OPERATING SYSTEMS LAB MANUAL

FOR

FOR B.TECH II YEAR II SEMESTER

IT



2019-2020

CVR COLLEGE OF ENGINEERING

(AUTONOMOUS)

Vastunagar, Mangalpalli(V), Ibrahimpatan (M),

R.R. District. Pin : 501510

Email : [info@cvr.ac.in](mailto:info@cvr.ac.in)

Web: <http://cvr.ac.in>

**INDEX**

|  |  |  |
| --- | --- | --- |
| **S.No.** | **PROGRAM/EXPERIMENT** | **Page No.** |
| **1** | **Write a program to simulate First Come First Serve CPU scheduling algorithm?** | **2** |
| **2** | **Write a program to simulate shortest job first CPU Scheduling algorithm ( with and with out preemption)** | **3** |
| **3** | **Write a program to simulate Round Robin cpu scheduling algorithm?** | **7** |
| **4** | **Write a program to simulate Priority scheduling algorithm?** | **12** |
| **5** | **simulate Bankers algorithm for Dead loack avoidance algorithm** | **15** |
| **6, 7, 8** | **Simulation of FIFO, LRU, OPTIMAL Page Replacement Algorithm?** | **17** |
| **9** | **Simulate Sequential file allocation strategy?** | **26** |
| **10** | **Simulate linked file allocation strategy?** | **28** |
| **11** | **Simulate Indexed file allocation strategy** | **29** |
| **12** | **Simulate single level directory structure** | **31** |
| **13** | **Simulate Two level Directory Structure** | **34** |
| **14** | **Simulate MFT AND MVT** | **37** |
|  |  |  |
|  |  |  |

1. Write a program to simulate First Come First Serve CPU scheduling algorithm?

Program:

#include<stdio.h>

 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 Timen");

    for(i=0;i<n;i++)

    {

        printf("P[%d]:",i+1);

        scanf("%d",&bt[i]);

    }

    wt[0]=0;

    for(i=1;i<n;i++)

    {

        wt[i]=0;

        for(j=0;j<i;j++)

            wt[i]+=bt[j];

    }

    printf("nProcessttBurst TimetWaiting TimetTurnaround Time");

    for(i=0;i<n;i++)

    {

        tat[i]=bt[i]+wt[i];

        avwt+=wt[i];

        avtat+=tat[i];

        printf("nP[%d]tt%dtt%dtt%d",i+1,bt[i],wt[i],tat[i]);

    }

    avwt/=i;

    avtat/=i;

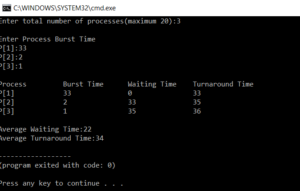
    printf("nnAverage Waiting Time:%d",avwt);

    printf("nAverage Turnaround Time:%d",avtat);

    return 0;

}

Output:



1. Write a program to simulate shortest job first CPU Scheduling algorithm ( with and with out preemption)

Program:

**There are two types of SJF**

* Pre-emptive SJF
* Non-Preemptive SJF

These algorithms schedule processes in the order in which the shortest job is done first. It has a minimum average waiting time.

There are 3 factors to consider while solving SJF, they are

1. BURST Time
2. Average waiting time
3. Average turnaround time

**Non-Preemptive Shortest Job First**

Here is an example

|  |  |  |  |
| --- | --- | --- | --- |
| **Processes Id** | **Burst Time** | **Waiting Time** | **Turn Around Time** |
| **4** | 3 | 0 | 3 |
| **1** | 6 | 3 | 9 |
| **3** | 7 | 9 | 16 |
| **2** | 8 | 16 | 25 |

Average waiting time = **7**

Average turnaround time = **13**

T.A.T= waiting time + burst time

CODE FOR NON PRE-EMPTIVE SCHEDULING

#include<stdio.h>

 int 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;

    }

   //sorting of burst times

    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;

    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;

    total=0;

    printf("nProcesst    Burst Time    tWaiting TimetTurnaround Time");

    for(i=0;i<n;i++)

    {

        tat[i]=bt[i]+wt[i];

        total+=tat[i];

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

    }

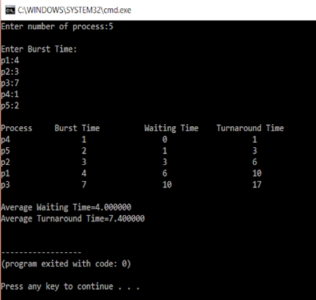
    avg\_tat=(float)total/n;

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

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

}

OUTPUT:



CODE FOR PREEMTIVE SCHEDULING

#include <stdio.h>

int main()

{

      int arrival\_time[10], burst\_time[10], temp[10];

      int i, smallest, count = 0, time, limit;

      double wait\_time = 0, turnaround\_time = 0, end;

      float average\_waiting\_time, average\_turnaround\_time;

      printf("nEnter the Total Number of Processes:t");

      scanf("%d", &limit);

      printf("nEnter Details of %d Processesn", limit);

      for(i = 0; i < limit; i++)

      {

            printf("nEnter Arrival Time:t");

            scanf("%d", &arrival\_time[i]);

            printf("Enter Burst Time:t");

            scanf("%d", &burst\_time[i]);

            temp[i] = burst\_time[i];

