OS CODETANTRA

LAB – 4 / 1-39

Write a program to implement the FCFS process scheduling algorithm.

```
#include<stdio.h>
#include<conio.h>
#define max 30
int main(){
int n,i,pn[max],at[max],bt[max],wt[max],tat[max],start[max],finish[max];
float awt=0,atat=0;
printf("Enter the number of processes: ");
scanf("%d",&n);
for(i=0;i<n;i++){
printf("Enter the Process Name, Arrival Time & Burst Time:");
scanf("%d%d%d",&pn[i],&at[i],&bt[i]);

}
printf("Process Name\tArrival Time\tBurst Time\n");
for(i=0;i<n;i++){
printf("  %d\t    %d\t    %d\n",pn[i],at[i],bt[i]);
}
printf("PName   Arrtime   Bursttime   Start   WT  TAT  Finish\n");
start[0]=at[0];
finish[0]=start[0]+bt[0];
for(i=0;i<n;i++){
if(i>0){
start[i]=finish[i-1];
}
finish[i]=start[i]+bt[i];
wt[i]=start[i]-at[i];
tat[i]=bt[i]+wt[i];

}
printf("%d\t  %d\t\t  %d\t  %d\t  %d\t  %d\t  %d\n",pn[0],at[0],bt[0],start[0],wt[0],tat[0],finish[0]);
for(i=1;i<n;i++){
  printf("%d\t  %d\t\t  %d\t  %d\t  %d\t  %d\t  %d\n",pn[i],at[i],bt[i],start[i],wt[i],tat[i],finish[i]);

}
for(i=0;i<n;i++){
awt+=wt[i];
atat+=tat[i];
}
awt=awt/n;
atat=atat/n;
printf("Average Waiting time:%f",awt);
printf("\nAverage Turn Around Time:%f",atat);
return 0;
}
```

LAB – 4 / 2-40

Write a program to implement the SJF Scheduling Algorithm.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main(){
int et[20],at[10],n,i,j,temp,st[10],ft[10],wt[10],ta[10];
int totwt=0,totta=0;
float awt,ata;
char pn[10][10],t[10];
printf("Enter the number of process:");
scanf("%d",&n);
for(int i=0;i<n;i++){
printf("Enter process name, arrival time & execution time:");
scanf("%s%d%d",pn[i],&at[i],&et[i]);
}
for(i=0;i<n;i++){
for(j=0;j<n;j++){
if(et[i]<et[j]){
temp=at[i];
at[i]=at[j];
at[j]=temp;
temp=et[i];
et[i]=et[j];
et[j]=temp;
strcpy(t,pn[i]);
strcpy(pn[i],pn[j]);
strcpy(pn[j],t);
}
}
}
for(i=0;i<n;i++){
if(i==0)
st[i]=at[i];
else
st[i]=ft[i-1];
wt[i]=st[i]-at[i];
ft[i]=st[i]+et[i];
ta[i]=ft[i]-at[i];
totwt+=wt[i];
totta+=ta[i];
}
awt=(float)totwt/n;
ata=(float)totta/n;
printf("Pname\tarrivaltime\texecutiontime\twaitingtime\ttatime\n");
for(i=0;i<n;i++)
printf("%s\t%5d\t\t%5d\t\t%5d\t\t%5d\n",pn[i],at[i],et[i],wt[i],ta[i]);
printf("Average waiting time is:%f\n",awt);
printf("Average turnaroundtime is:%f\n",ata);
}
```

LAB – 4 / 3-41

Write a program to implement the PRIORITY based cpu scheduling algorithm.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main(){
  int bt[20],at[10],n,i,j,temp,st[10],ft[10],wt[10],ta[10],p[10];
  int totwt=0,totta=0;
  float awt,ata;
  char pn[10][10],t[10];
  printf("Enter the number of process:");
  scanf("%d",&n);
  for(int i=0;i<n;i++){
    printf("Enter process name,arrivaltime,execution time & priority:");
    scanf("%s%d%d%d",pn[i],&at[i],&bt[i],&p[i]);
  }
  for(i=0;i<n;i++){
    for(j=0;j<n;j++){
      if(p[i]<p[j]){
        temp=p[i];
        p[i]=p[j];
        p[j]=temp;
        temp=at[i];
        at[i]=at[j];
        at[j]=temp;
        strcpy(t,pn[i]);
        strcpy(pn[i],pn[j]);
        strcpy(pn[j],t);
      }
    }
}
for(i=0;i<n;i++){
  if(i==0)
  st[i]=at[i];
  else
  st[i]=ft[i-1];
  wt[i]=st[i]-at[i];
  ft[i]=st[i]+bt[i];
  ta[i]=ft[i]-at[i];
  totwt+=wt[i];
  totta+=ta[i];
}
  awt=(float)totwt/n;
  ata=(float)totta/n;
  printf("Pname\tarrivaltime\texecutiontime\tpriority\twaitingtime\ttatime\n");
   printf("%s\t  %d\t\t  %d\t\t  %d\t\t  %d\t\t  %d\n",pn[0],at[0],bt[0],p[0],wt[0],ta[0]);
  for(i=1;i<n;i++)
    printf("%s\t  %d\t\t  %d\t\t  %d\t\t  %d\t\t  %d\n",pn[i],at[i],bt[i],p[i],wt[i],ta[i]);
  printf("Average waiting time is:%f\n",awt);
  printf("Average turnaroundtime is:%f\n",ata);
}
```

