DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
RAJIV GANDHI UNIVERSITY OF KNOWLEDGE TECHNOLOGIES,
Basar (Village & Mandal), Nirmal (District), Telangana State - 504107

DESIGN AND ANALYSIS OF ALGORITHMS

LABORATORY MANUAL

B.TECH II YEAR – I SEM (R18) (2019-20)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Operating Systems Lab

Course Objectives:

1. To understand and implement basic services and functionalities of the operating system using system calls.

2. To use modern operating system calls and synchronization libraries in software/ hardware interfaces.

3. To understand the benefits of thread over process and implement synchronized programs using multithreading concepts.

4. To Analyze and simulate CPU Scheduling Algorithms like FCFS, Round Robin, SJF, and Priority.

5. To implement memory management schemes and page replacement schemes.

6. To simulate file allocation and organization techniques.

7. To understand the concepts of deadlock in operating systems and implement them in multiprogramming system

INDEX

	SCAN c) C-SCAN	
Week 9	10.Write a C program to simulate disk scheduling algorithms a) look b) b) C-look	
Week 10	11.Write a C program to simulate page replacement algorithms a)Reference Bit b)MFU	
Week 11	12.Write a C program to simulate Resource request allocation algorithm for the purpose of deadlock avoidance	
Week 12	13.Write a C program to simulate strict alternation problem using Threads?	
Week 13	14.Write a C program to simulate peterson's solution.	

Course Outcomes:

- Upon the completion of Operating Systems practical course, the student will be able to gain practical experience with designing and implementing concepts of operating systems such as system calls, CPU scheduling, process management, memory management, file systems and deadlock handling, using C language in Linux environment.

1.Write a C program to simulate the following non-preemptive CPU Sheduling algorithms to find turnaround time,waiting time, average turnaround time and average waiting time.

a) FCFS b) SJF

a)FCFS

Program:

```
#include<stdio.h>
int main()
{
 int AT[10],BT[10],CT[10],TAT[10],WT[10],I[10],GC[10],P[10];
 int n,temp,temp1,I,j;
 float avg_WT,avg_TAT;
 printf(“enter the no of process”);
 scanf(“%d”,&n);
 printf(“enter the AT”);
 for(int i=0;i<n;i++)
 {

 scanf(“%d”,&AT[i]);
  P[i]=i+1;
 }
 printf(“enter the BT”);
 for(int i=0;i<n;i++)
 {
  scanf(“%d”,&BT[i]);
 }
   int k=0;
 for(i=0;i<n;i++)
    {
 for(j=0;j<n-1;j++)
 {
```

```
 if(AT[j]>AT[j+1])
 {
  temp=AT[j];
  AT[j]=AT[j+1];
  AT[j+1]=temp;
  temp1=BT[j];
  BT[j]=BT[j+1];
  BT[j+1]=temp1;
  int temp2=P[j];
  P[j]=P[j+1];
  P[j+1]=temp2;
 }
} }
CT[0]=AT[0]+BT[0];
 I[0]=0;
 for(i=1;i<n;i++)
 {
  I[i]=AT[i]-CT[i-1];
  if(I[i]>0)
  CT[i]=CT[i-1]+BT[i]+I[i];
  else
  CT[i]=CT[i-1]+BT[i];
 }
for(i=0;i<n;i++)
{
 TAT[i]=CT[i]-AT[i];
 WT[i]=TAT[i]-BT[i];
}
for(i=0;i<n;i++)
{
```

```
 avg_TAT+=TAT[i];
 avg_WT+=WT[i];
}
printf("PNO|\t AT|\t BT|\t CT|\t TAT|\t WT\n");
for(i=0;i<n;i++)
{
 printf("%d|\t %d|\t %d|\t %d| \t %d|\t %d\n",i+1,AT[i],BT[i],CT[i],TAT[i],WT[i]);
}
printf("avg TAT is:  %f",(avg_TAT/n));
printf("avg WT is: %f",(avg_WT/n));
printf("\n");
printf("GANTT CHART\n ");
for(i=0;i<n;i++)
printf("p%d\t",P[i]);
 return 0;
}
```

b)SJF

Program:

```
#include<stdio.h>
void main()
{
 int at[10],bt[10],ct[10],tat[10],wt[10],p[10],I,j,k,temp1,temp2,temp3,n,id[10],f;
 int at1[10],bt1[10];
 float atat=0,awt=0;
 printf("enter the no.of processes:");
 scanf("%d",&n);
 printf("enter the arrival time:");
 for(i=0;i<n;i++)
 {
  scanf("%d",&at[i]);
```

```
 p[i]=i+1;
 at1[i]=0;
}
printf("enter the burst times:");
for(i=0;i<n;i++)
{
 scanf("%d",&bt[i]);
}
for(i=0;i<n;i++)
{
 for(j=0;j<n-1;j++)
 {
  if(at[j]>at[j+1])
  {
   temp1=at[j];
   at[j]=at[j+1];
   at[j+1]=temp1;
   temp2=bt[j];
   bt[j]=bt[j+1];
   bt[j+1]=temp2;
   temp3=p[j];
   p[j]=p[j+1];
   p[j+1]=temp3;
  }
 }
}
for(i=0;i<n;i++)
{
 for(j=1;j<n;j++)
 {
```

```
  if(at[i]==at[j])
  {
   if(bt[i]>bt[j])
   {
    temp1=bt[i];
    bt[i]=bt[j];
    bt[j]=temp1;
    temp1=p[i];
    p[i]=p[j];
    p[j]=temp1;
   }
  }   } }
for(i=1;i<n;i++)
{
 for(j=1;j<n-1;j++)
 {
  if(bt[j]>bt[j+1])
  {
   temp1=at[j];
   at[j]=at[j+1];
   at[j+1]=temp1;
   temp1=bt[j];
   bt[j]=bt[j+1];
   bt[j+1]=temp1;
   temp1=p[j];
   p[j]=p[j+1];
   p[j+1]=temp1;
  }
 }  }
k=0;
```

```
 id[0]=0;
 ct[0]=at[0]+bt[0];
 for(i=1;i<n;i++)
 {
  id[i]=at[i]-ct[i-1];
  if(id[i]>0)
  ct[i]=bt[i]+id[i]+ct[i-1];
  else
  ct[i]=bt[i]+ct[i-1];
 }
 for(i=0;i<n;i++)
 {
  tat[i]=ct[i]-at[i];
  wt[i]=tat[i]-bt[i];
  atat+=tat[i];
  awt+=wt[i];
 }
 printf("p\tat\tbt\tct\ttat\twt\t\n");
 for(i=0;i<n;i++)
 {
  printf("p%d\t%d\t%d\t%d\t%d\t%d\t\n",p[i],at[i],bt[i],ct[i],tat[i],wt[i]);
 }
 printf("avg_tat:%f \t\t avg_wt:%f",(atat/n),(awt/n));
}
```

WEEK-2

2.Write a C program to simulate the following non-preemptive CPU Sheduling algorithms to find turnaround time and waiting time, average turnaround time and average waiting time.