      }

      burst\_time[9] = 9999;

      for(time = 0; count != limit; time++)

      {

            smallest = 9;

            for(i = 0; i < limit; i++)

            {

                  if(arrival\_time[i] <= time && burst\_time[i] < burst\_time[smallest] && burst\_time[i] > 0)

                  {

                        smallest = i;

                  }

            }

            burst\_time[smallest]--;

            if(burst\_time[smallest] == 0)

            {

                  count++;

                  end = time + 1;

                  wait\_time = wait\_time + end - arrival\_time[smallest] - temp[smallest];

                  turnaround\_time = turnaround\_time + end - arrival\_time[smallest];

            }

      }

      average\_waiting\_time = wait\_time / limit;

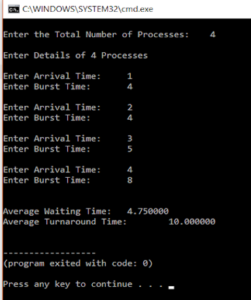
      average\_turnaround\_time = turnaround\_time / limit;

      printf("nnAverage Waiting Time:t%lfn", average\_waiting\_time);

      printf("Average Turnaround Time:t%lfn", average\_turnaround\_time);

      return 0;

}



Program 3: Write a program to simulate Round Robin cpu scheduling algorithm?

Solution:

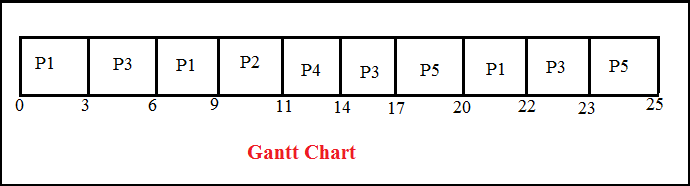
Suppose we have five processes P1, P2, P3, P4 and P5. The arrival and burst time of each process are mentioned in the following table, as shown below. The time quantum is three units.

|  |  |  |
| --- | --- | --- |
| **Process** | **Arrival Time (AT)** | **Burst Time (BT)** |
| P1 | 0 | 8 |
| P2 | 5 | 2 |
| P3 | 1 | 7 |
| P4 | 6 | 3 |
| P5 | 8 | 5 |

Now we have to create the **ready queue** and the **Gantt chart** for Round Robin CPU Scheduler.

Ready queue: P1, P3, P1, P2, P4, P3, P5, P1, P3, P5

Here is the Gantt chart:



#include<stdio.h>

int main()

{

      int i, limit, total = 0, x, counter = 0, time\_quantum;

      int wait\_time = 0, turnaround\_time = 0, arrival\_time[10], burst\_time[10], temp[10];

      float average\_wait\_time, average\_turnaround\_time;

      printf("nEnter Total Number of Processes:t");

      scanf("%d", &limit);

      x = limit;

      for(i = 0; i < limit; i++)

      {

            printf("nEnter Details of Process[%d]n", i + 1);

            printf("Arrival Time:t");

            scanf("%d", &arrival\_time[i]);

            printf("Burst Time:t");

            scanf("%d", &burst\_time[i]);

            temp[i] = burst\_time[i];

      }

      printf("nEnter Time Quantum:t");

      scanf("%d", &time\_quantum);

      printf("nProcess IDttBurst Timet Turnaround Timet Waiting Timen");

      for(total = 0, i = 0; x != 0;)

      {

            if(temp[i] <= time\_quantum && temp[i] > 0)

            {

                  total = total + temp[i];

                  temp[i] = 0;

                  counter = 1;

            }

            else if(temp[i] > 0)

            {

                  temp[i] = temp[i] - time\_quantum;

                  total = total + time\_quantum;

            }

            if(temp[i] == 0 && counter == 1)

            {

                  x--;

                  printf("nProcess[%d]tt%dtt %dttt %d", i + 1, burst\_time[i], total - arrival\_time[i], total - arrival\_time[i] - burst\_time[i]);

                  wait\_time = wait\_time + total - arrival\_time[i] - burst\_time[i];

                  turnaround\_time = turnaround\_time + total - arrival\_time[i];

                  counter = 0;

            }

            if(i == limit - 1)

            {

                  i = 0;

            }

            else if(arrival\_time[i + 1] <= total)

            {

                  i++;

            }

            else

            {

                  i = 0;

            }

      }

      average\_wait\_time = wait\_time \* 1.0 / limit;

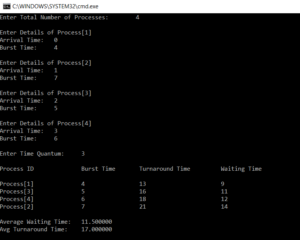
      average\_turnaround\_time = turnaround\_time \* 1.0 / limit;

      printf("nnAverage Waiting Time:t%f", average\_wait\_time);

      printf("nAvg Turnaround Time:t%fn", average\_turnaround\_time);

