        if(finish[i] == 0) {
            dead[j] = i;
            j++;
            flag = 1;
        }}
    if(flag == 1) {
        printf("\n\nSystem is in Deadlock and the Deadlocked processes are:\n");
        for(i = 0; i < n; i++)
            printf("P%d\t", dead[i]);
    } else {
        printf("\nNo Deadlock Occurred\n");
    }}
```

OUTPUT:

```
********** Deadlock Avoidance ************
Enter the number of Processes: 3
Enter the number of resource instances: 4
Enter the Max Matrix
7 5 3
3 2 2
9 0 2
Enter the Allocation Matrix
0 1 0
2 0 0
3 0 2
Enter the available Resources
3 3 2
Process  Allocation     Max     Available
P1       0 1 0   7 5 3   3 3 2
P2       2 0 0   3 2 2   3 3 2
P3       3 0 2   9 0 2   3 3 2
P2->P1->P3->

System is in Deadlock and the Deadlocked processes are:
P1 P2 P3
```

PROGRAM – 7 – DEADLOCK DETECTION

```c
#include<stdio.h>
int max[100][100],alloc[100][100],need[100][100],avail[100],n, r;
void input();
void show();
void cal();
int main() {
    printf("********** Deadlock Detection Algorithm ************\n");
    input();show();cal();
    return 0;
}
void input() {
    printf("Enter the number of Processes: ");
    scanf("%d", &n);
    printf("Enter the number of resource instances: ");
    scanf("%d", &r);
    printf("Enter the Max Matrix\n");
    for(int i = 0; i < n; i++) {
        for(int j = 0; j < r; j++) {
            scanf("%d", &max[i][j]);
    }}
    printf("Enter the Allocation Matrix\n");
    for(int i = 0; i < n; i++) {
        for(int j = 0; j < r; j++) {
            scanf("%d", &alloc[i][j]);
    }}

    printf("Enter the available Resources\n");
    for(int j = 0; j < r; j++) {
        scanf("%d", &avail[j]);
    }}
void show() {
    printf("Process\t Allocation\t Max\t Available\t\n");
    for(int i = 0; i < n; i++) {
        printf("P%d\t", i + 1);
        for(int j = 0; j < r; j++) {
            printf("%d ", alloc[i][j]);
        }
        printf("\t");
        for(int j = 0; j < r; j++) {
            printf("%d ", max[i][j]);
        }
        printf("\t");
        if(i == 0) {
            for(int j = 0; j < r; j++) {
                printf("%d ", avail[j]);
        }}
        printf("\n");
}}
void cal() {
    int finish[100], temp, flag = 1, k, c1 = 0;
    int dead[100];
    int safe[100];
    int i, j;

    for(i = 0; i < n; i++)
        finish[i] = 0;

    for(i = 0; i < n; i++) {
        for(j = 0; j < r; j++)
            need[i][j] = max[i][j] - alloc[i][j];
    }
```

```
    while(flag) {
        flag = 0;
        for(i = 0; i < n; i++) {
            int c = 0;
            for(j = 0; j < r; j++) {
                if((finish[i] == 0) && (need[i][j] <= avail[j])) {
                    c++;
                    if(c == r) {
                        for(k = 0; k < r; k++) {
                            avail[k] += alloc[i][j];
                            finish[i] = 1;
                            flag = 1;
                        }
                        printf("P%d->", i);
                        if(finish[i] == 1) {
                            i = n;
    }}}}}}

    j = 0;
    flag = 0;
    for(i = 0; i < n; i++) {
        if(finish[i] == 0) {
            dead[j] = i;
            j++;
            flag = 1;
    }}

    if(flag == 1) {
        printf("\n\nSystem is in Deadlock and the Deadlocked processes are:\n");
        for(i = 0; i < n; i++)
            printf("P%d\t", dead[i]);
    } else {
        printf("\nNo Deadlock Occurred\n");
        }}
```

OUTPUT:

```
********** Deadlock Detection Algorithm ************
Enter the number of Processes: 3
Enter the number of resource instances: 4
Enter the Max Matrix
7 5 3
3 2 2
9 0 2
Enter the Allocation Matrix
0 1 0
2 0 0
3 0 2
Enter the available Resources
3 3 2
Process  Allocation   Max   Available
P1       0 1 0       7 5 3   3 3 2
P2       2 0 0       3 2 2   3 3 2
P3       3 0 2       9 0 2   3 3 2