c) Round Robin (pre-emptive) d) Priority

c)Round Robin

```
#include<stdio.h>
void sort(int *a,int *b,int *p ,int n);
main()
{
 int n,I,sumb=0,k,j,m;
 printf(“enter no of procesers”);
 scanf(“%d”,&n);
 int p[n],st[n], a[n], b[n],tq,queue[100];
 int c[n],tat[n],wt[n];
 for(i=0;i<n;i++)
 {
  printf(“\nenter processor”);
  scanf(“%d”,&p[i]);
  printf(“\nenter eferen time:”);
  scanf(“%d”,&a[i]);
  printf(“\nenter burst time:”);
  scanf(“%d”,&b[i]);
 }
 printf(“\nenter quantam time:”);
 scanf(“%d”,&tq);
    for(i=0;i<n;i++)
    {
     sumb=sumb+b[i];
 }
 sort(a,b,p,n);
 for(i=0;i<n;i++)
 printf(“%d\t”,p[i]);
 k=0;
    if(b[k]<tq)
    {
```

```
        sumb=sumb-b[k];
         c[k]=a[k]+b[k];
         b[k]=0;
         queue[0]=p[k];
}
else
{
b[k]=b[k]-tq;
   queue[0]=p[k];
   sumb=sumb-tq;
   c[k]=a[k]+tq;
}
if(a[k]!=0)
    printf("0");
   printf("%d",a[k]);
   printf("..>%d<..",p[k]);
    printf("(%d)",c[k]);
    i=0;m=0;
   int h,pre=a[k],in=0;

while(sumb!=0)
{
 for(j=in+1;j<n;j++)
  {
          if(b[i]>0)
{
         m++;
       queue[m]=p[j];

     }
```

```
}
  if(b[i]!=0)
  {
 m++;
   queue[m]=p[i];

}
     in++;
     int ind;
     for(j=0;j<n;j++)
     {
      if(queue[in]==p[j])
      {
       ind=j;
       break;
 }

}
if(b[ind]>=tq)
{
 b[ind]=b[ind]-tq;
c[ind]=c[k]+tq;
sumb=sumb-tq;
 pre=c[ind]-tq;
}
else
{
 c[ind]=c[k]+b[ind];
  sumb=sumb-b[ind];
 pre=c[ind]-b[ind];
```

```
   b[ind]=0;
      }
  printf("..>%d<..",p[ind]);
  printf("(%d)",c[ind]);
 k=ind;
 i=ind;
 if(sumb==0)
 {
  break;
 }
}
int sum=0,sum1=0;
for(i=0;i<n;i++)
{
 tat[i]=c[i]-a[i];
 wt[i]=tat[i]-b[i];
 sum=sum+tat[i];
 sum1=sum1+wt[i];
}
printf("\n\n");
printf("p  a  b  c  tat  wt\n");
for(i=0;i<n;i++)
{
 printf("%d  %d  %d  %d  %d  %d ",p[i],a[i],b[i],c[i],tat[i],wt[i]);
 printf("\n");
}
printf("average tat:%d",sum/n);
printf("average wt:%d",sum1/n);
}
void sort(int *a,int *b,int *p,int n)
```

```
{
 int I,j,t;
 for(i=0;i<n;i++)
 {
  for(j=I;j<n;j++)
  {
   if(a[i]>=a[j])
   {
    t=a[i];
    a[i]=a[j];
    a[j]=t;
    t=p[i];
       p[i]=p[j];
       p[j]=t;
       t=b[i];
       b[i]=b[j];
       b[j]=t;
   }
   else if(a[i]==a[j])
   {
    if(p[i]>p[j])
    {
     t=b[i];
        b[i]=b[j];
        b[j]=t;
     t=p[i];
           p[i]=p[j];
           p[j]=t;
    }
```

```
  }
 }
}
}
```

d)Priority

Program:

```
#include<stdio.h>

int main()
{
 int AT[10],BT[10],CT[10],TAT[10],WT[10],I[10],P[10],ATT[10],PR[10];
 int n,temp,temp1,I,j;
 float avg_WT,avg_TAT;
 printf("enter the no of process");
 scanf("%d",&n);
 printf("enter the AT");
 for( i=0;i<n;i++)
 {  scanf("%d",&AT[i]);
 ATT[i]=AT[i];
  P[i]=i+1;
 }
  printf("enter the BT");
 for( i=0;i<n;i++)
 {
   scanf("%d",&BT[i]);
 }
  printf("enter the priority");
 for( i=0;i<n;i++)
 {
```

```
 scanf("%d",&PR[i]);
 }
 for(i=0;i<n;i++)
 {
  for(j=0;j<n-1;j++)
  {
    if(PR[j]>PR[j+1] && PR>0)
    {
     temp=AT[j];
     AT[j]=AT[j+1];
     AT[j+1]=temp;
     temp1=BT[j];
     BT[j]=BT[j+1];
     BT[j+1]=temp1;
     int temp2=P[j];
     P[j]=P[j+1];
     P[j+1]=temp2;
     temp=PR[j];
     PR[j]=PR[j+1];
     PR[j+1]=temp;
   }
  }
 }
 int min=576,k=0;
for(i=0;i<n;i++)
{
 if(AT[i]<min)
 {
  min=AT[i];
  k=I;
```

```
 }
}
 CT[0]=BT[k]+AT[k];
temp=P[k];
P[k]=P[0];
P[0]=temp;
temp=BT[k];
BT[k]=BT[0];
BT[0]=temp;
temp=PR[k];
PR[k]=PR[0];
PR[0]=temp;
 for(i=1;i<n;i++)
 {
  for(j=1;j<n-1;j++)
  {
    if(PR[j]>PR[j+1])
    {
      temp=AT[j];
     AT[j]=AT[j+1];
     AT[j+1]=temp;
     temp1=BT[j];
     BT[j]=BT[j+1];
     BT[j+1]=temp1;
     int temp2=P[j];
     P[j]=P[j+1];
     P[j+1]=temp2;
     temp=PR[j];
     PR[j]=PR[j+1];
```

```
    PR[j+1]=temp;
     }
    }
     }
   for(i=1;i<n;i++)
 {
  for(j=1;j<n;j++)
  {
   if(PR[i]==PR[j])
   {
    if(AT[i]<AT[j])
    {
     temp=AT[i];
     AT[i]=AT[j];
     AT[j]=temp;
     temp=BT[i];
     BT[i]=BT[j];
     BT[j]=temp;
     temp=P[i];
     P[i]=P[j];
     P[j]=temp;
     temp=PR[i];
     PR[i]=PR[j];
     PR[j]=temp;
       }
   } } }
 I[0]=0;
for(i=1;i<n;i++)
{
 I[i]=AT[i]-CT[i-1];
```

```
 if(I[i]>0)
 CT[i]=CT[i-1]+BT[i]+I[i];
 else
 CT[i]=CT[i-1]+BT[i];
}
for(i=0;i<n;i++)
{
 TAT[i]=CT[i]-AT[i];
 WT[i]=TAT[i]-BT[i];
}
 for(i=0;i<n;i++)
{
 avg_TAT+=TAT[i];
 avg_WT+=WT[i];
}
 for(i=0;i<n;i++)
{
 printf("%d\t",CT[i]);
}
printf("GANTT CHART\n ");
for(i=0;i<n;i++)
printf("p%d\t",P[i]);
printf("\n");
  printf("avg TAT is:  %f",(avg_TAT/n));
printf("avg WT is: %f",(avg_WT/n)); }
```

WEEK-3

Write a C program to simulate the following contiguous memory allocation techniques a) Worst-fit b) Best-fit c) First-fit

a)Worst –fit

```
#include<stdio.h>
main()
{
 int p[20],b[20],check[20],first[20],max,internal[20];
 int I,j,k,l,n,n1,f=0;
 printf(“enter the no of process:”);
 scanf(“%d”,&n);
 printf(“enter thesizes of each process:”);
 for(i=0;i<n;i++)
 {
  scanf(“%d”,&p[i]);
}
j=I;
 printf(“enter the no. of block:”);
 scanf(“%d”,&n1);

 printf(“enter the size of each block:”);
 for(i=0;i<n1;i++)
 {
  scanf(“%d”,&b[i]);
  check[i]=0;
  first[i]=0;
     internal[i]=0;
  j++;
}
k=0;
 for(i=0;i<n;i++)
{
 I=1;
 f=0;
```

```
 max=-999;
 for(j=0;j<n1;j++)
 {
  if(b[j]>max && b[j]>=p[i] && check[j]!=1)
  {
   max=b[j];
   k=j;
   f=1;
  }
 }
   if(f==1)
 {
 first[k]=i+1;
 check[k]=1;
 internal[k]=b[k]-p[i];
 k++;
 l++;
}
}
 printf("\n");
 printf("b_s\tp_s\torder\tinternal_frag\n");
 for(i=0;i<n1;i++)
 printf("%d\t%d\t%d%\t%d\n",b[i],p[i],first[i],internal[i]);
  }
```

b)Best-Fit

Program:

```
 #include<stdio.h>
 main()
 {
  int p[20],b[20],check[20],first[20],max,m,internal[20];
```

```
 int l,j,k,l,n,n1;
 printf("enter the no of process:");
 scanf("%d",&n);
 printf("enter the sizes of each process:");
 for(i=0;i<n;i++)
 {
  scanf("%d",&p[i]);
}
 printf("enter the no. of block:");
 scanf("%d",&n1);

 printf("enter the size of each block:");
 for(i=0;i<n1;i++)
 {
  scanf("%d",&b[i]);
  check[i]=0;
  first[i]=0;
  internal[i]=0;
}
k=0;
int f=0;
for(i=0;i<n;i++)
{
 max=-999;
 //min=76556;
 for(j=0;j<n1;j++)
 {
  f=0;
  if(b[j]>max && b[j]>=p[i] && check[j]!=1)
  {
```

```
  max=b[j];
  k=j;
  f=1;
 }
}
 for(m=0;m<n1;m++)
 {
    if(max>=b[m] && b[m]>=p[i] && check[m]!=1)
  {
   f=1;
   max=b[m];
   k=m;
  }
 }
if(f==1)
{
first[k]=i+1;
check[k]=1;
internal[k]=b[k]-p[i];
k++;
}
}
printf("\n");
printf("order\tinternal_frag");
printf("\n");
for(i=0;i<n1;i++)
{
  printf("%d\t%d",first[i],internal[i]);
  printf("\n");
} }
```

