SJF

#include<stdio.h>

void main()

{

    int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,pos,temp;

    float avg\_wt,avg\_tat;

    printf("Enter number of process:");

    scanf("%d",&n);

    printf("\nEnter Burst Time:\n");

    for(i=0;i<n;i++)

    {

        printf("p%d:",i+1);

        scanf("%d",&bt[i]);

        p[i]=i+1;           //contains process number

    }

    //sorting burst time in ascending order using selection sort

    for(i=0;i<n;i++)

    {

        pos=i;

        for(j=i+1;j<n;j++)

        {

            if(bt[j]<bt[pos])

                pos=j;

        }

        temp=bt[i];

        bt[i]=bt[pos];

        bt[pos]=temp;

        temp=p[i];

        p[i]=p[pos];

        p[pos]=temp;

    }

    wt[0]=0;            //waiting time for first process will be zero

    //calculate waiting time

    for(i=1;i<n;i++)

    {

        wt[i]=0;

        for(j=0;j<i;j++)

            wt[i]+=bt[j];

        total+=wt[i];

    }

    avg\_wt=(float)total/n;      //average waiting time

    total=0;

    printf("\nProcess\t    Burst Time    \tWaiting Time\tTurnaround Time");

    for(i=0;i<n;i++)

    {

        tat[i]=bt[i]+wt[i];     //calculate turnaround time

        total+=tat[i];

        printf("\np%d\t\t  %d\t\t    %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);

    }

    avg\_tat=(float)total/n;     //average turnaround time

    printf("\n\nAverage Waiting Time=%f",avg\_wt);

    printf("\nAverage Turnaround Time=%f\n",avg\_tat);