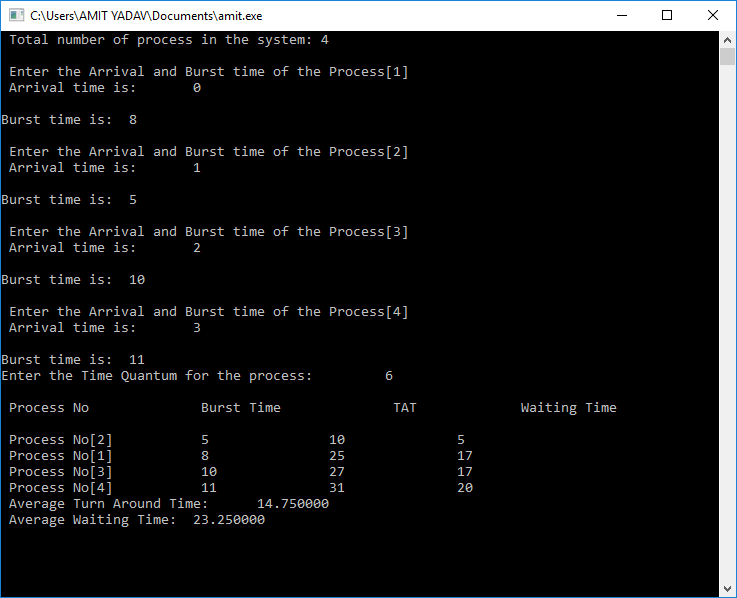
      return 0;

}



1. #include<stdio.h>
2. #include<conio.h>
4. **void** main()
5. {
6. // initlialize the variable name
7. **int** i, NOP, sum=0,count=0, y, quant, wt=0, tat=0, at[10], bt[10], temp[10];
8. **float** avg\_wt, avg\_tat;
9. printf(" Total number of process in the system: ");
10. scanf("%d", &NOP);
11. y = NOP; // Assign the number of process to variable y
13. // Use for loop to enter the details of the process like Arrival time and the Burst Time
14. **for**(i=0; i<NOP; i++)
15. {
16. printf("\n Enter the Arrival and Burst time of the Process[%d]\n", i+1);
17. printf(" Arrival time is: \t");  // Accept arrival time
18. scanf("%d", &at[i]);
19. printf(" \nBurst time is: \t"); // Accept the Burst time
20. scanf("%d", &bt[i]);
21. temp[i] = bt[i]; // store the burst time in temp array
22. }
23. // Accept the Time qunat
24. printf("Enter the Time Quantum for the process: \t");
25. scanf("%d", &quant);
26. // Display the process No, burst time, Turn Around Time and the waiting time
27. printf("\n Process No \t\t Burst Time \t\t TAT \t\t Waiting Time ");
28. **for**(sum=0, i = 0; y!=0; )
29. {
30. **if**(temp[i] <= quant && temp[i] > 0) // define the conditions
31. {
32. sum = sum + temp[i];
33. temp[i] = 0;
34. count=1;
35. }
36. **else** **if**(temp[i] > 0)
37. {
38. temp[i] = temp[i] - quant;
39. sum = sum + quant;
40. }
41. **if**(temp[i]==0 && count==1)
42. {
43. y--; //decrement the process no.
44. printf("\nProcess No[%d] \t\t %d\t\t\t\t %d\t\t\t %d", i+1, bt[i], sum-at[i], sum-at[i]-bt[i]);
45. wt = wt+sum-at[i]-bt[i];
46. tat = tat+sum-at[i];
47. count =0;
48. }
49. **if**(i==NOP-1)
50. {
51. i=0;
52. }
53. **else** **if**(at[i+1]<=sum)
54. {
55. i++;
56. }
57. **else**
58. {
59. i=0;
60. }
61. }
62. // represents the average waiting time and Turn Around time
63. avg\_wt = wt \* 1.0/NOP;
64. avg\_tat = tat \* 1.0/NOP;
65. printf("\n Average Turn Around Time: \t%f", avg\_wt);
66. printf("\n Average Waiting Time: \t%f", avg\_tat);
67. }

Output:



Program 4 : Write a program to simulate Priority scheduling algorithm?

Solution:

#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

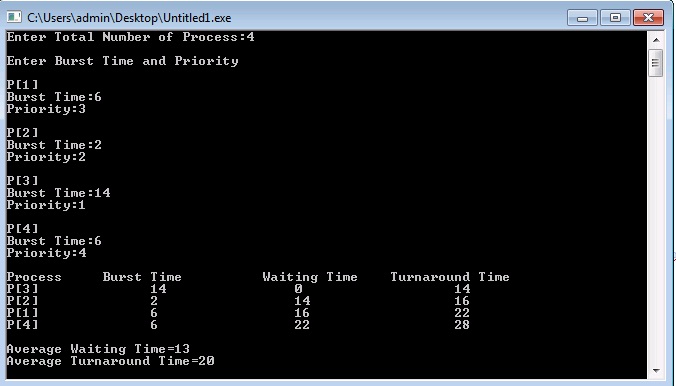
    printf("\n\nAverage Waiting Time=%d",avg\_wt);

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

return 0;

}

Output:



Program 5: simulate Bankers algorithm for Dead loack avoidance algorithm?