c)First-Fit

```
#include<stdio.h>
main()
{
 int p[20],b[20],check[20],first[20],max,internal[20];
 int I,j,k,l,n,n1;
 printf("enter the no of process:");
 scanf("%d",&n);
 printf("enter the sizes of each process:");
 for(i=0;i<n;i++)
 {
  scanf("%d",&p[i]);
}
 printf("enter the no. of block:");
 scanf("%d",&n1);

 printf("enter the size of each block:");
 for(i=0;i<n1;i++)
 {
  scanf("%d",&b[i]);
  check[i]=0;
  first[i]=0;
  internal[i]=0;
}
k=0;
int f=0;
 for(i=0;i<n;i++)
{
 f=0;
  for(j=0;j<n1;j++)
```

```
{
 if(b[j]>=p[i] && check[j]!=1)
 {
  k=j;
  f=1;
  break;
 }
}
if(f==1)
{
  first[k]=i+1;
check[k]=1;
internal[k]=b[k]-p[i];
printf("%d\t",i+1);   }
else
{
 printf("waiting:p%d",i+1);
}
 }
printf("\n");
printf("order\tinternal_fragmentation\n");
for(i=0;i<n1;i++)
{
 if(first[i]>0)
 printf("p%d\t%d\n",first[i],internal[i]);
 else
 printf("-\n");
}  }
```

WEEK-4

Write a C program to simulate page replacement algorithms a) FIFO b) LRU c) Optimal

a)FIFO

```
#include<stdio.h>
main()
{
int n,r,l,j,k,c=0,l=0,m;
printf("enter the no of frames:");
scanf("%d",&n);
int frame[n],frame1[n];
for(i=0;i<n;i++)
{
 frame[i]=-1;  }
printf("enter the length of eference string :");
scanf("%d",&r);
int ref[r];
printf("enter the order of reference string:");
for(i=0;i<r;i++)
{
 scanf("%d",&ref[i]); }
  for(i=0;i<r;i++)
{
 for(j=0;j<n;j++)
 {
  if(frame[j]==ref[i])
  {
   break;
  } }
 if(j==n)
  {
```

```
   frame[l]=ref[i];
   l++;
   c++;
  }
    for(m=0;m<n;m++)
    printf("\t%d",frame[m]);
     if(j==n)
    printf("\tPF No. %d",c);
    printf("\n");
    if(l==n)
    l=0;
}
printf("total no of page fault is:%d",c);
printf("\ntotal no of page hit is:%d",r-c);
}
```

b)LRU

```
#include<stdio.h>
main()
{
 int r,f,j,k,m,ind,l=0,min,s,pf=0;
 printf("enter the no of frame:\n");
 scanf("%d",&f);
 int n=f;
 printf("enter the no of reference string:\n");
 scanf("%d",&r);
 int frame[f],ref[r],l,index[f],check[f];
 printf("enter the refence string:\n");
 for(i=0;i<r;i++)
 {
  scanf("%d",&ref[i]);
```

```
}
for(i=0;i<f;i++)
{
 frame[i]=-1;
 check[i]=0;
 index[i]=-1;
}
for(i=0;i<r;i++)
{
 f=0;
 for(k=0;k<n;k++)
 {
 if(frame[k]==ref[i])
 {
   f=1;
   j=k;
   break; } }
 if(f==0)
 {
  pf++;
  if(frame[l]==-1)
  {
   frame[l]=ref[i];
   index[l]=l;
   l++;
    }
  else
  {
   min=999;
   for(m=0;m<n;m++)
```

```
   {
    if( index[m]<min)
    {
     min=index[m];
     ind=m;
    }
     }
   frame[ind]=ref[i];
index[ind]=I;   } }
 else
 {
    index[k]=I;
 }
   for(s=0;s<n;s++)
 {  printf("%d\t",frame[s]);  }
 printf("\n");
}
 printf("page faults:%d\t",pf);
}
```

c)Optimal

```
#include<stdio.h>
 main()
{
 int n,r,I,j,k,c=0,I=0,m,f=0,max,ind,found=0,pf=0,find,num=200;
 printf("enter the no of frames:");
 scanf("%d",&n);
 int frame[n],frame1[n],index[n],in=0;
 for(i=0;i<n;i++)
 {
  frame[i]=-1;
```

```
 }
printf(“enter the length of eference string :”);
scanf(“%d”,&r);
int ref[r];
printf(“enter the order of reference string:”);
for(i=0;i<r;i++)
{
 scanf(“%d”,&ref[i]);
}
l=0;
for(i=0;i<r;i++)
{
 f=0;
 found=0;
 for(j=0;j<n;j++)
 {
  if(frame[j]==ref[i])
  {
   f=1;
   find=j;
   break;
  }  }
   if(f==1)
 {
  for(k=i+1;k<r;k++)
  {
   if(frame[find]==ref[k])
   {
    found=1;
    ind=k;
```

```
   break;
  }
 }
 if(found==1)
 {
  index[find]=ind;
 }
 else
 {
index[find]=num++;
 }
}
if(f==0)
{
 if(frame[i]==-1)
 {
  frame[l]=ref[i];
  for(k=i+1;k<r;k++)
  {
   if(frame[l]==ref[k])
   {
    ind=k;
    found=1;
    break;
   }
  }
  if(found==1)
  {
  index[l]=ind;
  l++;
```

```
}
 else
 {
  index[l]=num++;
  l++;
 }  }
else
{
// printf("ji\n");
 max=-200;
 for(m=0;m<n;m++)
 {
  if(index[m]>max)
  {
   max=index[m];
   ind=m;
  }
 }
// printf("max:%d\n",max);
 frame[ind]=ref[i];
 for(k=i+1;k<r;k++)
 {
  if(frame[ind]==ref[k])
  {
   found=1;
   in=k;
   break;
  }
 }
 if(found==1)
```

```
  index[ind]=in;
  else
  {
  index[ind]=num++;
   }
}
 pf++;
}
if(f==0)
{
 for(m=0;m<n;m++)
 {
 printf("%d\t",frame[m]);
 }
 printf("page fault:%d\t",pf);
}
 else
{
 for(m=0;m<n;m++)
 {
  printf("%d\t",frame[m]);
 }
 printf("\n");
} printf("\n");
 printf("\n");
 }
printf("page faults:%d ",pf);
}
```

WEEK-5

5.Write a C program to simulate Bankers algorithm for the purpose of deadlock avoidance.

Program:

```
#include<stdio.h>
#include<dos.h>
void main()
{
 int n,m;
 printf("enter the number of process:");
 scanf("%d",&n);
 printf("no of resources:\n");
 scanf("%d",&m);
 int i,j,k,finish[n],allocation[10][10],max[10][10],need[10][10],work[10][10];
 for(i=0;i<n;i++)
 finish[i]=0;
 printf("enter the allocation matrix\n");
 for(i=0;i<n;i++)
 {
  for(j=0;j<m;j++)
  {
   scanf("%d",&allocation[i][j]);
  }
 }
 printf("enter the max matrix\n");
 for(i=0;i<n;i++)
 {
  for(j=0;j<m;j++)
  {
   scanf("%d",&max[i][j]);
  }
```

```
}  printf("enter the available resources:\n");
for(i=0;i<1;i++)
{
 for(j=0;j<m;j++)
 {
   scanf("%d",&work[i][j]);
 }
  }for(i=0;i<n;i++)
{
for(j=0;j<m;j++)
 {
  need[i][j]=max[i][j]-allocation[i][j];
 }
}
for(i=0;i<n;i++)
{
 for(j=0;j<m;j++)
 {
  printf("%d\t",need[i][j]);
 }
 printf("\n");
}
int count=0;
int c=0;
printf("The sequence is: ");
for(i=0;;i++)
{
 count=0;
 if(finish[i]==0)
 {
```

```
   for(k=0;k<m;k++)
  {
   if(need[i][k]<=work[0][k])
    {
     count++;
    }
   }
 if(count==m)
 {
  for(k=0;k<m;k++)
  {
   work[0][k]=work[0][k]+allocation[i][k];
   finish[i]=1;
  }
  c++;

  printf("%d\t",i);
 } }
 if(c==n)
 break;
 if(i==n-1)
 {
  i=0; }
}
}
```

WEEK-6

6. Write a C program to simulate readers –writers solution using semaphore?

Program:

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

```
        #include<semaphore.h>
 sem_t mutex,writeblock;
 int data = 0,rcount = 0;
 void *reader(void *arg)
{
int f;   f = ((int)arg);
 sem_wait(&mutex);
  rcount = rcount + 1;
 if(rcount==1)
 sem_wait(&writeblock);
 sem_post(&mutex);
printf("Data read by the reader%d is %d\n",f,data);
 sleep(1);
 sem_wait(&mutex);
rcount = rcount - 1;
 if(rcount==0)
sem_post(&writeblock);
sem_post(&mutex); }
void *writer(void *arg)
 {
int f;   f = ((int) arg);
 sem_wait(&writeblock);
 data++;
printf("Data writen by the writer%d is %d\n",f,data);
sleep(1);
 sem_post(&writeblock);
}
 int main()
{
 int i,b;
```

```
 pthread_t rtid[5],wtid[5];
  sem_init(&mutex,0,1);
 sem_init(&writeblock,0,1);
  for(i=0;i<=2;i++)
  {
     pthread_create(&wtid[i],NULL,writer,(void *)i);
 pthread_create(&rtid[i],NULL,reader,(void *)i);
  }
 for(i=0;i<=2;i++)
 {
  pthread_join(wtid[i],NULL);
 pthread_join(rtid[i],NULL);
  }
 return 0;
}
```

7.Write a C program to simulate the dining-philosophers problem that leads to the deadlock situations?

