**AVANTHI PG COLLEGE**

**Affiliated to Osmania University**

**Department of MCA**

**OPERATING SYSTEMS LAB MANUAL**

With effect from the academic year 2020-2021

**Operating Systems Lab**

LCC251 Credits : 2

Instruction 3P hrs per week Duration of SEE 3 hours

CIE 25 marks SEE 50 marks

**Course Objectives**

1. Learn shell commands and scripting

2. Learn CPU scheduling algorithms

3. Learn memory management algorithms

4. Learn synchronization problems

5. Explore file allocation strategies and disk scheduling algorithms

**Course Outcomes**

1. Be able to execute shell commands and write shell scripts

2. Be able to write programs on CPU scheduling

3. Be able to create memory management algorithms

4. Be able to execute programs to demonstrate synchronization problems

5. Be able to implement file allocation methods and be able to create disk scheduling algorithms

**Programs**

1. Unix Shell Commands

a) File handling commands b) Directory handling commands c) General purpose commands

2. Unix Shell Scripts

a) Print Multiplication table of a given no. using all loops

b) Perform all arithmetic operations

c) Print the type of a file

d) Rename all files whose names end with .c as .old

e) Display the no. of lines in each of text file in a given dir

3. Simulate the following CPU scheduling algorithms.

a. FCFS

b. SJF

c. Round Robin

d. Priority.

4. Write a C program to simulate producer-consumer problem using Semaphores

5. Write a C program to simulate the concept of Dining-philosophers problem.

6. Simulate MVT and MFT.

7. Write a C program to simulate the following contiguous memory allocation techniques

a. Worst fit

b. Best fit

c. First fit.

8. Simulate following page replacement algorithms

a. FIFO

b. LRU

c. OPTIMAL

9. Simulate following File Organization Techniques a. Single level directory b. Two level directory

10. Simulate following file allocation strategies

a. Sequential

b. Indexed

c. Linked.

11. Simulate Bankers Algorithm for Dead Lock Avoidance.

12. Simulate Bankers Algorithm for Dead Lock Prevention.

13. Write a C program to simulate disk scheduling algorithms.

a. FCFS

b. SCAN

c. C-SCAN

# GENERAL LABORATORY INSTRUCTIONS

1. Students are advised to come to the laboratory at least 5 minutes before (to the starting time), those who come after 5 minutes will not be allowed into the lab.
2. Plan your task properly much before to the commencement, come prepared to the lab with the synopsis / program / experiment details.
3. Student should enter into the laboratory with:
   1. Laboratory observation notes with all the details (Problem statement, Aim, Algorithm, Procedure, Program, Expected Output, etc.,) filled in for the lab session.
   2. Laboratory Record updated up to the last session experiments and other utensils (if any) needed in the lab.
   3. Proper Dress code and Identity card.
4. Sign in the laboratory login register, write the TIME-IN, and occupy the computer system allotted to you by the faculty.
5. Execute your task in the laboratory, and record the results / output in the lab observation note book, and get certified by the concerned faculty.
6. All the students should be polite and cooperative with the laboratory staff, must maintain the discipline and decency in the laboratory.
7. Computer labs are established with sophisticated and high end branded systems, which should be utilized properly.
8. Students / Faculty must keep their mobile phones in SWITCHED OFF mode during the lab sessions. Misuse of the equipment, mis behaviours with the staff and systems etc., will attract severe punishment.
9. Students must take the permission of the faculty in case of any urgency to go out; if anybody found loitering outside the lab / class without permission during working hours will be treated seriously and punished appropriately.
10. Students should LOG OFF/ SHUT DOWN the computer system before he/she leaves the lab after completing the task (experiment) in all aspects. He/she must ensure the system / seat is kept properly.

**Head of the Department Principal**

# EXPERIMENT NO.1

**CPU SCHEDULINGALGORITHMS**

1. **FIRST COME FIRST SERVE:**

**AIM:** To write a c program to simulate the CPU scheduling algorithm First Come First Serve (FCFS)

# DESCRIPTION:

To calculate the average waiting time using the FCFS algorithm first the waiting time of the first process is kept zero and the waiting time of the second process is the burst time of the first process and the waiting time of the third process is the sum of the burst times of the first and the second process and so on. After calculating all the waiting times the average waiting time is calculated as the average of all the waiting times. FCFS mainly says first come first serve the algorithm which came first will be served first.