Solution:

#include<stdio.h>

#include<conio.h>

void 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];

    //clrscr();

    printf("Enter the no of processes:");

    scanf("%d",&n);

    printf("Enter the no ofresource 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:");

    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]=0;

                    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");

    }

    else

    {

        for(k=1; k<=r; k++)

        {

            resalloc[p][k]=newreq[k];

            totalloc[k]=newreq[k];

        }

        printf("Deadlock will occur");

    }

}

Output:

Enter the no of processes:4

Enter the no ofresource classes:3

Enter the total existed resource in each class:3 2 2

Enter the allocated resources:1 0 0 5 1 1 2 1 1 0 0 2

Enter the process making the new request:2

Enter the requested resource:1 1 2

Enter the process which are n blocked or running:process 2:

1 2

process 4:

1 0

process 5:

1 0

Deadlock will occur

Program 6: Simulation of FIFO, LRU, OPTIMAL Page Replacement Algorithm?

Solution:

#include<stdio.h>

int n,nf;

int in[100];

int p[50];

int hit=0;

int i,j,k;

int pgfaultcnt=0;

void getData()

{

    printf("\nEnter length of page reference sequence:");

    scanf("%d",&n);

    printf("\nEnter the page reference sequence:");

    for(i=0; i<n; i++)

        scanf("%d",&in[i]);

    printf("\nEnter no of frames:");

    scanf("%d",&nf);

}

void initialize()

{

    pgfaultcnt=0;

    for(i=0; i<nf; i++)

        p[i]=9999;

}

int isHit(int data)

{

    hit=0;

    for(j=0; j<nf; j++)

    {

        if(p[j]==data)

        {

            hit=1;

            break;

        }

    }

    return hit;

}

int getHitIndex(int data)

{

    int hitind;

    for(k=0; k<nf; k++)

    {

        if(p[k]==data)

        {

            hitind=k;

            break;

        }

    }

    return hitind;

}

void dispPages()

{

    for (k=0; k<nf; k++)

    {

        if(p[k]!=9999)

            printf(" %d",p[k]);

    }

}

void dispPgFaultCnt()

{

    printf("\nTotal no of page faults:%d",pgfaultcnt);

}

void fifo()

{

    initialize();

    for(i=0; i<n; i++)

    {

        printf("\nFor %d :",in[i]);

        if(isHit(in[i])==0)

        {

            for(k=0; k<nf-1; k++)

                p[k]=p[k+1];

            p[k]=in[i];

            pgfaultcnt++;

            dispPages();

        }

        else

            printf("No page fault");

    }

    dispPgFaultCnt();

}

void optimal()

{

    initialize();

    int near[50];

    for(i=0; i<n; i++)

    {

        printf("\nFor %d :",in[i]);

        if(isHit(in[i])==0)

        {

            for(j=0; j<nf; j++)

            {

                int pg=p[j];

                int found=0;

                for(k=i; k<n; k++)

                {

                    if(pg==in[k])

                    {

                        near[j]=k;

                        found=1;

                        break;

                    }

                    else

                        found=0;

                }

                if(!found)

                    near[j]=9999;

            }

            int max=-9999;

            int repindex;

            for(j=0; j<nf; j++)

            {

                if(near[j]>max)

                {

                    max=near[j];

                    repindex=j;

                }

            }

            p[repindex]=in[i];

            pgfaultcnt++;

            dispPages();

        }

        else

            printf("No page fault");

    }

    dispPgFaultCnt();

}

void lru()

{

    initialize();

    int least[50];

    for(i=0; i<n; i++)

    {

        printf("\nFor %d :",in[i]);

        if(isHit(in[i])==0)

        {

            for(j=0; j<nf; j++)

            {

                int pg=p[j];

                int found=0;

                for(k=i-1; k>=0; k--)

                {

                    if(pg==in[k])

                    {

                        least[j]=k;

                        found=1;

                        break;

                    }

                    else

                        found=0;

                }

                if(!found)

                    least[j]=-9999;

            }

            int min=9999;

            int repindex;

            for(j=0; j<nf; j++)

            {

                if(least[j]<min)

                {

                    min=least[j];

                    repindex=j;

                }

            }

            p[repindex]=in[i];

            pgfaultcnt++;

            dispPages();

        }

        else

            printf("No page fault!");

    }

    dispPgFaultCnt();

}

void lfu()

{

    int usedcnt[100];

    int least,repin,sofarcnt=0,bn;

    initialize();

    for(i=0; i<nf; i++)

        usedcnt[i]=0;

    for(i=0; i<n; i++)

    {

        printf("\n For %d :",in[i]);

        if(isHit(in[i]))

        {

            int hitind=getHitIndex(in[i]);

            usedcnt[hitind]++;

            printf("No page fault!");

        }

        else

        {

            pgfaultcnt++;

            if(bn<nf)

            {

                p[bn]=in[i];

                usedcnt[bn]=usedcnt[bn]+1;

                bn++;

            }

            else

            {

                least=9999;

                for(k=0; k<nf; k++)

                    if(usedcnt[k]<least)

                    {

                        least=usedcnt[k];

                        repin=k;

                    }

                p[repin]=in[i];

                sofarcnt=0;

                for(k=0; k<=i; k++)

                    if(in[i]==in[k])

                        sofarcnt=sofarcnt+1;

                usedcnt[repin]=sofarcnt;