Program:

```
#include<stdio.h>
#include<pthread.h>
 pthread_t philosopher[5];
 pthread_mutex_t chopstick[5];
 void *left(int n)
 {
printf("left\n");
  printf("\n philosopher %d is thinking !!\n",n);
  pthread_mutex_lock(&chopstick[n-1]);
   pthread_mutex_lock(&chopstick[(n)%5]);
  printf("\n philosopher %d is eating using chopstick[%d] and chopstick[%d]!!\n",n,(n-1),(n%5));
 sleep(2);
   pthread_mutex_unlock(&chopstick[n-1]);
```

```
   pthread_mutex_unlock(&chopstick[(n)%5]);
   printf("\n philosopher %d finished eating !!\n",n);
   }
void *right(int n)
{
printf("\nright\n");
printf("\n philosopher %d is thinking !!\n",n);
    pthread_mutex_lock(&chopstick[(n)%5]);
   pthread_mutex_lock(&chopstick[n-1]);
  printf("\n philosopher %d is eating using chopstick[%d] and chopstick[%d]!!",n,(n-1),(n%5));
 sleep(2);
  pthread_mutex_unlock(&chopstick[(n)%5]);
  pthread_mutex_unlock(&chopstick[n-1]);
 printf("\n philosopher %d finished eating !!\n",n);      }

int main()
{
  int i;
    for(i=0;i<5;i++)
  {
  pthread_mutex_init(&chopstick[i],NULL);
  }
        for(i=1;i<5;i++)
  {
  pthread_create(&philosopher[i],NULL,(void *)left,(int *)i );
        }
    pthread_create(&philosopher[i],NULL,(void *)right,(int *)i );
     for(i=1;i<=5;i++)
 {
  pthread_join(philosopher[i],NULL);
```

```
  }
return 0;
 }
```

WEEK-7

8.Write a C program to simulate Producer-Consumers problem ?

```
#include<stdio.h>
#include<dos.h>
int mutex=1,full=0,empty,dummy;
void producer();
void consumer();
void wait(int *);
void signal(int *);
void producer()
{
wait(&empty);
wait(&mutex);
printf("item produced\n");
signal(&mutex);
signal(&full);
}
void consumer()
{
 wait(&full);
wait(&mutex);
 printf("item consumed\n");
 signal(&mutex);
signal(&empty);
}
void wait(int *x)
```

```
{
while(*x<=0);
 *x=*x-1;
}
 void signal(int *y)
 {
  *y=*y-1;
}
 void main()
{
  printf("enter the buffer size\n");
 scanf("%d",&empty);
 dummy=empty;
 int op;
do
 {
 printf("1.producer\t2.consumer\n");
   printf("enter the option:\n");
 scanf("%d",&op);
switch(op)
 {
  case 1:
 producer();
  break;
   case 2:
   consumer();
  break;
default:
   printf("no option\n");
    break;
```

```
}
 }
while(op!=3);
 }
```

WEEK-8

9.Write a C program to simulate disk scheduling algorithms a) FCFS b) SCAN c) C-SCAN

a)FCFS

Program:

```
 #include<stdio.h>
//98 183 37 122 14 124 65 67
void main()
{
 int n,arr[25],i,j,k,m=0,l,h,f,temp,flag[20],fl=0;
 printf("enter the n");
 scanf("%d",&n);
 printf("enter the limit:");
 scanf("%d",&l);
 printf("enter the requests:");
 for(i=0;i<n;i++)
 {
  scanf("%d",&arr[i]);
  flag[i]=0;
 }
 printf("enter the head:");
 scanf("%d",&h);
 i=h;
 for(j=0;j<n;j++)
 {
  if(arr[j]>i)
```

```
 {
  while(i<arr[j])
  i++;
  m=m+(i-h);
  h=i;
  printf("%d\t",h);
 }
 else if(arr[j]<i)
 {
  while(i>arr[j])
  i--;
  m=m+(h-i);
  h=i;
  printf("%d\t",h);
 }
 printf("%d\t",m);
}
printf("%d\t",m);
}
```

b)SCAN

Program:

```
#include<stdio.h>
void main()
{
 int n,arr[25],i,j,k,m=0,l,h,f,temp,flag[20];
 printf("enter the n");
 scanf("%d",&n);
 printf("enter the limit:");
 scanf("%d",&l);
 printf("enter the requests:");
```

```
for(i=0;i<n;i++)
{
 scanf("%d",&arr[i]);
 flag[i]=0;
}
for(i=0;i<n;i++)
{
 for(j=0;j<n-1;j++)
 {
  if(arr[j]>arr[j+1])
  {
   temp=arr[j];
   arr[j]=arr[j+1];
   arr[j+1]=temp;
  } } }
printf("enter the head:");
scanf("%d",&h);
printf("enter the direction 0 for left 1 for right:");
scanf("%d",&f);
if(f==1)
{
 i=h;
 j=0;
 k=h;
 while(i<l)
 {
  for(j=0;j<n;j++)
  {
   if(i==arr[j]&&flag[j]==0)
   {
```

```
   m=m+(i-h);
   flag[j]=1;
   h=arr[j];
   }
  }
  i++;
 }
 if(i==l)
 {
  i=l;j=0;
  m=m+(i-h);
  h=l;
  while(i>0)
  {
   for(j=0;j<n;j++)
   {
    if(i==arr[j]&&flag[j]==0)
   {
    m=m+(h-i);
    h=arr[j];
    flag[j]=1;
   }
   }
   i--;
  }
  m=m+(h-i);
 }
}
else if(f==0)
{
```

```
i=h;
k=h;
while(i>0)
{
 for(j=0;j<n;j++)
 {
  if(i==arr[j]&&flag[j]==0)
  {
   m=m+(h-i);
   flag[j]=1;
   h=arr[j];
  }
 }
 i--;
}
m=m+(h-i);
if(i==0)
{
 i=0;
 h=0;
 while(i<l)
 {
  for(j=0;j<n;j++)
  {
   if(i==arr[j]&&flag[j]==0)
   {
   m=m+(i-h);
   flag[j]=1;
   h=arr[j];
   }
```

```
   }
    i++;
  }
   m=m+(i-h);
 }
}
 printf("%d",m);
}
```

c)C-SCAN

Program:

```
#include<stdio.h>
//98 183 37 122 14 124 65 67
void main()
{
 int n,arr[25],i,j,k,m=0,l,h,f,temp,flag[20];
 printf("enter the n");
 scanf("%d",&n);
 printf("enter the limit:");
 scanf("%d",&l);
 printf("enter the requests:");
 for(i=0;i<n;i++)
 {
  scanf("%d",&arr[i]);
  flag[i]=0;
 }
 for(i=0;i<n;i++)
 {
  for(j=0;j<n-1;j++)
  {
   if(arr[j]>arr[j+1])
```

```
  {
   temp=arr[j];
   arr[j]=arr[j+1];
   arr[j+1]=temp;
  }
 }
}
printf("enter the head:");
scanf("%d",&h);
printf("enter the direction 0 for left 1 for right:");
scanf("%d",&f);
if(f==1)
{
 i=h;
 j=0;
 k=h;
 while(i<l)
 {
  for(j=0;j<n;j++)
  {
   if(i==arr[j]&&flag[j]==0)
   {
   m=m+(i-h);
   flag[j]=1;
   h=arr[j];
   }
  }
  i++;
 }
 if(i==l)
```

```
 {
  m=m+(i-h);
  i=0;j=0;
  h=0;
  while(i<k)
  {
   for(j=0;j<n;j++)
   {
    if(i==arr[j]&&flag[j]==0)
   {
    m=m+(i-h);
    h=arr[j];
    flag[j]=1;
   }
   }
   i++;
  }
 }
}
else if(f==0)
{
 i=h;
 k=h;
 while(i>0)
 {
  for(j=0;j<n;j++)
  {
   if(i==arr[j]&&flag[j]==0)
   {
    m=m+(h-i);
```

```
    flag[j]=1;
    h=arr[j];
   }
  }
  i--;
 }
 if(i==0)
 {
  m=m+(h-i);
  i=0;
  h=0;
  while(i<l)
  {
   for(j=0;j<n;j++)
   {
    if(i==arr[j]&&flag[j]==0)
    {
    m=m+(i-h);
    flag[j]=1;
    h=arr[j];
    }
   }
   i++;
  }
  m=m+(i-h);
 }
}
 printf("%d",m);
}
```