P0->P1->P2->
System is in Deadlock and the Deadlocked processes are:
P3
```

PROGRAM – 8 – THREADING

```c
#include <stdio.h>
#include <string.h>
#include <pthread.h>
#include <stdlib.h>
#include <unistd.h>
pthread_t tid[2];
int counter;
void* doSomeThing(void *arg) {
    unsigned long i = 0;
    counter += 1;
    printf("\nJob %d started\n", counter);
    for(i = 0; i < (0xFFFFFFFF); i++);
    printf("\nJob %d finished\n", counter);
    return NULL;
}
int main(void) {
    int i = 0,int err;
    while(i < 2) {
        err = pthread_create(&(tid[i]), NULL, &doSomeThing, NULL);
        if (err != 0)
            printf("\nCan't create thread :[%s]", strerror(err));
        i++;
    }
    pthread_join(tid[0], NULL);
    pthread_join(tid[1], NULL);
    return 0;
}
```

OUTPUT:

```
Job 1 started
Job 2 started

Job 1 finished
Job 2 finished
```

PROGRAM – 9- MEMORY ALLOCATION

CASE-1 -FIRST FIT

```c
#include <stdio.h>
int main() {
    int p[10], np, b[10], nb, ch, c[10], d[10], alloc[10], flag[10], i, j;

    printf("\nEnter the number of processes: ");
    scanf("%d", &np);
    printf("\nEnter the number of blocks: ");
    scanf("%d", &nb);
    printf("\nEnter the size of each process:\n");
    for (i = 0; i < np; i++) {
        printf("Process %d: ", i);
        scanf("%d", &p[i]);
    }
    printf("\nEnter the block sizes:\n");
    for (j = 0; j < nb; j++) {
        printf("Block %d: ", j);
        scanf("%d", &b[j]);
        c[j] = b[j];
        d[j] = b[j];
    }
    if (np <= nb) {
        printf("\n1. First fit\n2. Best fit\n3. Worst fit\n");

        do {
            printf("\nEnter your choice: ");
            scanf("%d", &ch);

            switch (ch) {
                case 1:
                    printf("\nFirst Fit\n");
                    for (i = 0; i < np; i++) {
                        for (j = 0; j < nb; j++) {
                            if (p[i] <= b[j]) {
                                alloc[j] = p[i];
                                printf("\n\nAlloc[%d]", alloc[j]);
                                printf("\nProcess %d of size %d is allocated in block %d of size %d", i, p[i], j, b[j]);
                                flag[i] = 0;
                                b[j] = 0;
                                break;
                    }}}
                    for (i = 0; i < np; i++) {
                        if (flag[i] != 0)
                            printf("\nProcess %d of size %d is not allocated", i, p[i]);
                    }
                    break;
```

CASE -2 – BEST FIT

```
case 2:
    printf("\nBest Fit\n");
    for (i = 0; i < nb; i++) {
        for (j = i + 1; j < nb; j++) {
            if (c[i] > c[j]) {
                int temp = c[i];
                c[i] = c[j];
                c[j] = temp;
            }
        }
    }
    printf("\nAfter sorting block sizes:\n");
    for (i = 0; i < nb; i++)
        printf("Block %d: %d\n", i, c[i]);

    for (i = 0; i < np; i++) {
        for (j = 0; j < nb; j++) {
            if (p[i] <= c[j]) {
                alloc[j] = p[i];
                printf("\n\nAlloc[%d]", alloc[j]);
                printf("\nProcess %d of size %d is allocated in block %d of size %d", i, p[i], j, c[j]);
                flag[i] = 0;
                c[j] = 0;
                break;
            }
        }
    }
    for (i = 0; i < np; i++) {
        if (flag[i] != 0)
            printf("\nProcess %d of size %d is not allocated", i, p[i]);
    }
    break;
```

CASE -3 - WORST FIT