            }

            dispPages();

        }

    }

    dispPgFaultCnt();

}

void secondchance()

{

    int usedbit[50];

    int victimptr=0;

    initialize();

    for(i=0; i<nf; i++)

        usedbit[i]=0;

    for(i=0; i<n; i++)

    {

        printf("\nFor %d:",in[i]);

        if(isHit(in[i]))

        {

            printf("No page fault!");

            int hitindex=getHitIndex(in[i]);

            if(usedbit[hitindex]==0)

                usedbit[hitindex]=1;

        }

        else

        {

            pgfaultcnt++;

            if(usedbit[victimptr]==1)

            {

                do

                {

                    usedbit[victimptr]=0;

                    victimptr++;

                    if(victimptr==nf)

                        victimptr=0;

                }

                while(usedbit[victimptr]!=0);

            }

            if(usedbit[victimptr]==0)

            {

                p[victimptr]=in[i];

                usedbit[victimptr]=1;

                victimptr++;

            }

            dispPages();

        }

        if(victimptr==nf)

            victimptr=0;

    }

    dispPgFaultCnt();

}

int main()

{

    int choice;

    while(1)

    {

        printf("\nPage Replacement Algorithms\n1.Enter data\n2.FIFO\n3.Optimal\n4.LRU\n5.LFU\n6.Second Chance\n7.Exit\nEnter your choice:");

        scanf("%d",&choice);

        switch(choice)

        {

        case 1:

            getData();

            break;

        case 2:

            fifo();

            break;

        case 3:

            optimal();

            break;

        case 4:

            lru();

            break;

        case 5:

            lfu();

            break;

        case 6:

            secondchance();

            break;

        default:

            return 0;

            break;

        }

    }

}

OUTPUT:

Page Replacement Algorithms

1.Enter data

2.FIFO

3.Optimal

4.LRU

5.LFU

6.Second Chance

7.Exit

Enter your choice:1

Enter length of page reference sequence:8

Enter the page reference sequence:2

3

4

2

3

5

6

2

Enter no of frames:3

Page Replacement Algorithms

1.Enter data

2.FIFO

3.Optimal

4.LRU

5.LFU

6.Second Chance

7.Exit

Enter your choice:2

For 2 : 2

For 3 : 2 3

For 4 : 2 3 4

For 2 :No page fault

For 3 :No page fault

For 5 : 3 4 5

For 6 : 4 5 6

For 2 : 5 6 2

Total no of page faults:6

Page Replacement Algorithms

1.Enter data

2.FIFO

3.Optimal

4.LRU

5.LFU

6.Second Chance

7.Exit

Enter your choice:3

For 2 : 2

For 3 : 2 3

For 4 : 2 3 4

For 2 :No page fault

For 3 :No page fault

For 5 : 2 5 4

For 6 : 2 6 4

For 2 :No page fault

Total no of page faults:5

Page Replacement Algorithms

1.Enter data

2.FIFO

3.Optimal

4.LRU

5.LFU

6.Second Chance

7.Exit

Enter your choice:4

For 2 : 2

For 3 : 2 3

For 4 : 2 3 4

For 2 :No page fault!

For 3 :No page fault!

For 5 : 2 3 5

For 6 : 6 3 5

For 2 : 6 2 5

Total no of page faults:6

Page Replacement Algorithms

1.Enter data

2.FIFO

3.Optimal

4.LRU

5.LFU

6.Second Chance

7.Exit

Enter your choice:5

For 2 : 2

For 3 : 2 3

For 4 : 2 3 4

For 2 :No page fault!

For 3 :No page fault!

For 5 : 2 3 5

For 6 : 2 3 6

For 2 :No page fault!

Total no of page faults:5

Page Replacement Algorithms

1.Enter data

2.FIFO

3.Optimal

4.LRU

5.LFU

6.Second Chance

7.Exit

Enter your choice:7

Program 9: Simulate Sequential file allocation strategy?

Solution:

#include <stdio.h>

#include <conio.h>

#include <stdlib.h>

void recurse(int files[]){

int flag = 0, startBlock, len, j, k, ch;

printf("Enter the starting block and the length of the files: ");

scanf("%d%d", &startBlock, &len);

for (j=startBlock; j<(startBlock+len); j++){

if (files[j] == 0)

flag++;

}

if(len == flag){

for (int k=startBlock; k<(startBlock+len); k++){

if (files[k] == 0){

files[k] = 1;

printf("%d\t%d\n", k, files[k]);

}

}

if (k != (startBlock+len-1))

printf("The file is allocated to the disk\n");

}

else

printf("The file is not allocated to the disk\n");

printf("Do you want to enter more files?\n");

printf("Press 1 for YES, 0 for NO: ");

scanf("%d", &ch);

if (ch == 1)

recurse(files);

else

exit(0);

return;

}

int main()

{

int files[50];

for(int i=0;i<50;i++)

files[i]=0;

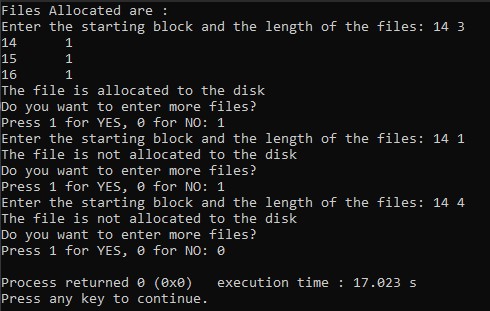
printf("Files Allocated are :\n");

recurse(files);

return 0;

}

Output:



Program 10: Simulate linked file allocation strategy?