}

FFCS

|  |
| --- |
| int main()  {      int n,bt[20],wt[20],tat[20],avwt=0,avtat=0,i,j;      printf("Enter total number of processes(maximum 20):");      scanf("%d",&n);        printf("\nEnter Process Burst Time\n");      for(i=0;i<n;i++)      {          printf("P[%d]:",i+1);          scanf("%d",&bt[i]);      }        wt[0]=0;    //waiting time for first process is 0        //calculating waiting time      for(i=1;i<n;i++)      {          wt[i]=0;          for(j=0;j<i;j++)              wt[i]+=bt[j];      }        printf("\nProcess\t\tBurst Time\tWaiting Time\tTurnaround Time");        //calculating turnaround time      for(i=0;i<n;i++)      {          tat[i]=bt[i]+wt[i];          avwt+=wt[i];          avtat+=tat[i];          printf("\nP[%d]\t\t%d\t\t%d\t\t%d",i+1,bt[i],wt[i],tat[i]);      }        avwt/=i;      avtat/=i;      printf("\n\nAverage Waiting Time:%d",avwt);      printf("\nAverage Turnaround Time:%d",avtat);        return 0;  }  Round Robbin  #include<stdio.h>    int main()  {      int count,j,n,time,remain,flag=0,time\_quantum;    int wait\_time=0,turnaround\_time=0,at[10],bt[10],rt[10];    printf("Enter Total Process:\t ");    scanf("%d",&n);    remain=n;    for(count=0;count<n;count++)    {      printf("Enter Arrival Time and Burst Time for Process Process Number %d :",count+1);      scanf("%d",&at[count]);      scanf("%d",&bt[count]);      rt[count]=bt[count];    }    printf("Enter Time Quantum:\t");    scanf("%d",&time\_quantum);    printf("\n\nProcess\t|Turnaround Time|Waiting Time\n\n");    for(time=0,count=0;remain!=0;)    {      if(rt[count]<=time\_quantum && rt[count]>0)      {        time+=rt[count];        rt[count]=0;        flag=1;      }      else if(rt[count]>0)      {        rt[count]-=time\_quantum;        time+=time\_quantum;      }      if(rt[count]==0 && flag==1)      {        remain--;        printf("P[%d]\t|\t%d\t|\t%d\n",count+1,time-at[count],time-at[count]-bt[count]);        wait\_time+=time-at[count]-bt[count];        turnaround\_time+=time-at[count];        flag=0;      }      if(count==n-1)        count=0;      else if(at[count+1]<=time)        count++;      else        count=0;    }    printf("\nAverage Waiting Time= %f\n",wait\_time\*1.0/n);    printf("Avg Turnaround Time = %f",turnaround\_time\*1.0/n);      return 0;  }  Priority Scheduling  #include<stdio.h>    int main()  {      int bt[20],p[20],wt[20],tat[20],pr[20],i,j,n,total=0,pos,temp,avg\_wt,avg\_tat;      printf("Enter Total Number of Process:");      scanf("%d",&n);        printf("\nEnter Burst Time and Priority\n");      for(i=0;i<n;i++)      {          printf("\nP[%d]\n",i+1);          printf("Burst Time:");          scanf("%d",&bt[i]);          printf("Priority:");          scanf("%d",&pr[i]);          p[i]=i+1;           //contains process number      }        //sorting burst time, priority and process number in ascending order using selection sort      for(i=0;i<n;i++)      {          pos=i;          for(j=i+1;j<n;j++)          {              if(pr[j]<pr[pos])                  pos=j;          }            temp=pr[i];          pr[i]=pr[pos];          pr[pos]=temp;            temp=bt[i];          bt[i]=bt[pos];          bt[pos]=temp;            temp=p[i];          p[i]=p[pos];          p[pos]=temp;      }        wt[0]=0;    //waiting time for first process is zero        //calculate waiting time      for(i=1;i<n;i++)      {          wt[i]=0;          for(j=0;j<i;j++)              wt[i]+=bt[j];            total+=wt[i];      }        avg\_wt=total/n;      //average waiting time      total=0;        printf("\nProcess\t    Burst Time    \tWaiting Time\tTurnaround Time");      for(i=0;i<n;i++)      {          tat[i]=bt[i]+wt[i];     //calculate turnaround time          total+=tat[i];          printf("\nP[%d]\t\t  %d\t\t    %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);      }        avg\_tat=total/n;     //average turnaround time      printf("\n\nAverage Waiting Time=%d",avg\_wt);      printf("\nAverage Turnaround Time=%d\n",avg\_tat);        return 0;  }  SRTF  #include <stdio.h>  int main()  {  int a[10],b[10],x[10],i,j,smallest,count=0,time,n;  double avg=0,tt=0,end;  printf("enter the number of Processes:\n");  scanf("%d",&n);  printf("enter arrival time\n");  for(i=0;i<n;i++)  scanf("%d",&a[i]);  printf("enter burst time\n");  for(i=0;i<n;i++)  scanf("%d",&b[i]);  for(i=0;i<n;i++)  x[i]=b[i];  b[9]=9999;    for(time=0;count!