//cp

```
#include<stdio.h>
#include<fcntl.h>
#include<unistd.h>
main(){
   int fdr,fdw,size=0;
   char buf[30];
   fdr=open("renu.txt",O_RDONLY);
   if(fdr<=0)
   {
   printf("File does not exist");
   }
   fdw=open("riki.txt",O_WRONLY|O_CREAT ,0777);
   while(size=read(fdr,buf,10))
      write(fdw,buf,size);
   printf("File copied");
}
```

//ls

```
#include<stdio.h>
#include<dirent.h>
int main(){
   DIR *p;
   char dirname[10];
   struct dirent *d;
   printf("Enter directory name: ");
   scanf("%s",dirname);
   p=opendir(dirname);
   if(p==NULL)
   {
      printf("Directory is not present");
      return 0;
   }
   while(d=readdir(p))
   {
      printf("%s\t",d->d_name);
   }

}
```

//ls-l

```
#include<sys/types.h>
#include<sys/stat.h>
```

```
#include<stdio.h>
#include<unistd.h>
#include<time.h>
#include<pwd.h>
#include<grp.h>
#include<stdlib.h>
int main(){
   struct stat st;
   struct passwd *pw;
   struct group *gr;
   if(stat("renu.txt",&st)<0)
   {
      printf("file doesnt exist");
      return 0;
   }
   if(S_ISREG(st.st_mode))
   {
      printf("-");
   }
   else if(S_ISDIR(st.st_mode))
   {
      printf("d");
   }
   else if(S_ISBLK(st.st_mode))
   {
      printf("b");
   }
   else if(S_ISCHR(st.st_mode))
   {
      printf("c");
   }
   else if(S_ISLNK(st.st_mode))
   {
      printf("l");
   }
   else if(S_ISSOCK(st.st_mode))
   {
      printf("s");
   }
   else if(S_ISFIFO(st.st_mode))
   {
      printf("p");
   }

   if(S_IRUSR &st.st_mode)
   {
      printf("r");
```

```
}
else
{
   printf("-");
}
if(S_IWUSR &st.st_mode)
{
   printf("w");
}
else
{
   printf("-");
}
if(S_IXUSR & st.st_mode)
{
   printf("x ");
}
else
{
   printf("- ");
}
if(S_IRGRP & st.st_mode)
{
   printf("r");
}
else
{
   printf("-");
}
if(S_IWGRP &st.st_mode)
{
   printf("w");
}
else
{
   printf("-");
}
if(S_IXGRP & st.st_mode)
{
   printf("x ");
}
else
{
   printf("- ");
}
if(S_IROTH &st.st_mode)
{
```

```
        printf("r");
    }
    else
    {
        printf("-");
    }
    if(S_IWOTH & st.st_mode)
    {
        printf("w");
    }
    else
    {
        printf("-");
    }
    if(S_IXOTH & st.st_mode)
    {
        printf("x ");
    }
    else
    {
        printf("- ");
    }
    printf("%d ",st.st_nlink);
    pw=getpwuid(st.st_uid);
    gr=getgrgid(st.st_gid);
    printf("%s ",pw->pw_name);
    printf("%s ",gr->gr_name);
    printf("%d ",st.st_size);
    printf("%s",ctime(&st.st_ctime));
    printf("\n");
    }


//fcfs

#include<stdio.h>
struct process{
int pid;
int at;
int burst;
int ct;
int wt;
int tat;
};
void calculate(struct process p[],int n);
void display(struct process p[],int n);
int main(){
```

```
    int n;
    printf("Enter the no.of processes:");
    scanf("%d",&n);
    struct process p[n];
    printf("Enter the arrival time,burst time for each process");
    for(int i=0;i<n;i++)
    {

        p[i].pid=i+1;
        printf("process %d:",i+1);
        scanf("%d %d",&p[i].at,&p[i].burst);
    }
        for(int i=0;i<n-1;i++)
        {
            for(int j=0;j<n-i-1;j++)
            {
                if(p[j].at>p[j+1].at)
                {
                    struct process temp=p[j];
                    p[j]=p[j+1];
                    p[j+1]=temp;
                }
            }
        }

calculate(p,n);
display(p,n);
return 0;
}
void calculate(struct process p[],int n)
{
    p[0].wt=0;
    p[0].ct=p[0].burst;
    p[0].tat=p[0].burst;
    for(int i=1;i<n;i++)
    {
        p[i].ct=p[i-1].ct+p[i].burst;
        p[i].tat=p[i].ct-p[i].at;
        p[i].wt=p[i].tat-p[i].burst;
    }
}
void display(struct process p[],int n)
{
    int tot_wt=0;
    int tot_tat=0;
    printf("id \t at \t bt \t ct \t tat \t wt");
    for(int i=0;i<n;i++)
```

```
    {
        printf("\n%d \t %d \t %d \t %d \t %d \t
%d",p[i].pid,p[i].at,p[i].burst,p[i].ct,p[i].tat,p[i].wt);
        tot_wt+=p[i].wt;
        tot_tat+=p[i].tat;
    }
    printf("\n avg waiting time : %f",(tot_wt)/(float)n);
    printf("\n avg waiting time : %f",(tot_tat)/(float)n);
}


//SJF

#include<stdio.h>
struct process{
    int pid;
    int at;
    int bt;
    int wt;
    int tat;
    int ct;
};
void calculate(struct process p[],int n);
void display(struct process p[], int n);
int main(){
    int n;
    printf("Enter the no. of processes:");
    scanf("%d",&n);
    struct process p[n];
    printf("enter the at and bt for rocesses:");
    for(int i=0;i<n;i++)
    {
        p[i].pid=i+1;
        printf("process %d",i+1);
        scanf("%d %d",&p[i].at,&p[i].bt);
    }
    for(int i=0;i<n-1;i++)
    {
        for(int j=0;j<n-i-1;j++)
        {
            if(p[j].bt>p[j+1].bt){
                struct process temp=p[j];
                p[j]=p[j+1];
                p[j+1]=temp;

            }
        }
```

```
    }
    calculate(p,n);
    display(p,n);
}
void calculate(struct process p[],int n){
    p[0].wt=0;
    p[0].ct=p[0].bt;
    p[0].tat=p[0].bt;
    for(int i=1;i<n;i++)
    {
        p[i].ct=p[i-1].ct+p[i].bt;
        p[i].tat=p[i].ct-p[i].at;
        p[i].wt=p[i].tat-p[i].bt;
    }
}
void display(struct process p[],int n){
    int tot_wt=0;
    int tot_tat=0;
    printf("pid \t at \t bt \t ct \t tat \t wt ");
    for(int i=0;i<n;i++)
    {
        printf("\n%d \t %d \t %d \t %d \t %d \t
%d",p[i].pid,p[i].at,p[i].bt,p[i].ct,p[i].tat,p[i].wt);
        tot_tat+=p[i].tat;
        tot_wt+=p[i].wt;
    }
    printf("\nAvg TAT:%f ",(tot_tat)/(float)n);
    printf("\nAvg WT : %f",(tot_wt)/(float)n);

}

//prioruty

Int priorityy

Enter bt and priority
at=0
for(int I=0;i<n-1;i++){
for(int j=0;j<n-i-1;j++){
        if(p[j].priority>p[j+1].priority){
Struct process temp=p[j]
p[j]=p[j+1];
p[j+1]=temp;
}
}
}
```