# ALGORITHM:

Step 1: Start the process

Step 2: Accept the number of processes in the ready Queue

Step 3: For each process in the ready Q, assign the process name and the burst time Step 4: Set the waiting of the first process as ‗0‘and its burst time as its turnaround time

Step 5: for each process in the Ready Q calculate a).Waiting time (n) = waiting time (n-1) + Burst time (n-1) b). Turnaround time (n)= waiting time(n)+Burst time(n)

Step 6: Calculate

* 1. Average waiting time = Total waiting Time / Number of process
  2. Average Turnaround time = Total Turnaround Time / Number of process Step 7: Stop the process

# SOURCE CODE :

// First Come First Serve ( FCFS Program in C #include<stdio.h> #include<conio.h>

main()

{

int bt[20], wt[20], tat[20], i, n; float wtavg, tatavg; clrscr();

printf("\nEnter the number of processes -- "); scanf("%d", &n); for(i=0;i<n;i++)

{

printf("\nEnter Burst Time for Process %d -- ", i); scanf("%d", &bt[i]);

}

wt[0] = wtavg = 0; tat[0] = tatavg = 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("\t PROCESS \tBURST TIME \t WAITING TIME\t TURNAROUND TIME\n");

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

printf("\n\t P%d \t\t %d \t\t %d \t\t %d", i, bt[i], wt[i], tat[i]); printf("\nAverage Waiting Time --

%f", wtavg/n);

printf("\nAverage Turnaround Time -- %f", tatavg/n); getch();

}

|  |  |  |  |
| --- | --- | --- | --- |
| **INPUT:** |  |  |  |
| Enter the number of processes -- 3  Enter Burst Time for Process 0 -- 24  Enter Burst Time for Process 1 -- 3  Enter Burst Time for Process 2 -- 3 | | | |
| **OUTPUT** |  |  |  |
| PROC ESS | BURST TIME | WAITING TIME | TURNAROUND TIME |
| P0 | 24 | 0 | 24 |
| P1 | 3 | 24 | 27 |
| P2 | 3 | 27 | 30 |
| Average Waiting Time-- 17.000000 | | |  |
| Average Turnaround Time -- 27.000000 | | |  |

# SHORTEST JOB FIRST:

**AIM:** To write a program to stimulate the CPU scheduling algorithm Shortest job first (Non- Preemption)

# DESCRIPTION:

To calculate the average waiting time in the shortest job first algorithm the sorting of the process based on their burst time in ascending order then calculate the waiting time of each process as the sum of the bursting times of all the process previous or before to that process.

# ALGORITHM:

Step 1: Start the process

Step 2: Accept the number of processes in the ready Queue

Step 3: For each process in the ready Q, assign the process id and accept the CPU burst time Step 4: Start the Ready Q according the shortest Burst time by sorting according to lowest to

highest burst time.

Step 5: Set the waiting time of the first process as ‗0‘ and its turnaround time as its burst time. Step 6: Sort the processes names based on their Burt time

Step 7: For each process in the ready queue, calculate

1. Waiting time(n)= waiting time (n-1) + Burst time (n-1)
2. Turnaround time (n)= waiting time(n)+Burst time(n)

Step 8: Calculate

1. Average waiting time = Total waiting Time / Number of process
2. Average Turnaround time = Total Turnaround Time / Number of process Step 9: Stop the process

# SOURCE CODE :

#include<stdio.h> #include<conio.h> main()

{

int p[20], bt[20], wt[20], tat[20], i, k, n, temp; float wtavg, tatavg; clrscr();

printf("\nEnter the number of processes -- "); scanf("%d", &n); for(i=0;i<n;i++)

{

p[i]=i;

printf("Enter Burst Time for Process %d -- ", i); scanf("%d", &bt[i]);

}

for(i=0;i<n;i++) for(k=i+1;k<n;k++) if(bt[i]>bt[k])

{

temp=bt[i]; bt[i]=bt[k]; bt[k]=temp;

temp=p[i]; p[i]=p[k]; p[k]=temp;

}

wt[0] = wtavg = 0;

tat[0] = tatavg = 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("\n\t PROCESS \tBURST TIME \t WAITING TIME\t TURNAROUND TIME\n");

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

printf("\n\t P%d \t\t %d \t\t %d \t\t %d", p[i], bt[i], wt[i], tat[i]); printf("\nAverage Waiting Time

-- %f", wtavg/n);

printf("\nAverage Turnaround Time -- %f", tatavg/n); getch();