10.Write a C program to simulate disk scheduling algorithms

a) look b) b) C-look

a)Look:

```
#include<stdio.h>
void main()
{
 int arr[25],n,i,j,k,l,h,m=0,max=0,min=999,flag[20],temp,f;
 printf("enter n");
 scanf("%d",&n);
 printf("enter the limit:");
 scanf("%d",&l);
 printf("enter the requests:");
 for(i=0;i<n;i++)
 {
  scanf("%d",&arr[i]);
  flag[i]=0;
 }
 for(i=0;i<n;i++)
 {
  for(j=0;j<n-1;j++)
  {
   if(arr[j]>arr[j+1])
   {
    temp=arr[j];
    arr[j]=arr[j+1];
    arr[j+1]=temp;
   } } }
 min=arr[0];max=arr[n-1];
 printf("enter the head:");
```

```
scanf("%d",&h);
printf("enter the direction 0 for left 1 for right:");
scanf("%d",&f);
if(f==1)
{
 i=h;
 k=h;
 while(i<=max)
 {
  for(j=0;j<n;j++)
  {
   if(i==arr[j]&&flag[j]==0)
   {
    m=m+(i-h);
    flag[j]=0;
    h=arr[j];    }   }
  i++;
 }
 h=max;
 while(i>=min)
 {
  for(j=0;j<n;j++)
  {
   if(i==arr[j]&&flag[j]==0)
   {
    m=m+(h-i);
    flag[j]=1;
    h=arr[j];
   }   }
  i--;
```

```
 } }
if(f==0)
{
 i=h;
 k=h;
 while(i>=min)
 {
  for(j=0;j<n;j++)
  {
   if(i==arr[j]&&flag[j]==0)
   {
    m=m+(h-i);
    flag[j]=1;
    h=arr[j];
   } }
  i--;
 }
 h=min;
 i=max;
 m=m+(i-h);
 h=max;
 while((i-1)>k)
 {
  for(j=0;j<n;j++)
  {
   if(i==arr[j]&&flag[j]==0)
   {
    m=m+(h-i);
    flag[j]=1;
    h=arr[j];
```

```
   }  }
  i--;
 } }
 printf("%d",m);
}
```

b) C-look

```
#include<stdio.h>
void main()
{
 int arr[25],n,i,j,k,l,h,m=0,max=0,min=999,flag[20],temp,f;
 printf("enter n");
 scanf("%d",&n);
 printf("enter the limit:");
 scanf("%d",&l);
 printf("enter the requests:");
 for(i=0;i<n;i++)
 {
  scanf("%d",&arr[i]);
  flag[i]=0;
 }
 for(i=0;i<n;i++)
 {
  for(j=0;j<n-1;j++)
  {
   if(arr[j]>arr[j+1])
   {
    temp=arr[j];
    arr[j]=arr[j+1];
    arr[j+1]=temp;
   } } }
```

```
min=arr[0];max=arr[n-1];
printf("enter the head:");
scanf("%d",&h);
printf("enter the direction 0 for left 1 for right:");
scanf("%d",&f);
if(f==1)
{
 i=h;
 k=h;
 while(i<=max)
 {
  for(j=0;j<n;j++)
  {
   if(i==arr[j]&&flag[j]==0)
   {
    m=m+(i-h);
    flag[j]=0;
    h=arr[j];
   }
  }
  i++;
}
 i=0;
 h=0;
 while(i<k)
 {
  for(j=0;j<n;j++)
  {
   if(i==arr[j]&&flag[j]==0)
   {
```

```
    m=m+(i-h);
    flag[j]=1;
    h=arr[j];
   }   }
  i++;
 }  }
if(f==0)
{
 i=h;
 k=h;
 while(i>=min)
 {
  for(j=0;j<n;j++)
  {
   if(i==arr[j]&&flag[j]==0)
   {
    m=m+(h-i);
    flag[j]=1;
    h=arr[j];
   }
  }
  i--;
 }
 i=max;
 h=max;
 while(i>k)
 {
  for(j=0;j<n;j++)
  {
   if(i==arr[j]&&flag[j]==0)
```

```
  {
   m=m+(h-i);
   flag[j]=1;
   h=arr[j];
  }
 }
 i--;
}
}
printf("%d",m); }
```

ASSIGNMENT

WEEK -1

Write a C program to simulate the following preemptive CPU Scheduling algorithms to find turnaround time, waiting time, average turnaround time and average waiting time. a) SJF b) SRTF

a)SJF

Program:

```
#include<stdio.h>
void main()
{
 int at[10],bt[10],ct[10],tat[10],wt[10],p[10],i,j,k,temp1,temp2,temp3,n,id[10],f;
 int at1[10],bt1[10];
 float atat=0,awt=0;
 printf("enter the no.of processes:");
 scanf("%d",&n);
 printf("enter the arrival time:");
 for(i=0;i<n;i++)
 {
  scanf("%d",&at[i]);
  p[i]=i+1;
  at1[i]=0;
```

```
}
printf("enter the burst times:");
for(i=0;i<n;i++)
{
 scanf("%d",&bt[i]);
}
for(i=0;i<n;i++)
{
 for(j=0;j<n-1;j++)
 {
  if(at[j]>at[j+1])
  {
   temp1=at[j];
   at[j]=at[j+1];
   at[j+1]=temp1;
   temp2=bt[j];
   bt[j]=bt[j+1];
   bt[j+1]=temp2;
   temp3=p[j];
   p[j]=p[j+1];
   p[j+1]=temp3;
  }   }}
for(i=0;i<n;i++)
{
 for(j=1;j<n;j++)
 {
  if(at[i]==at[j])
  {
   if(bt[i]>bt[j])
   {
```

```
    temp1=bt[i];
    bt[i]=bt[j];
    bt[j]=temp1;
    temp1=p[i];
    p[i]=p[j];
    p[j]=temp1;
  } }  } }
for(i=1;i<n;i++)
{
 for(j=1;j<n-1;j++)
 {
  if(bt[j]>bt[j+1])
  {
   temp1=at[j];
   at[j]=at[j+1];
   at[j+1]=temp1;
   temp1=bt[j];
   bt[j]=bt[j+1];
   bt[j+1]=temp1;
   temp1=p[j];
   p[j]=p[j+1];
   p[j+1]=temp1;
  }  } }
k=0;
id[0]=0;
ct[0]=at[0]+bt[0];
for(i=1;i<n;i++)
{
 id[i]=at[i]-ct[i-1];
 if(id[i]>0)
```

```
 ct[i]=bt[i]+id[i]+ct[i-1];
 else
 ct[i]=bt[i]+ct[i-1];
}
for(i=0;i<n;i++)
{
 tat[i]=ct[i]-at[i];
 wt[i]=tat[i]-bt[i];
 atat+=tat[i];
 awt+=wt[i];
}
printf("p\tat\tbt\tct\ttat\twt\t\n");
for(i=0;i<n;i++)
{
 printf("p%d\t%d\t%d\t%d\t%d\t%d\t\n",p[i],at[i],bt[i],ct[i],tat[i],wt[i]);
}
printf("avg_tat:%f \t\t avg_wt:%f",(atat/n),(awt/n));
}
```