Solution:

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

void main()

{

int f[50], p,i, st, len, j, c, k, a;

clrscr();

for(i=0;i<50;i++)

f[i]=0;

printf("Enter how many blocks already allocated: ");

scanf("%d",&p);

printf("Enter blocks already allocated: ");

for(i=0;i<p;i++)

{

scanf("%d",&a);

f[a]=1;

}

x: printf("Enter index starting block and length: ");

scanf("%d%d", &st,&len);

k=len;

if(f[st]==0)

{

for(j=st;j<(st+k);j++)

{

if(f[j]==0)

{

f[j]=1;

printf("%d-------->%d\n",j,f[j]);

}

else

{

printf("%d Block is already allocated \n",j);

k++;

}

}

}

else

printf("%d starting block is already allocated \n",st);

printf("Do you want to enter more file(Yes - 1/No - 0)");

scanf("%d", &c);

if(c==1)

goto x;

else

exit(0);

}

Program Output:

Enter how many blocks already allocated: 3

Enter blocks already allocated: 1 3 5

Enter index starting block and length: 2 2

2-------->1

3 Block is already allocated

4-------->1

Do you want to enter more file(Yes - 1/No - 0)0

Program 11: Simulate Indexed file allocation strategy

solution:

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

void main()

{

int f[50], index[50],i, n, st, len, j, c, k, ind,count=0;

clrscr();

for(i=0;i<50;i++)

f[i]=0;

x:printf("Enter the index block: ");

scanf("%d",&ind);

if(f[ind]!=1)

{

printf("Enter no of blocks needed and no of files for the index %d on the disk : \n", ind);

scanf("%d",&n);

}

else

{

printf("%d index is already allocated \n",ind);

goto x;

}

y: count=0;

for(i=0;i<n;i++)

{

scanf("%d", &index[i]);

if(f[index[i]]==0)

count++;

}

if(count==n)

{

for(j=0;j<n;j++)

f[index[j]]=1;

printf("Allocated\n");

printf("File Indexed\n");

for(k=0;k<n;k++)

printf("%d-------->%d : %d\n",ind,index[k],f[index[k]]);

}

else

{

printf("File in the index is already allocated \n");

printf("Enter another file indexed");

goto y;

}

printf("Do you want to enter more file(Yes - 1/No - 0)");

scanf("%d", &c);

if(c==1)

goto x;

else

exit(0);

getch();

}

Program Output:

Enter the index block: 5

Enter no of blocks needed and no of files for the index 5 on the disk :

4

1 2 3 4

Allocated

File Indexed

5-------->1 : 1

5-------->2 : 1

5-------->3 : 1

5-------->4 : 1

Do you want to enter more file(Yes - 1/No - 0)1

Enter the index block: 4

4 index is already allocated

Enter the index block: 6

Enter no of blocks needed and no of files for the index 6 on the disk :

2

7 8

A5llocated

File Indexed

6-------->7 : 1

6-------->8 : 1

Do you want to enter more file(Yes - 1/No - 0)0

Program 12: Simulate single level directory structure

Solution:

#include<stdio.h>

struct

{

char dname[10],fname[10][10];

int fcnt;

}dir;

void main()

{

int i,ch;

char f[30];

clrscr();

dir.fcnt = 0;

printf("\nEnter name of directory -- ");

scanf("%s", dir.dname);

while(1)

{

printf("\n\n 1. Create File\t2. Delete File\t3. Search File \n 4. Display Files\t5. Exit\nEnter your choice -- ");

scanf("%d",&ch);

switch(ch)

{

case 1: printf("\n Enter the name of the file -- ");

scanf("%s",dir.fname[dir.fcnt]);

dir.fcnt++;

break;

case 2: printf("\n Enter the name of the file -- ");

scanf("%s",f);

for(i=0;i<dir.fcnt;i++)

{

if(strcmp(f, dir.fname[i])==0)

{

printf("File %s is deleted ",f);

strcpy(dir.fname[i],dir.fname[dir.fcnt-1]);

break;

}

}

if(i==dir.fcnt)

printf("File %s not found",f);

else

dir.fcnt--;

break;

case 3: printf("\n Enter the name of the file -- ");

scanf("%s",f);

for(i=0;i<dir.fcnt;i++)

{

if(strcmp(f, dir.fname[i])==0)

{

printf("File %s is found ", f);

break;

}

}

if(i==dir.fcnt)

printf("File %s not found",f);

break;

case 4: if(dir.fcnt==0)

printf("\n Directory Empty");

else

{

printf("\n The Files are -- ");

for(i=0;i<dir.fcnt;i++)

printf("\t%s",dir.fname[i]);

}

break;

default: exit(0);

}

}

getch();

}

OUTPUT:

Enter name of directory -- CSE

1. Create File 2. Delete File 3. Search File

4. Display Files 5. Exit Enter your choice – 1

Enter the name of the file -- A

1. Create File 2. Delete File 3. Search File

4. Display Files 5. Exit Enter your choice – 1

Enter the name of the file -- B

1. Create File 2. Delete File 3. Search File

4. Display Files 5. Exit Enter your choice – 1

Enter the name of the file -- C

1. Create File 2. Delete File 3. Search File

4. Display Files 5. Exit Enter your choice – 4

The Files are -- A B C

1. Create File 2. Delete File 3. Search File

4. Display Files 5. Exit Enter your choice – 3

Enter the name of the file – ABC

File ABC not found

1. Create File 2. Delete File 3. Search File

4. Display Files 5. Exit Enter your choice – 2

Enter the name of the file – B

File B is deleted

1. Create File 2. Delete File 3. Search File

4. Display Files 5. Exit Enter your choice – 5

#include<stdio.h>

#include<stdlib.h>

#include<graphics.h>

main()

{

int gd=DETECT,gm,count,i,j,mid,cir\_x;

char fname[10][20];

clrscr();

initgraph(&gd,&gm,"c:\\tc\\bgi");

cleardevice();

setbkcolor(GREEN);

puts("Enter no of files do u have?");

scanf("%d",&count);

for(i=0;i<count;i++)

{

cleardevice();

setbkcolor(GREEN);

printf("Enter file %d name",i+1);

scanf("%s",fname[i]);

setfillstyle(1,MAGENTA);

mid=640/count;

cir\_x=mid/3;

bar3d(270,100,370,150,0,0);

settextstyle(2,0,4);

settextjustify(1,1);

outtextxy(320,125,"Root Directory");

setcolor(BLUE);

for(j=0;j<=i;j++,cir\_x+=mid)

{

line(320,150,cir\_x,250);

fillellipse(cir\_x,250,30,30);

outtextxy(cir\_x,250,fname[j]);

}

}

}

Program 13: Simulate Two level Directory Structure:

Solution:

#include<stdio.h>

struct

{

char dname[10],fname[10][10];

int fcnt;

}dir[10];

void main()

{

int i,ch,dcnt,k;

char f[30], d[30];

clrscr();

dcnt=0;

while(1)

{

printf("\n\n 1. Create Directory\t 2. Create File\t 3. Delete File");

printf("\n 4. Search File \t \t 5. Display \t 6. Exit \t Enter your choice -- ");

scanf("%d",&ch);

switch(ch)

{

case 1: printf("\n Enter name of directory -- ");

scanf("%s", dir[dcnt].dname);

dir[dcnt].fcnt=0;

dcnt++;

printf("Directory created");

break;

case 2: printf("\n Enter name of the directory -- ");

scanf("%s",d);

for(i=0;i<dcnt;i++)

if(strcmp(d,dir[i].dname)==0)

{

printf("Enter name of the file -- ");

scanf("%s",dir[i].fname[dir[i].fcnt]);

dir[i].fcnt++;

printf("File created");

break;

}

if(i==dcnt)

printf("Directory %s not found",d);

break;

case 3: printf("\nEnter name of the directory -- ");

scanf("%s",d);

for(i=0;i<dcnt;i++)

{

if(strcmp(d,dir[i].dname)==0)

{

printf("Enter name of the file -- ");

scanf("%s",f);

for(k=0;k<dir[i].fcnt;k++)

{

if(strcmp(f, dir[i].fname[k])==0)

{

printf("File %s is deleted ",f);

dir[i].fcnt--;

strcpy(dir[i].fname[k],dir[i].fname[dir[i].fcnt]);

goto jmp;

}

}

printf("File %s not found",f);

goto jmp;

}

}

printf("Directory %s not found",d);

jmp : break;

case 4: printf("\nEnter name of the directory -- ");

scanf("%s",d);

for(i=0;i<dcnt;i++)

{

if(strcmp(d,dir[i].dname)==0)

{

printf("Enter the name of the file -- ");

scanf("%s",f);

for(k=0;k<dir[i].fcnt;k++)

{

if(strcmp(f, dir[i].fname[k])==0)

{

printf("File %s is found ",f);

goto jmp1;

}

}

printf("File %s not found",f);

goto jmp1;

}

}

printf("Directory %s not found",d);

jmp1: break;

case 5: if(dcnt==0)

printf("\nNo Directory's ");

else

{

printf("\nDirectory\tFiles");

for(i=0;i<dcnt;i++)

{

printf("\n%s\t\t",dir[i].dname);

for(k=0;k<dir[i].fcnt;k++)

printf("\t%s",dir[i].fname[k]);

}

}

break;

default:exit(0);

}

}

getch();

}

OUTPUT:

1. Create Directory 2. Create File 3. Delete File

4. Search File 5. Display 6. Exit Enter your choice -- 1

Enter name of directory -- DIR1

Directory created

1. Create Directory 2. Create File 3. Delete File

4. Search File 5. Display 6. Exit Enter your choice -- 1

Enter name of directory -- DIR2

Directory created

1. Create Directory 2. Create File 3. Delete File

4. Search File 5. Display 6. Exit Enter your choice -- 2

Enter name of the directory – DIR1

Enter name of the file -- A1

File created

1. Create Directory 2. Create File 3. Delete File

4. Search File 5. Display 6. Exit Enter your choice -- 2

Enter name of the directory – DIR1

Enter name of the file -- A2

File created

1. Create Directory 2. Create File 3. Delete File

4. Search File 5. Display 6. Exit Enter your choice -- 2

Enter name of the directory – DIR2

Enter name of the file -- B1

File created

1. Create Directory 2. Create File 3. Delete File

4. Search File 5. Display 6. Exit Enter your choice -- 5

Directory Files

DIR1 A1 A2

DIR2 B1

1. Create Directory 2. Create File 3. Delete File

4. Search File 5. Display 6. Exit Enter your choice -- 4

Enter name of the directory – DIR

Directory not found

1. Create Directory 2. Create File 3. Delete File

4. Search File 5. Display 6. Exit Enter your choice -- 3

Enter name of the directory – DIR1

Enter name of the file -- A2

File A2 is deleted

1. Create Directory 2. Create File 3. Delete File

4. Search File 5. Display 6. Exit Enter your choice – 6

Program 14: Simulate MFT AND MVT

Solutions:

**MFT MEMORY MANAGEMENT TECHNIQUE**

#include<stdio.h>

#include<conio.h>

main()

{

int ms, bs, nob, ef,n, mp[10],tif=0;

int i,p=0;

clrscr();

printf("Enter the total memory available (in Bytes) -- ");

scanf("%d",&ms);

printf("Enter the block size (in Bytes) -- ");

scanf("%d", &bs);

nob=ms/bs;

ef=ms - nob\*bs;

printf("\nEnter the number of processes -- ");

scanf("%d",&n);

for(i=0;i<n;i++)

{

printf("Enter memory required for process %d (in Bytes)-- ",i+1);

scanf("%d",&mp[i]);

}

printf("\nNo. of Blocks available in memory -- %d",nob);

printf("\n\nPROCESS\tMEMORY REQUIRED\t ALLOCATED\tINTERNAL FRAGMENTATION");

for(i=0;i<n && p<nob;i++)

{

printf("\n %d\t\t%d",i+1,mp[i]);

if(mp[i] > bs)

printf("\t\tNO\t\t---");

else

{

printf("\t\tYES\t%d",bs-mp[i]);

tif = tif + bs-mp[i];

p++;

}

}

if(i<n)

printf("\nMemory is Full, Remaining Processes cannot be accomodated");

printf("\n\nTotal Internal Fragmentation is %d",tif);

printf("\nTotal External Fragmentation is %d",ef);

getch();

}

**INPUT**

Enter the total memory available (in Bytes) -- 1000

Enter the block size (in Bytes)-- 300

Enter the number of processes – 5

Enter memory required for process 1 (in Bytes) -- 275

Enter memory required for process 2 (in Bytes) -- 400

Enter memory required for process 3 (in Bytes) -- 290

Enter memory required for process 4 (in Bytes) -- 293

Enter memory required for process 5 (in Bytes) -- 100

No. of Blocks available in memory -- 3

**OUTPUT**

PROCESS MEMORY-REQUIRED ALLOCATED INTERNAL-FRAGMENTATION

1                             275                              YES                               25

2                             400                               NO                                 -----

3                             290                              YES                               10

4                             293                              YES                                 7

Memory is Full, Remaining Processes cannot be accommodated

Total Internal Fragmentation is 42

Total External Fragmentation is 100

 --------------------------------------------------------------------------------------------

**MVT MEMORY MANAGEMENT TECHNIQUE**

#include<stdio.h>

#include<conio.h>

main()

{

int ms,mp[10],i, temp,n=0;

char ch = 'y';

clrscr();

printf("\nEnter the total memory available (in Bytes)-- ");

scanf("%d",&ms);

temp=ms;

for(i=0;ch=='y';i++,n++)

{

printf("\nEnter memory required for process %d (in Bytes) -- ",i+1);

scanf("%d",&mp[i]);

if(mp[i]<=temp)

{

printf("\nMemory is allocated for Process %d ",i+1);

temp = temp - mp[i];

}

else

{

printf("\nMemory is Full");

break;

}

printf("\nDo you want to continue(y/n) -- ");

scanf(" %c", &ch);

}

printf("\n\nTotal Memory Available -- %d", ms);

printf("\n\n\tPROCESS\t\t MEMORY ALLOCATED ");

for(i=0;i<n;i++)

printf("\n \t%d\t\t%d",i+1,mp[i]);

printf("\n\nTotal Memory Allocated is %d",ms-temp);

printf("\nTotal External Fragmentation is %d",temp);

 getch();

}

**INPUT**

Enter the total memory available (in Bytes) -- 1000

Enter memory required for process 1 (in Bytes) -- 400

Memory is allocated for Process 1

Do you want to continue(y/n) -- y

Enter memory required for process 2 (in Bytes) -- 275

Memory is allocated for Process 2

Do you want to continue(y/n) -- y

Enter memory required for process 3 (in Bytes) -- 550

**OUTPUT**

Memory is Full

Total Memory Available -- 1000

PROCESS MEMORY-ALLOCATED

1                        400

2                        275

Total Memory Allocated is 675

Total External Fragmentation is 325