}

|  |
| --- |
| INPUT: |
| Enter the number of processes --4 Enter Burst Time for Process 0 --6 Enter Burst Time for Process 1 --8  Enter Burst Time for Process 2 --7 Enter Burst Time for Process 3 --3 |

|  |  |  |  |
| --- | --- | --- | --- |
| **OUTPUT:** |  |  |  |
| PROCES S | BURST TIME | WAITING TIME | TURNAROUND TIME |
| P3 | 3 | 0 | 3 |
| P0 | 6 | 3 | 9 |
| P2 | 7 | 9 | 16 |
| P1 | 8 | 16 | 24 |
| Average Waiting Time -- 7.000000 Average Turnaround Time --13.000000 | | |  |

# ROUND ROBIN:

**AIM:** To simulate the CPU scheduling algorithm round-robin.

# DESCRIPTION:

To aim is to calculate the average waiting time. There will be a time slice, each process should be executed within that time-slice and if not it will go to the waiting state so first check whether the burst time is less than the time-slice. If it is less than it assign the waiting time to the sum of the total times. If it is greater than the burst-time then subtract the time slot from the actual burst time and increment it by time-slot and the loop continues until all the processes are completed.

# ALGORITHM:

Step 1: Start the process

Step 2: Accept the number of processes in the ready Queue and time quantum (or) time slice Step 3: For each process in the ready Q, assign the process id and accept the CPU burst time Step 4: Calculate the no. of time slices for each process where No. of time slice for process (n) = burst time process (n)/time slice

Step 5: If the burst time is less than the time slice then the no. of time slices =1.

Step 6: Consider the ready queue is a circular Q, calculate

1. Waiting time for process (n) = waiting time of process(n-1)+ burst timeof process(n-1 ) + the time difference in getting the CPU fromprocess(n-1)
2. Turnaround time for process(n) = waiting time of process(n) + burst time of process(n)+ the time difference in getting CPU from process(n).

Step 7: Calculate

1. Average waiting time = Total waiting Time / Number of process
2. Average Turnaround time = Total Turnaround Time / Number ofprocess Step 8: Stop the process

# SOURCE CODE

#include<stdio.h> main()

{

int i,j,n,bu[10],wa[10],tat[10],t,ct[10],max; float awt=0,att=0,temp=0; clrscr();

printf("Enter the no of processes -- "); scanf("%d",&n); for(i=0;i<n;i++)

{

printf("\nEnter Burst Time for process %d -- ", i+1); scanf("%d",&bu[i]); ct[i]=bu[i];

}

printf("\nEnter the size of time slice -- "); scanf("%d",&t);

max=bu[0]; for(i=1;i<n;i++) if(max<bu[i]) max=bu[i]; for(j=0;j<(max/t)+1;j++) for(i=0;i<n;i++) if(bu[i]!=0)

if(bu[i]<=t)

{

tat[i]=temp+bu[i]; temp=temp+bu[i]; bu[i]=0;

}

else

{

bu[i]=bu[i]-t; temp=temp+t;

}

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

{

wa[i]=tat[i]-ct[i]; att+=tat[i]; awt+=wa[i];

}

printf("\nThe Average Turnaround time is -- %f",att/n); printf("\nThe Average Waiting time is --

%f ",awt/n);

printf("\n\tPROCESS\t BURST TIME \t WAITING TIME\tTURNAROUND TIME\n");

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

printf("\t%d \t %d \t\t %d \t\t %d \n",i+1,ct[i],wa[i],tat[i]); getch();

}

# INPUT:

Enter the no of processes – 3

Enter Burst Time for process 1 – 24 Enter Burst Time for process 2 -- 3 Enter Burst Time for process 3 – 3 Enter the size of time slice – 3

|  |  |  |  |
| --- | --- | --- | --- |
| **OUTPUT:**  PROCESS | BURST TIME | WAITING TIME | TURNAROUNDTIME |
| 1 | 24 | 6 | 30 |
| 2 | 3 | 4 | 7 |
| 3 | 3 | 7 | 10 |

The Average Turnaround time is – 15.666667 The Average Waiting time is 5.666667

# PRIORITY:

**AIM**: To write a c program to simulate the CPU scheduling priority algorithm.

# DESCRIPTION:

To calculate the average waiting time in the priority algorithm, sort the burst times according to their priorities and then calculate the average waiting time of the processes. The waiting time of each process is obtained by summing up the burst times of all the previous processes.

# ALGORITHM:

Step 1: Start the process

Step 2: Accept the number of processes in the ready Queue

Step 3: For each process in the ready Q, assign the process id and accept the CPU burst time Step 4: Sort the ready queue according to the priority number.