b)SRTF

```
include<stdio.h>
void main()
{
int AT[10],BT[10],x[10],P[10],pr[50],s=0,process[10];
int WT[10],TAT[10],CT[10];
int i,j,min,c=0,ct,n,time,l=0;
double avg_WT=0,avg_TAT=0,end;
 printf("Enter the number of Processes: ");
scanf("%d",&n);
printf("Enter arrival time of process " );
for(i=0;i<n;i++)
```

```
{
   scanf("%d",&AT[i]);
}
 printf("Enter burst time of process");
for(i=0;i<n;i++)
{
    scanf("%d",&BT[i]);
  P[i]=i+1;
}
for(i=0;i<n;i++)
x[i]=BT[i];
  BT[9]=345;
  for(time=0;c!=n;time++)
{
 min=9;
 for(i=0;i<n;i++)
 {
  if(AT[i]<=time && BT[i]<BT[min] && BT[i]>0 )
  min=i;
  else if(BT[i]==BT[min])
  {
   if(AT[i]<AT[min])
   min=i;
   else
   min=min;
  }
 }
   BT[min]--;
 pr[l]=P[min];
 l++;
```

```
    if(BT[min]==0)
 {
  c++;
  ct=time+1;
  CT[min] = ct;
 WT[min] = ct- AT[min] - x[min];
 TAT[min] = ct - AT[min];
 process[s]=P[min];
 s++;
   }
}
 printf("PNO \t burst \t arrival \twaiting \tturnaround \tcompletion");
 for(i=0;i<n;i++)
 {
 printf("\n %d \t   %d \t %d\t\t%d   \t\t%d\t\t%d",i+1,x[i],AT[i],WT[i],TAT[i],CT[i]);
   avg_WT = avg_WT + WT[i];
   avg_TAT =avg_TAT + TAT[i];
 }
   printf("\n\nAverage waiting time = %lf\n",avg_WT/n);
 printf("Average Turnaround time = %lf\n",avg_TAT/n);
 for(i=0;i<l;i++)
printf("%d\t",pr[i]);
 printf("\nGANTT CHART\n");
for(i=0;i<n;i++)
printf("p%d\t",process[i]);
}
```

WEEK -2

Write a C program to simulate the following non preemptive CPU Scheduling algorithms to find turnaround time, waiting time, average turnaround time and average waiting time.

a) HRRN

```
#include<stdio.h>
 int main()
{
 float AT[10],BT[10],CT[10],TAT[10],WT[10],I[10],GC[10],P[10],process[1 0],ATT[10], hit=0;
   int n,temp,temp1,i,j;
 float avg_WT,avg_TAT,HT[10];
 printf("enter the no of process");
 scanf("%d",&n);
 printf("enter the AT");
 for( i=0;i<n;i++)
 {
   scanf("%f",&AT[i]);
   P[i]=i+1;
  HT[i]=0;
 }

 printf("enter the BT");
 for( i=0;i<n;i++)
 {
   scanf("%f",&BT[i]);
 }
 for(i=0;i<n;i++)
 {
  for(j=0;j<n-1;j++)
  {
    if(AT[j]>AT[j+1])
    {
      temp=AT[j];
     AT[j]=AT[j+1];
     AT[j+1]=temp;
```

```
    temp1=BT[j];
    BT[j]=BT[j+1];
    BT[j+1]=temp1;
    int temp2=P[j];
    P[j]=P[j+1];
    P[j+1]=temp2;
  } } }
int k=0,l,temp2,index;
float max=-45345,ratio=0;
 CT[0]=AT[0]+BT[0];
 process[0]=P[0];
 temp2=CT[0];
for(i=0;i<n;i++)
{
    l=i+1;
    hit=0;
    ratio=0;
    int min=9999;
 max=-45345;
 for(j=i+1;j<n;j++)
 {
  if(AT[j]<=temp2)
  {
   ratio=((temp2-AT[j])/BT[j]);
      printf("ratio  of %d is %f\n",j,ratio);
   hit=1;
  }
    if(ratio>max)
  {
   max=ratio;
```

```
   index=j;
   printf("max %f\n",max);
  }
 }
  if(hit==1)
  {
      temp=P[index];
   P[index]=P[l];
   P[l]=temp;
     temp=BT[index];
   BT[index]=BT[l];
   BT[l]=temp;
     temp=AT[index];
   AT[index]=AT[l];
   AT[l]=temp;

   CT[l]=BT[l]+CT[i];
   temp2=CT[l];
   printf("final max %f\n",max);
 printf("\n");
    }
    else
    {
     printf("idle\n");
     CT[l]=BT[l]+CT[i]+(AT[l]-CT[i]);
     temp2=CT[l];
     P[l]=P[l];
 } }
  for(i=0;i<n;i++)
{
```

```
 TAT[i]=CT[i]-AT[i];
}
 for(i=0;i<n;i++)
{
 WT[i]=TAT[i]-BT[i];
}
  for(i=0;i<n;i++)
{
 avg_TAT+=TAT[i];
 avg_WT+=WT[i];
}
 printf("PNO|\tAT|\tBT|\tCT|\tTAT|\tWT\n");
 for(i=0;i<n;i++)
 {
 printf("p%2.f\t%2.f\t%2.f\t%2.f\t%2.f\t%2.f\n",P[i],AT[i],BT[i],C T[i],TAT[i],WT[i]);
 }
 printf("AVG_TAT IS %2.f\n",avg_TAT/n);
 printf("AVG_WT IS %2.f\n",avg_WT/n);
   printf("\n");
 printf("gantt chart\t");
 for(i=0;i<j;i++)
 {
  printf("p%2.f\t",P[i]);
  }
}
```

b) LRTF

```
#include<stdio.h>
 void main()
{
 int AT[10],BT[10],x[10];
```

```
int WT[10],TAT[10],CT[10],P[10],pr[10];
int i,j,min,count=0,time,n,k=0;
double avg_WT=0,avg_TAT=0,end;
 printf("Enter the number of Processes: ");
scanf("%d",&n);
printf("Enter arrival time of process " );
for(i=0;i<n;i++)
{
   scanf("%d",&AT[i]);
}
 printf("Enter burst time of process");
for(i=0;i<n;i++)
{ scanf("%d",&BT[i]);
  P[i]=i+1;
}
for(i=0;i<n;i++)
x[i]=BT[i];
  BT[9]=-53;
 for(time=0;count!=n;time++)
{
 min=9;
 for(i=0;i<n;i++)
 {
  if(AT[i]<=time && BT[i]>BT[min] && BT[i]>0 )
  min=i;
  if(BT[i]==BT[min])
  {
   if(AT[i]<AT[min])
   min=i;
   else
```

```
   min=min;
  }
   }
 printf("p%d\t",P[min]);
 BT[min]--;
   if(BT[min]==0)
 {
  pr[k]=P[min];
  k++;
  count++;
  end=time+1;
  CT[min] = end;
  WT[min] = end - AT[min] - x[min];
  TAT[min] = end - AT[min]; }
}
 printf("\n");
printf("pid \t burst \t arrival \twaiting \tturnaround \tcompletion");
 for(i=0;i<n;i++)
 {
  printf("\n %d \t   %d \t %d\t\t%d   \t\t%d\t\t%d",i+1,x[i],AT[i],WT[i],TAT[i],CT[i]);
  avg_WT = avg_WT + WT[i];
  avg_TAT =avg_TAT + TAT[i];
}  printf("\n\nAverage waiting time = %lf\n",avg_WT/n);
 printf("Average Turnaround time = %lf",avg_TAT/n);
  printf("\n");
 printf("gantt chart\n");
 for(i=0;i<n;i++)
 {
  printf("p%d\t",pr[i]);
 } }
```

WEEK-3

1. Write a C program to simulate the following preemptive CPU Scheduling algorithms to find turnaround time and waiting time, average turnaround time and average waiting time. c) Priority

Program:

```
#include<stdio.h>
 void main()
{
   int WT[10],TAT[10],CT[10],PR[10],AT[10],BT[10],D[10];;
 int i,j,min,c=0,time,n;
 double avg_WT=0,avg_TAT=0,ct;
  printf("Enter the number of Processes: ");
 scanf("%d",&n);
 printf("Enter arrival time of process " );
 for(i=0;i<n;i++)
 {
    scanf("%d",&AT[i]);
 }
  printf("Enter burst time of process");
 for(i=0;i<n;i++)
 {
    scanf("%d",&BT[i]);
 }
  printf("Enter priority time of process");
 for(i=0;i<n;i++)
 {
  scanf("%d",&PR[i]);
 }
  for(i=0;i<n;i++)
 D[i]=BT[i];
  BT[9]=345;
```

```
 PR[9]=345;
  for(time=0;c!=n;time++)
{ min=9;
 for(i=0;i<n;i++)
 {
  if(AT[i]<=time && PR[i]<PR[min] && BT[i]>0 )
  min=i;
   else if(PR[i]==PR[min])
  {
   if(AT[i]<AT[min])
   min=i;
   else
   min=min;
  } }
 BT[min]--;

 if(BT[min]==0)
 {
  c++;
  ct=time+1;
  CT[min] = ct;
 WT[min] = ct - AT[min] - D[min];
 TAT[min] = ct - AT[min];
    }
}
printf("pno \t arrival \tburst \t completion\t tturnaround\t waiting ");
for(i=0;i<n;i++)
{
  printf("\n %d \t   %d \t %d\t\t%d   \t\t%d\t\t%d",i+1,AT[i],D[i],CT[i],TAT[i],WT[i]);
  avg_WT = avg_WT + WT[i];
```

```
  avg_TAT =avg_TAT + TAT[i];
}
  printf("\n\nAverage waiting time = %lf\n",avg_WT/n);
printf("Average Turnaround time = %lf",avg_TAT/n);
}
```

WEEK-4

Write a C program to simulate multi-level queue scheduling algorithm considering the following scenario. All the processes in the system are divided into two categories – system processes and user processes. System processes are to be given higher priority than user processes. Consider each process priority to be from 1 to 3. Use priority scheduling for the processes in each queue.