=n;time++)  {  smallest=9;  for(i=0;i<n;i++)  {  if(a[i]<=time && b[i]<b[smallest] && b[i]>0 )  smallest=i;  }  b[smallest]--;  if(b[smallest]==0)  {  count++;  end=time+1;  avg=avg+end-a[smallest]-x[smallest];  tt= tt+end-a[smallest];  }  }  printf("\n\nAverage waiting time = %lf\n",avg/n);  printf("Average Turnaround time = %lf",tt/n);  return 0;  }  NON PRIMITIVE PRIORITY SCHEDULING  #include<stdio.h>    int main()  {        int burst\_time[20], process[20], waiting\_time[20], turnaround\_time[20], priority[20];        int i, j, limit, sum = 0, position, temp;        float average\_wait\_time, average\_turnaround\_time;        printf("Enter Total Number of Processes:\t");        scanf("%d", &limit);        printf("\nEnter Burst Time and Priority For %d Processes\n", limit);        for(i = 0; i < limit; i++)        {              printf("\nProcess[%d]\n", i + 1);              printf("Process Burst Time:\t");              scanf("%d", &burst\_time[i]);              printf("Process Priority:\t");              scanf("%d", &priority[i]);              process[i] = i + 1;        }        for(i = 0; i < limit; i++)        {              position = i;              for(j = i + 1; j < limit; j++)              {                    if(priority[j] < priority[position])                    {                          position = j;                    }              }              temp = priority[i];              priority[i] = priority[position];              priority[position] = temp;              temp = burst\_time[i];              burst\_time[i] = burst\_time[position];              burst\_time[position] = temp;              temp = process[i];              process[i] = process[position];              process[position] = temp;        }        waiting\_time[0] = 0;        for(i = 1; i < limit; i++)        {              waiting\_time[i] = 0;              for(j = 0; j < i; j++)              {                    waiting\_time[i] = waiting\_time[i] + burst\_time[j];              }              sum = sum + waiting\_time[i];        }        average\_wait\_time = sum / limit;        sum = 0;        printf("\nProcess ID\t\tBurst Time\t Waiting Time\t Turnaround Time\n");        for(i = 0; i < limit; i++)        {              turnaround\_time[i] = burst\_time[i] + waiting\_time[i];              sum = sum + turnaround\_time[i];              printf("\nProcess[%d]\t\t%d\t\t %d\t\t %d\n", process[i], burst\_time[i], waiting\_time[i], turnaround\_time[i]);        }        average\_turnaround\_time = sum / limit;        printf("\nAverage Waiting Time:\t%f", average\_wait\_time);        printf("\nAverage Turnaround Time:\t%f\n", average\_turnaround\_time);        return 0;  }  CONSUMER AND PRODUCER  #include<stdio.h>  #include<stdlib.h>    int mutex=1,full=0,empty=3,x=0;    int main()  {      int n;      void producer();      void consumer();      int wait(int);      int signal(int);      printf("\n1.Producer\n2.Consumer\n3.Exit");      while(1)      {          printf("\nEnter your choice:");          scanf("%d",&n);          switch(n)          {              case 1:    if((mutex==1)&&(empty!=0))                          producer();                      else                          printf("Buffer is full!!");                      break;              case 2:    if((mutex==1)&&(full!=0))                          consumer();                      else                          printf("Buffer is empty!!");                      break;              case 3:                      exit(0);                      break;          }      }        return 0;  }    int wait(int s)  {      return (--s);  }    int signal(int s)  {      return(++s);  }    void producer()  {      mutex=wait(mutex);      full=signal(full);      empty=wait(empty);      x++;      printf("\nProducer produces the item %d",x);      mutex=signal(mutex);  }    void consumer()  {      mutex=wait(mutex);      full=wait(full);      empty=signal(empty);      printf("\nConsumer consumes item %d",x);      x--;      mutex=signal(mutex);  }  DINNING PHILOSOPHER  #include<stdio.h>    #define n 4    int compltedPhilo = 0,i;    struct fork{      int taken;  }ForkAvil[n];    struct philosp{      int left;      int right;  }Philostatus[n];    void goForDinner(int philID){ //same like threads concept here cases implemented      if(Philostatus[philID].left==10 && Philostatus[philID].right==10)          printf("Philosopher %d completed his dinner\n",philID+1);      //if already completed dinner      else if(Philostatus[philID].left==1 && Philostatus[philID].right==1){              //if just taken two forks              printf("Philosopher %d completed his dinner\n",philID+1);                Philostatus[philID].left = Philostatus[philID].right = 10; //remembering that he completed dinner by assigning value 10              int otherFork = philID-1;                if(otherFork== -1)                  otherFork=(n-1);                ForkAvil[philID].taken = ForkAvil[otherFork].taken = 0; //releasing forks              printf("Philosopher %d released fork %d and fork %d\n",philID+1,philID+1,otherFork+1);              compltedPhilo++;          }          else if(Philostatus[philID].left==1 && Philostatus[philID].right==0){ //left already taken, trying for right fork                  if(philID==(n-1)){                      if(ForkAvil[philID].taken==0){ //KEY POINT OF THIS PROBLEM, THAT LAST PHILOSOPHER TRYING IN reverse DIRECTION                          ForkAvil[philID].taken = Philostatus[philID].right = 1;                          printf("Fork %d taken by philosopher %d\n",philID+1,philID+1);                      }else{                          printf("Philosopher %d is waiting for fork %d\n",philID+1,philID+1);                      }                  }else{ //except last philosopher case                      int dupphilID = philID;                      philID-=1;                        if(philID== -1)                          philID=(n-1);                        if(ForkAvil[philID].taken == 0){                          ForkAvil[philID].taken = Philostatus[dupphilID].right = 1;                          printf("Fork %d taken by Philosopher %d\n",philID+1,dupphilID+1);                      }else{                          printf("Philosopher %d is waiting for Fork %d\n",dupphilID+1,philID+1);                      }                  }              }              else if(Philostatus[philID].left==0){ //nothing taken yet                      if(philID==(n-1)){                          if(ForkAvil[philID-1].taken==0){ //KEY POINT OF THIS PROBLEM, THAT LAST PHILOSOPHER TRYING IN reverse DIRECTION                              ForkAvil[philID-1].taken = Philostatus[philID].left = 1;                              printf("Fork %d taken by philosopher %d\n",philID,philID+1);                          }else{                              printf("Philosopher %d is waiting for fork %d\n",philID+1,philID);                          }                      }else{ //except last philosopher case                          if(ForkAvil[philID].taken == 0){                              ForkAvil[philID].taken = Philostatus[philID].left = 1;                              printf("Fork %d taken by Philosopher %d\n",philID+1,philID+1);                          }else{                              printf("Philosopher %d is waiting for Fork %d\n",philID+1,philID+1);                          }                      }          }else{}  }    int main(){      for(i=0;i<n;i++)          ForkAvil[i].taken=Philostatus[i].left=Philostatus[i].right=0;        while(compltedPhilo<n){          /\* Observe here carefully, while loop will run until all philosophers complete dinner          Actually problem of deadlock occur only thy try to take at same time          This for loop will say that they are trying at same time. And remaining status will print by go for dinner function          \*/          for(i=0;i<n;i++)              goForDinner(i);          printf("\nTill now num of philosophers completed dinner are %d\n\n",compltedPhilo);      }        return 0;  }  READERS WRITERS PROBLEM  #include<stdio.h> #include<pthread.h> #include<semaphore.h>  sem\_t readCountAccess; sem\_t databaseAccess; int readCount=0;  void \*Reader(void \*arg); void \*Writer(void \*arg);  int main() { int i=0,NumberofReaderThread=0,NumberofWriterThread; sem\_init(&readCountAccess,0,1); sem\_init(&databaseAccess,0,1);  pthread\_t Readers\_thr[100],Writer\_thr[100]; printf(“\nEnter number of Readers thread(MAX 10)”); scanf(“%d”,&NumberofReaderThread); printf(“\nEnter number of Writers thread(MAX 10)”); scanf(“%d”,&NumberofWriterThread);  for(i=0;i<numberofreaderthread;i++) { pthread\_create(&Readers\_thr[i],NULL,Reader,(void \*)i); } for(i=0;i<numberofwriterthread;i++) { pthread\_create(&Writer\_thr[i],NULL,Writer,(void \*)i); } for(i=0;i<NumberofWriterThread;i++) { pthread\_join(Writer\_thr[i],NULL); }  for(i=0;i<NumberofReaderThread;i++) { pthread\_join(Readers\_thr[i],NULL); } sem\_destroy(&databaseAccess); sem\_destroy(&readCountAccess); return 0; }  void \* Writer(void \*arg) {  sleep(1); int temp=(int)arg; printf(“\nWriter %d is trying to enter into database for modifying the data”,temp); sem\_wait(&databaseAccess); printf(“\nWriter %d is writting into the database”,temp); printf(“\nWriter %d is leaving the database”); sem\_post(&databaseAccess); }  void \*Reader(void \*arg) { sleep(1); int temp=(int)arg; printf(“\nReader %d is trying to enter into the Database for reading the data”,temp); sem\_wait(&readCountAccess); readCount++; if(readCount==1) { sem\_wait(&databaseAccess); printf(“\nReader %d is reading the database”,temp); } sem\_post(&readCountAccess); sem\_wait(&readCountAccess); readCount–; if(readCount==0) { printf(“\nReader %d is leaving the database”,temp); sem\_post(&databaseAccess); } sem\_post(&readCountAccess); }    OUTPUT    Enter number of Readers thread(MAX 10)2  Enter number of Writers thread(MAX 10)1  Reader 0 is trying to enter into the Database for reading the data Reader 0 is reading the database Reader 0 is leaving the database Reader 1 is trying to enter into the Database for reading the data Reader 1 is reading the database Reader 1 is leaving the database Writer 0 is trying to enter into database for modifying the data Writer 0 is writing into the database Writer 0 is leaving the database  BEST FIT  #include<stdio.h>    void main()  {      int fragment[20],b[20],p[20],i,j,nb,np,temp,lowest=9999;      static int barray[20],parray[20];        printf("\n\t\t\tMemory Management Scheme - Best Fit");      printf("\nEnter the number of blocks:");      scanf("%d",&nb);      printf("Enter the number of processes:");      scanf("%d",&np);        printf("\nEnter the size of the blocks:-\n");      for(i=1;i<=nb;i++)      {          printf("Block no.%d:",i);          scanf("%d",&b[i]);      }        printf("\nEnter the size of the processes :-\n");      for(i=1;i<=np;i++)      {          printf("Process no.%d:",i);          scanf("%d",&p[i]);      }        for(i=1;i<=np;i++)      {          for(j=1;j<=nb;j++)          {              if(barray[j]!=1)              {                  temp=b[j]-p[i];                  if(temp>=0)                      if(lowest>temp)                      {                          parray[i]=j;                          lowest=temp;                      }              }          }            fragment[i]=lowest;          barray[parray[i]]=1;          lowest=10000;      }        printf("\nProcess\_no\tProcess\_size\tBlock\_no\tBlock\_size\tFragment");      for(i=1;i<=np && parray[i]!=0;i++)          printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,p[i],parray[i],b[parray[i]],fragment[i]);  }  FIRST FIT    #include<stdio.h>  void main()  {      int bsize[10], psize[10], bno, pno, flags[10], allocation[10], i, j;        for(i = 0; i < 10; i++)      {          flags[i] = 0;          allocation[i] = -1;      }        printf("Enter no. of blocks: ");      scanf("%d", &bno);        printf("\nEnter size of each block: ");      for(i = 0; i < bno; i++)          scanf("%d", &bsize[i]);        printf("\nEnter no. of processes: ");      scanf("%d", &pno);        printf("\nEnter size of each process: ");      for(i = 0; i < pno; i++)          scanf("%d", &psize[i]);      for(i = 0; i < pno; i++)         //allocation as per first fit          for(j = 0; j < bno; j++)              if(flags[j] == 0 && bsize[j] >= psize[i])              {                  allocation[j] = i;                  flags[j] = 1;                  break;              }        //display allocation details      printf("\nBlock no.\tsize\t\tprocess no.\t\tsize");      for(i = 0; i < bno; i++)      {          printf("\n%d\t\t%d\t\t", i+1, bsize[i]);          if(flags[i] == 1)              printf("%d\t\t\t%d",allocation[i]+1,psize[allocation[i]]);          else              printf("Not allocated");      }  }  WORST FIT  #include<stdio.h>    int main()  {        int fragments[10], blocks[10], files[10];        int m, n, number\_of\_blocks, number\_of\_files, temp, top = 0;        static int block\_arr[10], file\_arr[10];        printf("\nEnter the Total Number of Blocks:\t");        scanf("%d",&number\_of\_blocks);        printf("Enter the Total Number of Files:\t");        scanf("%d",&number\_of\_files);        printf("\nEnter the Size of the Blocks:\n");        for(m = 0; m < number\_of\_blocks; m++)        {              printf("Block No.