Step 5: Set the waiting of the first process as ‗0‘ and its burst time as its turnaround time Step 6: Arrange the processes based on process priority

Step 7: For each process in the Ready Q calculate

1. Waiting time(n)= waiting time (n-1) + Burst time (n-1)
2. Turnaround time (n)= waiting time(n)+Burst time(n)

Step 8: Calculate

1. Average waiting time = Total waiting Time / Number of process
2. Average Turnaround time = Total Turnaround Time / Number of process Print the results in an order.

Step9:Stop

# SOURCE CODE :

#include<stdio.h> main()

{

int i,j,n,bu[10],wa[10],tat[10],t,ct[10],max; float awt=0,att=0,temp=0; clrscr();

printf("Enter the no of processes -- "); scanf("%d",&n); for(i=0;i<n;i++)

{

printf("\nEnter Burst Time for process %d -- ", i+1); scanf("%d",&bu[i]); ct[i]=bu[i];

}

printf("\nEnter the size of time slice -- "); scanf("%d",&t);

max=bu[0]; for(i=1;i<n;i++) if(max<bu[i]) max=bu[i]; for(j=0;j<(max/t)+1;j++) for(i=0;i<n;i++) if(bu[i]!=0)

if(bu[i]<=t)

{

tat[i]=temp+bu[i]; temp=temp+bu[i]; bu[i]=0;

}

else

o

{

bu[i]=bu[i]-t; temp=temp+t;

}

for(i=0;i<n;i++) { wa[i]=tat[i]-ct[i]; att+=tat[i]; awt+=wa[i];

}

printf("\nThe Average Turnaround time is -- %f",att/n); printf("\nThe Average Waiting time is --

%f ",awt/n);

printf("\n\tPROCESS\t BURST TIME \t WAITING TIME\tTURNAROUND TIME\n");

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

printf("\t%d \t %d \t\t %d \t\t %d \n",i+1,ct[i],wa[i],tat[i]); getch();

}

# INPUT:

Enter the no of processes – 3

Enter Burst Time for process 1 – 24 Enter Burst Time for process 2 -- 3 Enter Burst Time for process 3 – 3 Enter the size of time slice – 3 **OUTPUT:**

|  |  |  |  |
| --- | --- | --- | --- |
| PROCESS | BURST TIME | WAITING TIME | TURNAROUNDTIME |
| 1 | 24 | 6 | 30 |
| 2 | 3 | 4 | 7 |
| 3 | 3 | 7 | 10 |

The Average Turnaround time is – 15.666667 The Average Waiting time is 5.666667

# EXPERIMENT.NO.2

**FILE ALLOCATION STRATEGIES**

* 1. **SEQUENTIAL:**

The most common form of file structure is the sequential file in this type of file, a fixed format is used for records. All records (of the system) have the same length, consisting of the same number of fixed length fields in a particular order because the length and position of each field are known, only the values of fields need to be stored, the field name and length for each field are attributes of the file structure.

# ALGORITHM:

Step 1: Start the program.

Step 2: Get the number of files.

Step 3: Get the memory requirement of each file.

Step 4: Allocate the required locations to each in sequential order.Randomly select a location from availablelocation s1= random(100);

1. Check whether the required locations are free from the selected location. if(b[s1].flag==0)…{

for (j=s1;j<s1+p[i];j++){ if((b[j].flag)==0)count++;

}

if(count==p[i]) break;

}

1. Allocate and set flag=1 to the allocated locations. for(s=s1;s<(s1+p[i]);s++)

{

k[i][j]=s; j=j+1; b[s].bno=s; b[s].flag=1;

}

Step 5: Print the results file no, length, Blocks allocated. Step 6: Stop the program

# SOURCE CODE :

#include<stdio.h> main()

{

int f[50],i,st,j,len,c,k; clrscr(); for(i=0;i<50;i++) f[i]=0; X:

printf("\n Enter the starting block & length of file"); scanf("%d%d",&st,&len); for(j=st;j<(st+len);j++) if(f[j]==0)

{ f[j]=1;

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

}

else

{

printf("Block already allocated"); break;

}

if(j==(st+len))

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

printf("\n if u want to enter more files?(y-1/n-0)"); scanf("%d",&c); if(c==1) goto X; else exit();

getch();

} OUTPUT:

Enter the starting block & length of file 4 10 4->1

5->1

6->1

7->1

8->1

9->1

10->1

11->1

12->1

13->1

The file is allocated to disk.