Program:

```
#include<stdio.h>
void sort(int *q1at,int *q1bt,int *q1pid,int t);
void sort1(int *q2at,int *q2bt,int *q2pid,int t1);
 void main()
{
 int at[10],bt[10],pr[10];
 int at1[10],bt1[10],tat1[10],wt1[10],ct1[10],i1[10],p1[10];
 int at2[10],bt2[10],tat2[10],wt2[10],ct2[10],p2[10];
 int n1=0,n2=0,check[2],n,max=0,min,p[10],i,j,m,s,sumb=0,ind,temp;
 float avg_TAT,avg_WT;
  int f=0;
 int tct=0;
 float avg=0;
 printf("enter number of proesses\n");
 scanf("%d",&n);
 printf("enter arrival time and burst time\n");
 for(i=0;i<n;i++)
 {
  scanf("%d %d",&at[i],&bt[i]);
  p[i]=i+1;
```

```
 check[i]=0;
}
printf("enter the priority of the queue");
for(i=0;i<2;i++)
{
 scanf("%d",&pr[i]);
}
for(i=0;i<n;i++)
{
 sumb=sumb+bt[i];
}
   avg=(sumb/n);
//printf("avg:%f",avg);
for(i=0;i<n;i++)
{
 if(bt[i]>avg)
 {
  at1[n1]=at[i];
  bt1[n1]=bt[i];
  p1[n1]=p[i];
  n1++;
 }
 else
 {
  at2[n2]=at[i];
  bt2[n2]=bt[i];
  p2[n2]=p[i];
  n2++;
 }
}
```

```
 for(s=0;s<2;s++)
{
 max=999;
 for(m=0;m<2;m++)
 {
  if(pr[m]<max && check[m]!=1)
  {
   max=pr[m];
   ind=m; } }
 if(ind==0)
 {
  temp=0;
  printf("_____FCFS_____\n");
  sort(at1,bt1,p1,n1);
  check[0]=1;
  if(f==0)
  ct1[0]=bt1[0]+at1[0];
  else
  ct1[0]=bt1[0]+tct;
  i1[0]=0;
  for(i=1;i<n1;i++)
  {
   i1[i]=at1[i]-ct1[i-1];
   if(i1[i]>0)
   ct1[i]=ct1[i-1]+bt1[i]+i1[i];
   else
   ct1[i]=ct1[i-1]+bt1[i];
    temp=ct1[i];
  }
  tct=temp;
```

```
  f=1;
  printf("%d tct\n",tct);
   for(i=0;i<n1;i++)
  {
   tat1[i]=ct1[i]-at1[i];
   wt1[i]=tat1[i]-bt1[i];
  }
 for(i=0;i<n1;i++)
   {
   avg_TAT+=tat1[i];
   avg_WT+=wt1[i];
   }
 printf("PNO|\t AT|\t BT|\t CT|\t TAT|\t WT\n");
    for(i=0;i<n1;i++)
    {
      printf("%d|\t %d|\t %d|\t %d| \t %d|\t %d\n",p1[i],at1[i],bt1[i],ct1[i],tat1[i],wt1[i]);
    }
    printf("avg TAT is:  %f",(avg_TAT/n1));
    printf("avg WT is: %f",(avg_WT/n1));
    printf("\n");
    printf("GANTT CHART\n ");
    for(i=0;i<n1;i++)
    printf("p%d\t",p1[i]);
    printf("\n");
  }
else
{
 printf("tct:%d\n",tct);
 printf("++++++++++++++++++SRJf++++++++++++++++++\n");
 int x[10],time,c=0,l=0,ct=0;
```

```
check[1]=1;
for(i=0;i<n2;i++)
 x[i]=bt2[i];
 bt2[9]=345;
  for(time=0;c!=n2;time++)
{
min=9;
 for(i=0;i<n2;i++)
 {
  if(at2[i]<=time && bt2[i]<bt2[min] && bt2[i]>0 )
  min=i;
 }
 if(bt2[min]>0)
 bt2[min]--;
  pr[l]=p2[min];

 l++;
   if(bt2[min]==0)
 {
  c++;
   ct=time+1;
    if(f==0)
  {
   ct2[min] = ct;
   tat2[min] = ct - at2[min];
 wt2[min] = tat2[min]-x[min];
    }
  else if(f==1)
  {   ct2[min]=ct+tct;
    tat2[min] = ct2[min] - at2[min];
```

```
  wt2[min] = tat2[min]-x[min];
   temp=ct2[min];
   }
   printf("p%d\t",p2[min]);
}
}
f=1;
printf("\nPNO \t burst \t arrival \twaiting \tturnaround \tcompletion");
for(i=0;i<n2;i++)
{
  printf("\n %d \t   %d \t %d\t\t%d   \t\t%d\t\t%d",p2[i],x[i],at2[i],wt2[i],tat2[i],ct2[i]);
  avg_WT = avg_WT + wt2[i];
  avg_TAT =avg_TAT + tat2[i];
 tct=ct2[i];
}
 printf("\ntct of srjf:%d\n",tct);
printf("\n\nAverage waiting time = %lf\n",avg_WT/n2);
printf("Average Turnaround time = %lf",avg_TAT/n2);
printf("\n");
} } }
void sort(int *q1at,int *q1bt,int *q1pid,int t)
{
 int i,j,temp;
 for(i=0;i<t-1;i++)
 {
  for(j=i+1;j<t;j++)
  {
   if(q1at[i]>q1at[j])
   {
    temp=q1at[i];
```

```
   q1at[i]=q1at[j];
   q1at[j]=temp;
   temp=q1bt[i];
   q1bt[i]=q1bt[j];
   q1bt[j]=temp;
   temp=q1pid[i];
   q1pid[i]=q1pid[j];
   q1pid[j]=temp;
  }  } }
```

WEEK-10

Write a C program to simulate page replacement algorithms

a)Reference Bit b)MFU

a)Reference Bit

Program:

```
#include<stdio.h>
#include <math.h>
//3 2 3 8 0 3 3 0 2 6 3 4 7
//7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
void main()
{
 int r,f,j,k,m,ind,l=0,min,s,pf=0,freq[10],max,maxx,sum,sumb[100],limit,limi,fra=0;
 printf("enter the no of frame:\n");
 scanf("%d",&f);
 int n=f;
 printf("enter the interval: ");
 scanf("%d",&limit);
 printf("enter the no of reference string:\n");
 scanf("%d",&r);
   int frame[f],ref[r],i,index[100][100],check[f];
```

```
printf("enter the refence string:\n");
for(i=0;i<r;i++)
{
 scanf("%d",&ref[i]);
}
for(i=0;i<n;i++)
{
 frame[i]=-1;
 check[i]=0;
   }
for(i=0;i<n;i++)
{
 for(j=0;j<8;j++)
 {
  index[i][j]=0;
 }
}
  limi=0;
for(i=0;i<r;i++)
{
 limi++;
// printf("%d",limi);
 f=0;
 for(j=0;j<n;j++)
 {
  if(ref[i]==frame[j])
  {
   f=1;
   k=j ;
   break;
```

```
 }
}
if(f==1)
{
 if(index[k][0]==0)
 {
  index[k][0]=1;
 }
 goto label;
}
 else if(f==0)
{
 pf++;
 if(frame[fra]==-1)
{
 frame[fra]=ref[i];
 index[fra][0]=1;
 fra++;
}
else
{
 for(k=0;k<n;k++)
 {
  s=0;
  sum=0;
  for(m=7;m>-1;m--)
  {
   sum+=index[k][m]*pow(2,s);
  // printf("sum: %d",sum);
   s++;
```

```
  }
   //printf("sum:%d",sum);
   sumb[k]=sum;
     }
  for(s=0;s<n;s++)
 // printf("%d\t",sumb[s]);
  printf("\n");
  int min=999;
  for(k=0;k<n;k++)
  {
   if(sumb[k]<min)
   {
    min=sumb[k];
    ind=k;
   }  }
   frame[ind]=ref[i];
   for(k=0;k<8;k++)
  {
   index[ind][k]=0;
  }
   index[ind][0]=1;
      }
 }
 goto label;
label: if(limit==limi)
 {
  limi=0;
  for(k=0;k<n;k++)
  {
   for(l=7;l>-1;l--)
```

```
    {
     index[k][l]=index[k][l-1];
    }
   }
  for(k=0;k<n;k++)
   {
     index[k][0]=0;
     } }
 for(s=0;s<n;s++)
  printf("%d\t",frame[s]);
  printf("\n");
     /* printf("------------index--------------------\n");
  for(s=0;s<n;s++)
  {
   for(k=0;k<8;k++)
   {
    printf("%d\t",index[s][k]);
   }
   printf("\n");
  }
  printf("-----------------------------------------------------\n");
  */ }
printf("no of page faults:%d ",pf);
}
```