[%d]:\t", m + 1);              scanf("%d", &blocks[m]);        }        printf("Enter the Size of the Files:\n");        for(m = 0; m < number\_of\_files; m++)        {              printf("File No.[%d]:\t", m + 1);              scanf("%d", &files[m]);        }        for(m = 0; m < number\_of\_files; m++)        {              for(n = 0; n < number\_of\_blocks; n++)              {                    if(block\_arr[n] != 1)                    {                          temp = blocks[n] - files[m];                          if(temp >= 0)                          {                                if(top < temp)                                {                                      file\_arr[m] = n;                                      top = temp;                                }                          }                    }                    fragments[m] = top;                    block\_arr[file\_arr[m]] = 1;                    top = 0;              }        }        printf("\nFile Number\tFile Size\tBlock Number\tBlock Size\tFragment");        for(m = 0; m < number\_of\_files; m++)        {              printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d", m, files[m], file\_arr[m], blocks[file\_arr[m]], fragments[m]);        }        printf("\n");        return 0;  }  LRU  #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], 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(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;  }  OPTIMAL PAGE  #include<stdio.h>    int main()  {      int no\_of\_frames, no\_of\_pages, frames[10], pages[30], temp[10], flag1, flag2, flag3, i, j, k, pos, max, faults = 0;      printf("Enter number of frames: ");      scanf("%d", &no\_of\_frames);        printf("Enter number of pages: ");      scanf("%d", &no\_of\_pages);        printf("Enter page 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]){                     flag1 = flag2 = 1;                     break;                 }          }            if(flag1 == 0){              for(j = 0; j < no\_of\_frames; ++j){                  if(frames[j] == -1){                      faults++;                      frames[j] = pages[i];                      flag2 = 1;                      break;                  }              }          }            if(flag2 == 0){              flag3 =0;                for(j = 0; j < no\_of\_frames; ++j){                  temp[j] = -1;                    for(k = i + 1; k < no\_of\_pages; ++k){                      if(frames[j] == pages[k]){                          temp[j] = k;                          break;                      }                  }              }                for(j = 0; j < no\_of\_frames; ++j){                  if(temp[j] == -1){                      pos = j;                      flag3 = 1;                      break;                  }              }                if(flag3 ==0){                  max = temp[0];                  pos = 0;                    for(j = 1; j < no\_of\_frames; ++j){                      if(temp[j] > max){                          max = temp[j];                          pos = j;                      }                  }              }                frames[pos] = pages[i];              faults++;          }            printf("\n");            for(j = 0; j < no\_of\_frames; ++j){              printf("%d\t", frames[j]);          }      }        printf("\n\nTotal Page Faults = %d", faults);        return 0;  }  FIFO  #include<stdio.h>  int main()  {  int i,j,n,a[50],frame[10],no,k,avail,count=0;              printf("\n ENTER THE NUMBER OF PAGES:\n");  scanf("%d",&n);              printf("\n ENTER THE PAGE NUMBER :\n");              for(i=1;i<=n;i++)              scanf("%d",&a[i]);              printf("\n ENTER THE NUMBER OF FRAMES :");              scanf("%d",&no);  for(i=0;i<no;i++)              frame[i]= -1;                          j=0;                          printf("\tref string\t page 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 Fault Is %d",count);                          return 0;  } |