```
                case 3:
                    printf("\nWorst Fit\n");
                    for (i = 0; i < nb; i++) {
                        for (j = i + 1; j < nb; j++) {
                            if (d[i] < d[j]) {
                                int temp = d[i];
                                d[i] = d[j];
                                d[j] = temp;
                    }}}
                    printf("\nAfter sorting block sizes:\n");
                    for (i = 0; i < nb; i++)
                        printf("Block %d: %d\n", i, d[i]);
                    for (i = 0; i < np; i++) {
                        for (j = 0; j < nb; j++) {
                            if (p[i] <= d[j]) {
                                alloc[j] = p[i];
                                printf("\n\nAlloc[%d]", alloc[j]);
                                printf("\nProcess %d of size %d is allocated in block %d of size %d", i, p[i], j, d[j]);
                                flag[i] = 0;
                                d[j] = 0;
                                break;
                    }}}
                    for (i = 0; i < np; i++) {
                        if (flag[i] != 0)
                            printf("\nProcess %d of size %d is not allocated", i, p[i]);
                    }
                    break;
                default:
                    printf("Invalid Choice…!");
                    break;
            }
        } while (ch <= 3);
    }return 0;
}
```

OUTPUT:

```
Enter the number of processes: 3
Enter the number of blocks: 3
Enter the size of each process:
Process 0: 5
Process 1: 2
Process 2: 3
Enter the block sizes:
Block 0: 4
Block 1: 5
Block 2: 3

1. First fit
2. Best fit
3. Worst fit

Enter your choice: 1

First Fit

Alloc[5]
Process 0 of size 5 is allocated in block 1 of size 5

Alloc[3]
Process 1 of size 2 is allocated in block 2 of size 3

Alloc[0]
Process 2 of size 3 is allocated in block 0 of size 4
Enter your choice: 2

Best Fit

After sorting block sizes:
Block 0: 3
Block 1: 4
Block 2: 5

Alloc[5]
Process 0 of size 5 is allocated in block 2 of size 5

Alloc[2]
Process 1 of size 2 is allocated in block 0 of size 3

Alloc[3]
Process 2 of size 3 is allocated in block 1 of size 4

Enter your choice: 3

Worst Fit

After sorting block sizes:
Block 0: 5
Block 1: 4
Block 2: 3

Alloc[5]
Process 0 of size 5 is allocated in block 0 of size 5

Alloc[3]
Process 1 of size 2 is allocated in block 1 of size 4

Alloc[2]
Process 2 of size 3 is allocated in block 2 of size 3
```

PROGRAM – 10 – PAGE REPLACEMENT ALGORITHM

PART A- FIFO page replacement algorithm

```c
#include<stdio.h>
int main() {
    int i, j, n, a[50], frame[10], no, k, avail, count = 0;
    printf("\nEnter the number of pages:\n");
    scanf("%d", &n);
    printf("\nEnter the page numbers:\n");
    for(i = 1; i <= n; i++)
        scanf("%d", &a[i]);
    printf("\nEnter the number of frames:");
    scanf("%d", &no);
    for(i = 0; i < no; i++)
        frame[i] = -1;
    j = 0;
    printf("\tRef String\tPage Frames\n");
    for(i = 1; i <= n; i++) {
        printf("%d\t\t", a[i]);
        avail = 0;
        for(k = 0; k < no; k++)
            if(frame[k] == a[i])
                avail = 1;
        if (avail == 0) {
            frame[j] = a[i];
            j = (j + 1) % no;
            count++;
            for(k = 0; k < no; k++)
                printf("%d\t", frame[k]);
        }
        printf("\n");
    }
    printf("Page faults: %d\n", count);
    return 0;
}
```

OUTPUT:

```
Enter the number of pages:
7

Enter the page numbers:
2 3 4 2 1 3 7

Enter the number of frames:3
    Ref String    Page Frames
2       2
3       2       3
4       2       3       4
2       2       3       4
1       1       3       4
3       1       3       4
7       1       7       4
Page faults: 5
```

PART – B – LRU page replacement