LAB – 5 / 1-42

Implementation of the Round Robin cpu scheduling algorithm

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#define max 50
void main(){
  int
i,n,sum=0,count=0,y,quant,wt=0,tat=0,aTime[max],bTime[max],temp[max],wTime[max],rem_bTime[max],taTime[max];
  float avg_wt,avg_tat;
  printf("Enter Total Number of Processes: ");
  scanf("%d",&n);
  y=n;
  for(i=0; i<n; i++){
    printf("Enter Details of Process[%d]: Arrival Time:\t",i+1);
    scanf("%d",&aTime[i]);
    printf("Burst Time:\t");
    scanf("%d",&bTime[i]);
    temp[i]=bTime[i];
  }
  printf("Enter Time Quantum:\t");
  scanf("%d",&quant);
  printf("Process ID\t\tBurst Time\t Turnaround Time\t Waiting Time\n");

    for(sum=0, i=0; y!=0;){
      if(temp[i]<= quant && temp[i]>0){
        sum=sum+temp[i];
        temp[i]=0;
        count=1;
      }
      else if(temp[i]>0){
        temp[i]=temp[i]-quant;
        sum=sum+quant;
      }
      if(temp[i] == 0 && count == 1){
        y--;
        printf("Process[%d]\t\t%d\t\t %d\t\t\t %d\n",i+1,bTime[i],sum-aTime[i],sum-aTime[i]-bTime[i]);
        wt=wt+sum-aTime[i]-bTime[i];
        tat=tat+sum-aTime[i];
        count=0;
      }
      if(i==n-1){
        i=0;
      }
      else if(aTime[i+1]<=sum){
        i++;
      }
      else{
        i=0;
      }
    }

      avg_wt=(float)wt/n;
      avg_tat=(float)tat/n;
      printf("Average Waiting Time:\t%f\n",avg_wt);
      printf("Avg Turnaround Time:\t%f\n",avg_tat);
}
```

LAB – 5 / 2-43

Write a program to implement the Multi Level Queue Scheduling

```
#include<stdio.h>
int main()
{
  int p[20],bt[20], su[20], wt[20],tat[20],i, k, n, temp;
  float wtavg, tatavg;
  printf("Enter the number of processes:");
  scanf("%d",&n);
  for(i=0;i<n;i++)
  {
    p[i] = i;
    printf("Enter the Burst Time of Process %d:",i);
    scanf("%d",&bt[i]);
    printf("System/User Process (0/1) ?");
    scanf("%d", &su[i]);
  }
  for(i=0;i<n;i++)
  for(k=i+1;k<n;k++)
  if(su[i] > su[k])
  {
    temp=p[i];
    p[i]=p[k];
    p[k]=temp;
    temp=bt[i];
    bt[i]=bt[k];
    bt[k]=temp;
    temp=su[i];
    su[i]=su[k];
    su[k]=temp;
  }
  wtavg = wt[0] = 0;
  tatavg = tat[0] = bt[0];
  for(i=1;i<n;i++)
  {
    wt[i] = wt[i-1] + bt[i-1];
    tat[i] = tat[i-1] + bt[i];
    wtavg = wtavg + wt[i];
    tatavg = tatavg + tat[i];
  }
  printf("PROCESS\t\t SYSTEM/USER PROCESS \tBURST TIME\tWAITING TIME\tTURNAROUND TIME\n");
  for(i=0;i<n;i++)
  printf("%d \t\t %d \t\t %d \t\t %d \t\t %d \n",p[i],su[i],bt[i],wt[i],tat[i]);
  printf("Average Waiting Time is --- %f\n",wtavg/n);
  printf("Average Turnaround Time is --- %f\n",tatavg/n);
  return 0;
}
```

LAB – 6 / 1-65

Write the C program to implement Banker's Algorithm