# INDEXED:

**AIM:** To implement allocation method using chained method

# DESCRIPTION:

In the chained method file allocation table contains a field which points to starting block of memory. From it for each bloc a pointer is kept to next successive block. Hence, there is no external fragmentation.

# ALGORITHM:

Step 1: Start the program.

Step 2: Get the number of files.

Step 3: Get the memory requirement of each file.

Step 4: Allocate the required locations by selecting a location randomly q= random(100); Check whether the selected location is free .

If the location is free allocate and set flag=1 to the allocated locations. q=random(100);

if(b[q].flag==0)

b[q].flag=1;

b[q].fno=j;

r[i][j]=q;

Step 5: Print the results file no, length, Blocks allocated. Step 6: Stop the program

# SOURCE CODE :

#include<stdio.h>

int f[50],i,k,j,inde[50],n,c,count=0,p; main()

{

clrscr(); for(i=0;i<50;i++) f[i]=0;

x: printf("enter index block\t"); scanf("%d",&p); if(f[p]==0)

{ f[p]=1;

printf("enter no of files on index\t"); scanf("%d",&n);

}

else

{

printf("Block already allocated\n"); goto x;

}

for(i=0;i<n;i++) scanf("%d",&inde[i]); for(i=0;i<n;i++) if(f[inde[i]]==1)

{

printf("Block already allocated"); goto x;

}

for(j=0;j<n;j++) f[inde[j]]=1;

printf("\n allocated"); printf("\n file indexed"); for(k=0;k<n;k++) printf("\n %d->%d:%d",p,inde[k],f[inde[k]]);

printf(" Enter 1 to enter more files and 0 to exit\t"); scanf("%d",&c); if(c==1) goto x; else exit();

getch();

}

# OUTPUT:

enter index block 9

Enter no of files on index 3 1 2 3 Allocated File indexed 9->1:1

9->2;1

9->3:1

enter 1 to enter more files and 0 to exit

# EXPERIMENT NO:3

**AIM:** To implement linked file allocation strategy.

# DESCRIPTION:

In the chained method file allocation table contains a field which points to starting block of memory. From it for each bloc a pointer is kept to next successive block. Hence, there is no external fragmentation

# ALGORTHIM:

Step 1: Start the program.

Step 2: Get the number of files.

Step 3: Get the memory requirement of each file.

Step 4: Allocate the required locations by selecting a location randomly q= random(100);

* + 1. Check whether the selected location is free .
    2. If the location is free allocate and set flag=1 to the allocated locations. While allocating next location address to attach it to previous location for(i=0;i<n;i++)

{

for(j=0;j<s[i];j++)

{

q=random(100); if(b[q].flag==0) b[q].flag=1; b[q].fno=j;

r[i][j]=q; if(j>0)

{

p=r[i][j-1]; b[p].next=q;}

}

}

Step 5: Print the results file no, length ,Blocks allocated. Step 6: Stop the program

# SOURCE CODE :

#include<stdio.h> main()

{

int f[50],p,i,j,k,a,st,len,n,c; clrscr(); for(i=0;i<50;i++) f[i]=0;

printf("Enter how many blocks that are already allocated"); scanf("%d",&p); printf("\nEnter the blocks no.s that are already allocated"); for(i=0;i<p;i++)

{

scanf("%d",&a); f[a]=1;

}

X:

printf("Enter the starting index block & length"); scanf("%d%d",&st,&len); k=len; for(j=st;j<(k+st);j++)

{

if(f[j]==0)

{ f[j]=1;

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

}

else

{

printf("\n %d->file is already allocated",j); k++;

}

}

printf("\n If u want to enter one more file? (yes-1/no-0)"); scanf("%d",&c); if(c==1)

goto X; else exit(); getch( );}

# OUTPUT:

Enter how many blocks that are already allocated 3 Enter the blocks no.s that are already allocated 4 7 9 Enter the starting index block & length 3 7

3->1

4->1 file is already allocated 5->1

6->1

7->1 file is already allocated 8->1 9->1file is already allocated 10->1 11->1

12->1

# EXPERIMENT NO. 4 MEMORY MANAGEMAENT

**A). MEMORY MANAGEMENT WITH FIXED PARTITIONING TECHNIQUE (MFT)**

**AIM:** To implement and simulate the MFT algorithm.

# DESCRIPTION:

In this the memory is divided in two parts and process is fit into it. The process which is best suited in to it is placed in the particular memory where it suits. We have to check memory partition. If it suits, its status should be changed.

# ALGORITHM:

Step1: Start the process.

Step2: Declarevariables.