```c
#include<stdio.h>
int findLRU(int time[], int n) {
    int i, minimum = time[0], pos = 0;
    for(i = 1; i < n; ++i) {
        if(time[i] < minimum) {
            minimum = time[i];
            pos = i;
    }}
    return pos;
}
int main() {
    int no_of_frames, no_of_pages, frames[10], pages[30];
    int counter = 0, time[10], flag1, flag2, i, j, pos, faults = 0;
    printf("Enter number of frames: ");
    scanf("%d", &no_of_frames);
    printf("Enter number of pages: ");
    scanf("%d", &no_of_pages);
    printf("Enter reference string: ");
    for(i = 0; i < no_of_pages; ++i)
        scanf("%d", &pages[i]);
    for(i = 0; i < no_of_frames; ++i)
        frames[i] = -1;
    for(i = 0; i < no_of_pages; ++i) {
        flag1 = flag2 = 0;
        for(j = 0; j < no_of_frames; ++j) {
            if(frames[j] == pages[i]) {
                counter++;
                time[j] = counter;
                flag1 = flag2 = 1;
                break;
        }}
        if(flag1 == 0) {
            for(j = 0; j < no_of_frames; ++j) {
                if(frames[j] == -1) {
                    counter++;
                    faults++;
                    frames[j] = pages[i];
                    time[j] = counter;
                    flag2 = 1;
                    break;
        }}}
        if(flag1 == 0) {
            for(j = 0; j < no_of_frames; ++j) {
                if(frames[j] == -1) {
                    counter++;
                    faults++;
                    frames[j] = pages[i];
                    time[j] = counter;
                    flag2 = 1;
                    break;
        }}}
        if(flag2 == 0) {
            pos = findLRU(time, no_of_frames);
            counter++;
            faults++;
            frames[pos] = pages[i];
            time[pos] = counter;
        }
        printf("\n");
        for(j = 0; j < no_of_frames; ++j)
            printf("%d\t", frames[j]);
    }
    printf("\n\nTotal Page Faults = %d", faults);
    return 0;
}
```

OUTPUT:

```
Enter number of frames: 3
Enter number of pages: 10
Enter reference string: 1 2 3 4 5 1 6 7 8 9

1       -1      -1
1       2       -1
1       2       3
1       2       3
1       2       3
5       2       3
5       6       3
5       6       7
5       6       8
9       6       8

Total Page Faults = 8
```

PART – C - LFU page replacement algorithm

```c
#include<stdio.h>
int n;
int main()
{
    int seq[30], fr[5], pos[5], find, flag, max, i, j, m, k, t, s, pf = 0;
    int count = 1, p = 0;
    float pfr;
    printf("ENTER MAX LIMIT OF THE SEQUENCE:");
    scanf("%d", &max);
    printf("ENTER THE SEQUENCE:");
    for(i = 0; i < max; i++)
        scanf("%d", &seq[i]);
    printf("ENTER THE NO OF FRAMES:");
    scanf("%d", &n);
    fr[0] = seq[0];
    pf++;
    printf("%d\t", fr[0]);
    i = 1;
    while(count < n)
    {
        flag = 1;
        p++;
        for(j = 0; j < i; j++)
        {
            if(seq[i] == seq[j])
                flag = 0;
        }
        if(flag != 0)
        {
            fr[count] = seq[i];
            printf("%d\t", fr[count]);
        }
        i++;
        count++;
    }
    printf("\n");
    for(i = p; i < max; i++)
    {
        flag = 1;
        for(j = 0; j < n; j++)
        {
            if(seq[i] == fr[j])
                flag = 0;
        }
        if(flag != 0)
        {
            for(j = 0; j < n; j++)
            {
                m = fr[j];
                for(k = i; k < max; k++)
                {
                    if(seq[k] == m)
                        pos[j] = k;
                    else
                        pos[j] = -1;
                    break;
            }}
```

```
            for(k = 0; k < n; k++)
            {
                if(pos[k] == -1)
                    flag = 0;
            }
            if(flag != 0)
                s = findmax(pos);
            if(flag == 0)
            {
                for(k = 0; k < n; k++)
                {
                    if(pos[k] == -1)
                    {
                        pf++;
                        fr[s] = seq[i];
                        break;
            }}}
            for(k = 0; k < n; k++)
                printf("%d\t", fr[k]);
            printf("\n");
    }}
    pfr = (float)pf / (float)max;
    printf("\n THE NO.OF PAGE FAULTS ARE:%d", pf);
    printf("\n PAGE FAULT RATE:%f", pfr);
    return 0;
}
int findmax(int a[])
{
    int max, i, k = 0;
    max = a[0];
    for(i = 0; i < n; i++)
    {
        if(max < a[i])
        {
            return k;
}}}
```

OUTPUT:

```
ENTER MAX LIMIT OF THE SEQUENCE: 10
ENTER THE SEQUENCE: 2 4 5 2 1 7 4 6 9 0
ENTER THE NO OF FRAMES: 3

2       -1      -1
2       4       -1
2       4       5
4       7       5
1       7       5
1       4       5
6       4       5
6       9       5
6       9       0

THE NO.OF PAGE FAULTS ARE:6
PAGE FAULT RATE:0.600000
```

PROGRAM – 11- FILE ORGANIZATION TECHNIQUE – Part- A Single Level

```
#include<stdio.h>
#include<stdlib.h>
#include<string.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\n2. Delete File\n3. Search File);
        printf("\n4. Display Files\n5. 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;

            case 5:
                exit(0);
                break;

            default:
                printf("\nTerminate");
}}}
```

OUTPUT:

```
Enter name of directory -- my_directory

1. Create File
2. Delete File
3. Search File
4. Display Files
5. Exit
Enter your choice -- 1
Enter the name of the file -- file1

1. Create File
2. Delete File
3. Search File
4. Display Files
5. Exit
Enter your choice -- 1
Enter the name of the file -- file2

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

The Files are --     file1    file2

1. Create File
2. Delete File
3. Search File
4. Display Files
5. Exit
Enter your choice -- 3
Enter the name of the file -- file2
File file2 is found

1. Create File
2. Delete File
3. Search File
4. Display Files
5. Exit
Enter your choice -- 2
Enter the name of the file -- file2
File file2 is deleted

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

PROGRAM – B Two-level directory Structure

```c
#include<stdio.h>
struct {
    char dname[10], fname[10][10];
    int fcnt;
} dir[10];
void main( ) {
    int i, ch, dcnt, k;
    char f[30], d[30];
    clrscr( );dcnt = 0;
    while(1) {
        printf("\n\n1. Create Directory\n2. Create File\n3. Delete File");
        printf("4. Search File\n5. Display\n6. Exit\nEnter 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 -- Documents
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 -- Documents
Enter name of the file -- resume.txt
File created

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 -- Documents
Enter name of the file -- resume.txt
File resume.txt is deleted
```

PROGRAM – C -Hierarchical directory:

```c
#include<stdio.h>
#include<graphics.h>
struct tree_element
{
    char name[20];
    int x, y, ftype, lx, rx, nc, level;
    struct tree_element *link[5];
};
typedef struct tree_element node;
void 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);
        if (lev == 0 || lev == 1)
            (*root)->ftype = 1;
        else
            (*root)->ftype = 2;
        (*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)
        {
            if (lev == 0 || lev == 1)
            {
                if ((*root)->level == 0)
                    printf("How many users");
                else
                    printf("How many files");
                printf(" (for %s): ", (*root)->name);
                scanf("%d", &(*root)->nc);
            }
            else
                (*root)->nc = 0;
            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;
    }
    return;
}
```

```
void 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]);
        }
    }
    return;
}
```

PROGRAM – D- Directed Acyclic Graph (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];
int nofl;
node *root;
void 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);
}}
void 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);
    }
}

void 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);
}}}
void 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;
    }}
void 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]);
}}}
int 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();
    return 0;
}
```

PROGRAM – FILE ALLOCATION STRATEGIES - a. Sequential File Allocation:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main() {
    int st[20], b[20], b1[20], ch, i, j, n, blocks[20][20], sz[20];
    char F[20][20], S[20];clrscr();
    printf("\nEnter no. of Files :: ");
    scanf("%d", &n);
    for (i = 0; i < n; i++) {
        printf("\nEnter file %d name :: ", i + 1);
        scanf("%s", &F[i]);
        printf("\nEnter file %d size (in kb) :: ", i + 1);
        scanf("%d", &sz[i]);
        printf("\nEnter Starting block of %d :: ", i + 1);
        scanf("%d", &st[i]);
        printf("\nEnter blocksize of File %d (in bytes) :: ", i + 1);
        scanf("%d", &b[i]);
    }
    for (i = 0; i < n; i++)
        b1[i] = (sz[i] * 1024) / b[i];
    for (i = 0; i < n; i++) {
        for (j = 0; j < b1[i]; j++)
            blocks[i][j] = st[i] + j;
    }
    do {
        printf("\nEnter the Filename :: ");
        scanf("%s", S);
        for (i = 0; i < n; i++) {
            if (strcmp(S, F[i]) == 0) {
                printf("\nFname\tStart\tNblocks\tBlocks\n");
                printf("\n\n");
                printf("%s\t%d\t%d\t", F[i], st[i], b1[i]);
                for (j = 0; j < b1[i]; j++)
                    printf("%d->", blocks[i][j]);
        }}
        printf("\n\n");
        printf("Do you want to continue? (Y:1/n:0) :: ");
        scanf("%d", &ch);
        if (ch != 1)
            break;
    } while (1);
}
```