Round Robin

```
#include<stdio.h>
int main(){
int i,n, time, remain, flag=0,t;
int wt=0 ,tat=0, at[10], bt[10] ,rt[10];
printf("Enter no. of processes");
scanf("%d",&n);
remain=n;

for(int i=0;i<n;i++){
printf("Enter arrival time and burst time for process %d".i+1);
scanf("%d",&at[i]);
scanf("%d",&bt[i]);
rt[i]=bt[i]
}


printf("enter the time quantum");
scanf("%d",&t);
printf("\n Process | AT |          BT |          CT    |       TAT    |          |WT);
for(time=0;i=0;remain!=0)
{
if(rt[i]<=t && rt[i]>0){
time+=rt[i];
rt[i]=0;
flag=1
}
else if(rt[i]>0){
rt[i]-=t;
time+=t;
}

if(rt[i]==0 and flag ==1){
remain—;
printf("\n %d \t, % d \t %d \t ,...i+1,at,bt,time-at[i], time-at[i]-bt[i]);
wt+=time-at[i]-bt[i];
tat+=time-at[i];
flag=0;
}

if(i==n-1){
i=0;}
else if(at[i+1]<=time){
        i++;
```

```
}
else
{
i=0;
}
}

printf(“\n Average Waiting Time = %.2f”,(float)wt/n );
printf(“\n Average Turnaround Time =%.2f”,(float)tat/n);
return 0;
}
```

Bankersss

```
#include <stdio.h>
#define MAX_PROCESSES 10
#define MAX_RESOURCES 10

int available[MAX_RESOURCES];
int max[MAX_PROCESSES][MAX_RESOURCES];
int allocation[MAX_PROCESSES][MAX_RESOURCES];
int need[MAX_PROCESSES][MAX_RESOURCES];
int n_process, n_resources;

int safety_algorithm();

int main() {
    int i, j;

    printf("Enter the no. of processes: ");
    scanf("%d", &n_process);
    printf("Enter no. of resources: ");
    scanf("%d", &n_resources);

    printf("Enter the available instance of each resource:\n");
    for (i = 0; i < n_resources; i++) {
        scanf("%d", &available[i]);
    }

    printf("Enter the maximum demand of each process:\n");
    for (i = 0; i < n_process; i++) {
        for (j = 0; j < n_resources; j++) {
            scanf("%d", &max[i][j]);
        }
    }
```

```
    printf("Enter the allocation of resources for each process:\n");
    for (i = 0; i < n_process; i++) {
        for (j = 0; j < n_resources; j++) {
            scanf("%d", &allocation[i][j]);
            need[i][j] = max[i][j] - allocation[i][j];
        }
    }

    if (safety_algorithm()) {
        printf("System is in a safe state.\n");
    } else {
        printf("System is not in a safe state.\n");
    }

    return 0;
}

int safety_algorithm() {
    int work[MAX_RESOURCES];
    int finish[MAX_PROCESSES] = {0};
    int safe_sequence[MAX_PROCESSES];
    int count = 0;

    for (int i = 0; i < n_resources; i++) {
        work[i] = available[i];
    }

    while (count < n_process) {
        int found = 0;

        for (int i = 0; i < n_process; i++) {
            if (finish[i] == 0) {
                int j;
                for (j = 0; j < n_resources; j++) {
                    if (need[i][j] > work[j]) {
                        break;
                    }
                }

                if (j == n_resources) {
                    for (int k = 0; k < n_resources; k++) {
                        work[k] += allocation[i][k];
                    }
                    safe_sequence[count++] = i;
                    finish[i] = 1;
                    found = 1;
```

```
            }
        }
    }

    if (!found) {
        return 0; // System is not in a safe state
    }
  }

  // Printing the safe sequence
  printf("Safe sequence: ");
  for (int i = 0; i < n_process; i++) {
    printf("%d", safe_sequence[i]);
    if (i != n_process - 1) {
        printf(" -> ");
    }
  }
  printf("\n");

  return 1; // System is in a safe state
}
```