b)MFU

Program:

```
#include<stdio.h>
main()
{
 int r,f,j,k,m,ind,l=0,min,s,pf=0,freq[10],max,maxx;
```

```
printf("enter the no of frame:\n");
scanf("%d",&f);
int n=f;
printf("enter the no of reference string:\n");
scanf("%d",&r);
int frame[f],ref[r],i,index[f],check[f];
printf("enter the refence string:\n");
for(i=0;i<r;i++)
{
 scanf("%d",&ref[i]);
}
for(i=0;i<f;i++)
{
 frame[i]=-1;
 check[i]=0;
 index[i]=-1;
}
for(i=0;i<r;i++)
{
 f=0;
 for(k=0;k<n;k++)
 {
 if(frame[k]==ref[i])
 {
   f=1;
   j=k;
   break;
   }
 }
 if(f==0)
```

```
{
 pf++;
 if(frame[l]==-1)
 {
  frame[l]=ref[i];
   freq[l]=1;
 index[i]=i;
 l++; }
else
{
max=-9999;int indi=0;
 int minn=8999;
 for(k=0;k<n;k++)
 {
  if(freq[k]>max)
  {
   max=freq[k];
   ind=k;
  }
 }
 for(k=0;k<n;k++)
 {
  if(freq[k]==max && index[k]<minn)
  {
   minn=index[k];
   ind=k;
  } }
   frame[ind]=ref[i];
 freq[ind]=1;
 index[ind]=i;
```

```
  }
 }
 if(f==1)
 {
 // index[j]=i;
  freq[j]+=1;
  //printf("hit\n");
 }
 for(s=0;s<n;s++)
 printf("%d\t",frame[s]);
 printf("\n");
   printf("\n");
 }
printf("no of page faults:%d",pf);
}
```

WEEK-11

12.Write a C program to simulate Resource request allocation algorithm for the purpose of deadlock avoidance

Program:

```
#include<stdio.h>
#include<dos.h>
void main()
 int n,m;
{
printf("enter the number of process:");
 scanf("%d",&n);
 printf("no of resources:\n");
 scanf("%d",&m);
 int i,j,k,finish[n],allocation[10][10],max[10][10],need[10][10],work[10][10],request[1 0][10],index,l=0;
 for(i=0;i<n;i++)
```

```
finish[i]=0;
printf("enter the allocation matrix\n");
for(i=0;i<n;i++)
{
 for(j=0;j<m;j++)
 {
  scanf("%d",&allocation[i][j]);
 }
}
printf("enter the max matrix\n");
for(i=0;i<n;i++)
{
 for(j=0;j<m;j++)
 {
  scanf("%d",&max[i][j]);
 }
}
 printf("enter the available resources:\n");
for(i=0;i<1;i++)
{
 for(j=0;j<m;j++)
 {
   scanf("%d",&work[i][j]);
 }
  }
  for(i=0;i<n;i++)
{
 for(j=0;j<m;j++)
 {
  need[i][j]=max[i][j]-allocation[i][j];
```

```
 }
}
for(i=0;i<n;i++)
{
 for(j=0;j<m;j++)
 {
  printf("%d\t",need[i][j]);
 }
 printf("\n");
}
  printf("enter the process number:");
scanf("%d",&index);
printf("enter the request matrix");
  for(j=0;j<m;j++)
 {
  scanf("%d",&request[index][j]);
 }
  for(i=0;i<m;i++)
{
 if(request[index][i]<=need[index][i])
 {
  if(request[index][i]<=work[0][i])
    l++;
 }
}
if(l==m)
{   printf("request is granted");
   for(i=0;i<m;i++)
 {
  work[0][i]-=request[index][i];
```

```
  allocation[index][i]+=request[index][i];
  need[index][i]-=request[index][i];
 }
   int count=0;
int c=0;
printf("The sequence is: ");
label:for(i=0;;i++)
{ count=0;
 if(finish[i]==0)
 {
  for(k=0;k<m;k++)
  {
    if(need[i][k]<=work[0][k])
    {
     count++;
    }
   }
   if(count==m)
 {
  for(k=0;k<m;k++)
  {
   work[0][k]=work[0][k]+allocation[i][k];
  }
  c++;
   finish[i]=1;
  printf("%d\t",i);
 } }
 if(i==n-1)
 goto label;
 if(c==n)
```

```
  break;
 }
}
else
 printf("not possible"); }.
```

WEEK-12

13.Write a C program to simulate strict alternation problem using Threads?

Program:

```
#include <stdio.h>
#include <pthread.h>
 pthread_t thread1,thread2;
int turn;
  void init()
{
turn=0;
 }
 void* print_once(void *s)
{
 int self = (int *)s;
  //printf("%d",self);
while(1)
 {
  while(turn!=0 );
 {
 printf("\n First process is in CS\n");
   turn=1;
     sleep(3);
  // printf("%d",self);
}
```

```
 while(turn!=1);
  {
 printf("\n Second thread is in CS\n");
    // printf("%d",self);
  turn=0;
    sleep(3);
} }      }
void main()
{
 init();
 pthread_create(&thread1, NULL, print_once, (void*)0);
 pthread_create(&thread2, NULL, print_once, (void*)1);
    pthread_join(thread1, NULL);     pthread_join(thread2, NULL);
     }
```

WEEK-13

14. Write a C program to simulate peterson's solution.

```
 #include<stdio.h>
void main()
 {
 int flag[5];
  int turn,i;
printf("take 1 as true and 0 as false");
  printf("enter flag values\n");
 for(i=0;i<2;i++)
{
scanf("%d",&flag[i]);
  }
 if(flag[0]==1 && flag[1]==0)
```

```
 {
flag[0]=1;
  turn=1;
  while(turn==1 && flag[1]==1);
  printf("p0 is in critical section\n");
 printf("p1 is waiting\n");
 flag[0]=0;
  }
 else if(flag[0]==0 && flag[1]==1)
{
   flag[1]=1;
 turn=0;
  while(turn==0 && flag[0]==1);
  printf("p1 is in critical section\n");
  printf("p0 is waiting\n");
   flag[1]=0;
 }
 else if(flag[0]==1 && flag[1]==1)
 {
printf("two process cant be in critical section\n");
 }  else
 {
  printf("critical section is empty\n");
} }
```

WEEK-10

Write a C program to simulate disk scheduling algorithms

a)sstf

Program:

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

```
#include<dos.h>
// 98 183 37 122 14 124 65 67
void main()
{
 int n,i,j,k,arr[25],flag[20],f,m=0,l,h,sub[20],ind,fl=0,min;
 printf("enter n:");
 scanf("%d",&n);
 printf("enter the limit");
 scanf("%d",&l);
 printf("enter the request:");
 for(i=0;i<n;i++)
 {
  scanf("%d",&arr[i]);
  flag[i]=0;
 }
 printf("enter head:");
 scanf("%d",&h);
 i=h;
 while(1)
 {
  fl=0;
  min=999;
  k=0;
  for(j=0;j<n;j++)
  {
    sub[j]=abs(arr[j]-h);
  }
  for(j=0;j<n;j++)
  {
   if(sub[j]<min&&flag[j]==0)
```

```
 {
  min=sub[j];
  ind=j;
 }
}
if(flag[ind]==0)
{
    m=m+abs((arr[ind]-h));
 flag[ind]=1;
 h=arr[ind];
   }
  for(j=0;j<n;j++)
{
 if(flag[j]==1)
 fl++;
 sub[j]=-1;
 //printf("flag:%d\t",flag[j]);
}
if(fl==n)
break;
}
printf("%d",m);  }
```