PROGRAM – B - Indexed File Allocation:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
int n;
void main() {
    int b[20], b1[20], i, j, blocks[20][20], sz[20];
    char F[20][20], S[20], ch;
    clrscr();
    printf("\nEnter no. of Files :: ");
    scanf("%d", &n);
    for (i = 0; i < n; i++) {
        printf("\nEnter file %d name :: ", i + 1);
        scanf("%s", &F[i]);
        printf("\nEnter file %d size (in kb) :: ", i + 1);
        scanf("%d", &sz[i]);
        printf("\nEnter blocksize of File %d (in bytes) :: ", i + 1);
        scanf("%d", &b[i]);
    }
    for (i = 0; i < n; i++) {
        b1[i] = (sz[i] * 1024) / b[i];
        printf("\n\nEnter blocks for file %d", i + 1);
        for (j = 0; j < b1[i]; j++) {
            printf("\nEnter the %d block :: ", j + 1);
            scanf("%d", &blocks[i][j]);
    }}
    do {
        printf("\nEnter the Filename :: ");
        scanf("%s", &S);
        for (i = 0; i < n; i++) {
            if (strcmp(F[i], S) == 0) {
                printf("\nFname\tFsize\tBsize\tNblocks\tBlocks\n");
                printf("\n\n");
                printf("%s\t%d\t%d\t%d\t", F[i], sz[i], b[i], b1[i]);
                for (j = 0; j < b1[i]; j++)
                    printf("%d->", blocks[i][j]);
        }}
        printf("\n\n");
        printf("Do you want to continue? (Y:1/n:0) :: ");
        scanf(" %c", &ch);
    } while (ch == 'Y' || ch == 'y');
}
```

PROGRAM – C – Linked File Allocation

```c
#include<stdio.h>
void main() {
    char a[10];
    int i, sb, eb, fb1[10];
    printf("\nEnter the file name: ");
    scanf("%s", a);
    printf("\nEnter the starting block: ");
    scanf("%d", &sb);
    printf("Enter the ending block: ");
    scanf("%d", &eb);
    for(i = 0; i < 5; i++) {
        printf("Enter the free block %d: ", i + 1);
        scanf("%d", &fb1[i]);
    }
    printf("\nFile name \tStarting block \tEnding block\n");
    printf("%s \t\t%d\t\t%d\n", a, sb, eb);
    printf("\n%s File Utilization of Linked type of following blocks:", a);
    printf("\n%d->", sb);
    for(i = 0; i < 5; i++) {
        printf("%d->", fb1[i]);
    }
    printf("%d\n", eb);
}
```

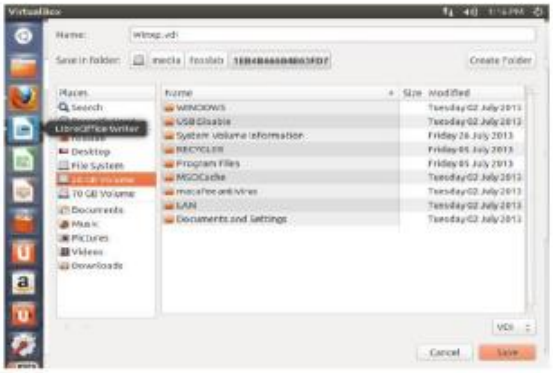

Welcome to the New Virtual Machine Wizard!


VM Name and OS Type


Memory


Welcome to the virtual disk creation wizard


Virtual disk storage details




Welcome to the First Run Wizard!


Virtual disk File location and size