```c
#include<stdio.h>
#include<conio.h>
int main()
{
  int n,r,i,j,k,p,u=0,s=0,m;
  int block[10],run[10],active[10],newreq[10];
  int max[10][10],resalloc[10][10],resreq[10][10];
  int totalloc[10],totext[10],simalloc[10];
  printf("Enter the no of processes: ");
  scanf("%d",&n);
  printf("Enter the no of resource classes: ");
  scanf("%d",&r);
  printf("Enter the total existed resource in each class: ");
  for(k=1; k<=r; k++)
  scanf("%d",&totext[k]);
  printf("Enter the allocated resources: ");
  for(i=1; i<=n; i++)
  for(k=1; k<=r; k++)
  scanf("%d",&resalloc);
  printf("Enter the process making the new request: ");
  scanf("%d",&p);
  printf("Enter the requested resource: ");
  for(k=1; k<=r; k++)
  scanf("%d",&newreq[k]);
  printf("Enter the process which are n blocked or running\n");
  for(i=1; i<=n; i++)
  {
    if(i!=p)
    {
      printf("process %d: \n",i+1);
      scanf("%d%d",&block[i],&run[i]);
    }
  }
  block[p]=0;
  run[p]=0;
  for(k=1; k<=r; k++)
  {
    j=0;
    for(i=1; i<=n; i++)
    {
      totalloc[k]=j+resalloc[i][k];
      j=totalloc[k];
    }
  }
  for(i=1; i<=n; i++)
  {
    if(block[i]==1||run[i]==1)
    active[i]=1;
    else
    active[i]=0;
  }
  for(k=1; k<=r; k++)
  {

      resalloc[p][k]+=newreq[k];
      totalloc[k]+=newreq[k];
  }
  for(k=1; k<=r; k++)
  {
    if(totext[k]-totalloc[k]<0)
    {
      u = 1;
      break;
    }
  }
  if(u==0)
  {
    for(k=1; k<=r; k++)
   simalloc[k]=totalloc[k];
    for(s=1; s<=n; s++)
    for(i=1;i<=n;i++)
    {
      if(active[i]==1)
      {
        j=0;
        for(k=1; k<=r; k++)
        {
          if((totext[k]-simalloc[k])<(max[i][k]-resalloc[i][k]))
          {
            j=1;
            break;
          }
        }
      }
      if(j==0)
      {
        active[i];
        for(k=1; k<=r; k++)
       simalloc[k]=resalloc[i][k];
      }
    }
    m=0;
    for(k=1;k<=r;k++)
   resreq[p][k]=newreq[k];
   printf("Deadlock willn't occur\n");
  }
  else
  {
    for(k=1; k<=r; k++)
    {
     resalloc[p][k]=newreq[k];
     totalloc[k]=newreq[k];
    }
   printf("Deadlock will occur\n");
  }
  return 0;
}
```

LAB – 6 / 2-66

Write a C program to implement the Contiguous allocation technique: - First-Fit

```
#include<stdio.h>
#define max 25
void main(){
   int frag[max],b[max],f[max],i,j,nb,nf,temp;
   static int bf[max],ff[max];
   printf("Enter the number of blocks: ");
   scanf("%d",&nb);
   printf("Enter the number of files: ");
   scanf("%d",&nf);
   printf("Enter the size of the blocks\n");
   for(i=1;i<=nb;i++){
     printf("Block %d: ",i);
     scanf("%d",&b[i]);

   }
   printf("Enter the size of the files\n");
   for(i=1;i<=nf;i++){
     printf("File %d: ",i);
     scanf("%d",&f[i]);
   }  for(i=1;i<=nf;i++){
     for(j=1;j<=nb;j++){
       if(bf[j]!=1){
         temp=b[j]-f[i];
         if(temp>=0){
           ff[i]=j;break;
         }
       }
     }
     frag[i]=temp;
     bf[ff[i]]=1;
   }
   printf("File_no\tFile_size\tBlock_no\tBlock_size\tFragement\n");
   for(i=1;i<=nf;i++)
   printf("%d\t%d\t%d\t%d\t%d\n",i,f[i],ff[i],b[ff[i]],frag[i]);  }
```

LAB – 6 / 3-67

Write a program to Implementation of Contiguous allocation technique: - Best-Fit

```
#include<stdio.h>
#define max 25
void main(){
   int frag[max],b[max],f[max],i,j,nb,nf,temp,lowest=10000;
   static int bf[max],ff[max];
   printf("Memory Management Scheme for contigus memeory allocation - Best Fit\n");
   printf("Enter the number of blocks:");
   scanf("%d",&nb);
   printf("Enter the number of files:");
   scanf("%d",&nf);
   printf("Enter the size of the blocks:-\n");
   for(i=1;i<=nb;i++){
     printf("Block %d:",i);
     scanf("%d",&b[i]);

   }
   printf("Enter the size of the files :-\n");
   for(i=1;i<=nf;i++){
     printf("File %d:",i);
     scanf("%d",&f[i]);

   }
   for(i=1;i<=nf;i++){
     for(j=1;j<=nb;j++){
       if(bf[j]!=1){
         temp=b[j]-f[i];
         if(temp>=0)
         if(lowest>temp){
           ff[i]=j;lowest=temp;

         }

       }

     }
     frag[i]=lowest;
     bf[ff[i]]=1;
     lowest=10000;
   }
   printf("File No\tFile Size \tBlock No\tBlock Size\tFragment");
   for(i=1;i<=nf && ff[i]!=0;i++)
   printf("%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);
}
```

LAB – 6 / 4-68

Write a program to Implementation of Contiguous allocation technique :- Worst-Fit

```
#include<stdio.h>
#include<limits.h>
#define MAX 30
#define foi(i, lb, ub) for(int i = lb; i< ub; i++)
int arr[MAX] = {0};
void main(){
  int n, m, b[MAX], f[MAX], temp, bn[MAX] = {0}, frag[MAX] = {0}, t=INT_MIN;
  printf("Enter the number of blocks: ");
  scanf("%d",&n);
  printf("Enter the number of files: ");
  scanf("%d", &m);
  printf("Enter the size of the blocks\n");
  foi(i, 0 , n){
    printf("Block %d: ", i+1);
    scanf("%d",&b[i]);
  }
  printf("Enter the size of the files\n");
  foi(i, 0, m){
    printf("File %d: ",i+1);
    scanf("%d",&f[i]);
  }

    foi(i, 0 , m){
      temp = -1;
      foi(j, 0 ,n){
        if(arr[j] == 0){
          temp = b[j] - f[i];
          if(temp < 0)
          continue;
          if(temp > t){
            t = temp;
            bn[i] = j+1;
          }
        }
      }

          if(temp >= 0){
            frag[i] = t;
            arr[bn[i]- 1]=1;
          }
          t = INT_MIN;
    }

      printf("File_no\tFile_size\tBlock_no\tBlock_size\tFragement\n");
      foi(i, 0, m){
        if(bn[i] != 0)
        printf("%d\t%d\t%d\t%d\t%d\n", i+1, f[i], bn[i], b[bn[i]-1], frag[i]);
        else
        printf("%d\t%d\t%d\t%d\t%d\n", i+1, f[i], bn[i], 0, frag[i]);
      }
}
```

LAB – 6 / 5-69

Write a program to Implementation of contiguous memory fixed partition technique(MFT)

```
#include<stdio.h>
#define MAX 30
#define foi(i, lb, ub) for(int i=lb; i < ub ; i++)
int len(int n){
  int c = 0;
  while(n){
    n = n/10;
    c ++;
  }
  return c;
}
void main(){
  int l=1,ms,n,p,alm[MAX],m,frag[MAX],f=0,temp,t;
  printf("Enter the memory size:");
  scanf("%d",&ms);
  printf("Enter the no of partitions:");
  scanf("%d",&n);
  printf("Each partn size is:%dEnter the no of processes:", ms/n);
  scanf("%d",&p);
  m=ms/n;
  foi(i, 0 , p){
    printf("Enter the memory req for process%d:",i + 1);
    scanf("%d",&alm[i]);
    frag[i]=m-alm[i];
    if(frag[i] >= 0){
      printf("Process is allocated in partition%d\n", i+1);
      printf("Internal fragmentation for process is:%d\n", frag[i]);
    }
    else{
      printf("Process not allocated in partition%d\n",i+1);
      printf("External fragmentation for partition is:%d", m);
      t=m;
      while(t<alm[i]){
        t=m*l;
        l++;
      }
      frag[i] = t - m;

    }
    f += frag[i];
  }
  printf("Process\tmemory\tallocatedmemory\n");
  foi(i,0,p){
    printf("   ");
    printf("%d\t",i+1);
    temp=5-len(m);
    while(temp--)
    printf(" ");
    printf("%d\t",m);
    temp=5-len(alm[i]);
    while(temp--)
    printf(" ");
    printf("%d\n",alm[i]);
  }
  printf("The tot no of fragmentation is:%d", f);
}
```

LAB – 6 / 6-70

Write a program to Implementation of contiguous memory Variable partition technique (MVT)

```
#include<stdio.h>
#include<conio.h>
int main(){
   int m=0,m1=0,m2=0,p,count=0,i;
   printf("enter the memory capacity:");
   scanf("%d",&m);
   printf("enter the no of processes:");
   scanf("%d",&p);
   for(i=0;i<p;i++){
     printf("enter memory req for process%d:",i+1);
     scanf("%d",&m1);
     count=count+m1;
     if(count==m)
     printf("there is no further memory remaining:\n");
     else if(m1<m){
       printf("the memory allocated for process%d is: %d ",i+1,m);
       m2=m-m1;
       printf("\nremaining memory is: %d\n",m2);
       m=m2;
     }
     else   {
       printf("memory is not allocated for process%d",i+1);
     }
     printf("external fragmentation for this process is:%d\n",m2);

   }
   return 0;

}
```