Step3: Enter total memory size ms. Step4: Allocate memory for os.

Ms=ms-os

Step5: Read the no partition to be divided n Partition size=ms/n. Step6: Read the process no and process size.

Step 7: If process size is less than partition size allot alse blocke the process. While allocating update memory wastage-external fragmentation.

if(pn[i]==pn[j])f=1;

if(f==0){ if(ps[i]<=siz)

{

extft=extft+size-ps[i];avail[i]=1; count++;

}

}

Step 8: Print the results

# SOURCE CODE :

#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 (i Bytes) -- | 275 |
| Enter memory required for process 2 (i Bytes) -- | 400 |
| Enter memory required for process 3 (i Bytes) -- | 290 |
| Enter memory required for process 4 (i  Bytes) -- | 293 |
| Enter memory required for process 5 (i  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

# B) MEMORY VARIABLE PARTIONING TYPE (MVT)

**AIM:** To write a program to simulate the MVT algorithm

# ALGORITHM:

Step1: start the process.

Step2: Declare variables.

Step3: Enter total memory size ms. Step4: Allocate memory for os.

Ms=ms-os

Step5: Read the no partition to be divided n Partition size=ms/n. Step6: Read the process no and process size.

Step 7: If process size is less than partition size allot alse blocke the process. While allocating update memory wastage-external fragmentation. if(pn[i]==pn[j]) f=1; if(f==0){ if(ps[i]<=size)

{

extft=extft+size-ps[i];avail[i]=1; count++;

}

}

Step 8: Print the results Step 9: Stop the process.

# SOURCE CODE:

#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();

}

# OUTPUT:

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 Memory is Full

Total Memory Available – 1000

PROCESS MEMORY ALLOCATED 1 400

2 275

Total Memory Allocated is 675 Total External Fragmentation is 325

# EXPERIMENT NO: 5

**FILE ORGANIZATION TECHNIQUES**

1. **SINGLE LEVEL DIRECTORY:**

**AIM:** Program to simulate Single level directory file organization technique.

# DESCRIPTION:

The directory structure is the organization of files into a hierarchy of folders. In a single-level directory system, all the files are placed in one directory. There is a root directory which has all files. It has a simple architecture and there are no sub directories. Advantage of single level directory system is that it is easy to find a file in the directory. In the two-level directory system, each user has own user file directory (UFD). The system maintains a master block that has one entry for each user. This master block contains the addresses of the directory of the users. When a user job starts or a user logs in, the system's master file directory (MFD) is searched. When a user refers to a particular file, only his own UFD is searched.

This effectively solves the name collision problem and isolates users from one another. Hierarchical directory structure allows users to create their own subdirectories and to organize their files accordingly. A tree is the most common directory structure. The tree has a root directory, and every file in the system has a unique path name. A directory (or subdirectory) contains a set of files or subdirectories.

# SOURCE CODE :

#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\n1. 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("\nEnter the name of the file -- "); scanf("%s",dir.fname[dir.fcnt]); dir.fcnt++; break;

case 2: printf("\nEnter 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("\nEnter 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("\nDirectory Empty"); else

{

printf("\nThe 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 24 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

# TWO LEVEL DIRECTORY

**AIM:** Program to simulate two level file organization technique

# ALGORITHM:

Step 1:Start

Step 2: Initialize structure elements struct tree\_ element char name[20]; Initialize integer variables x, y, ftype, lx, rx, nc, level;

struct tree\_element

\*link[5];}

typedef structure tree\_element node; Step 3:start main function

Step 4: Step variables gd=DETECT,gm; node \*root; root=NULL; Step 5:create structure using create(&root,0,"null",0,639,320);

Step 6:initgraph(&gd, &gm,"c:\tc\bgi"); display(root); closegraph(); Step 7: end main function

Step 8: Initialize variables i,gap;

Step 9:if check \*root==NULL (\*root)=(node\*)malloc(sizeof(node)); enter name of ir file namein dname; fflush(stdin);

gets((\*root)->name);

Step 10 if check lev==0||lev==1 (\*root)->ftype=1; else(\*root)->ftype=2;

(\*root)->level=lev; (\*root)->y=50+lev\*5; (\*root)->x=x; (\*root)->lx=lx; (\*root)->rx=rx; Step 11:for i=0&&i<5 increment i

(\*root)->link[i]=NULL;

if check (\*root)->ftype==1

Step 12: if check (lev==0||lev==1) if check(\*root)->level==0

print "how many users" else print"how many files" print (\*root)->name

read (\*root)->nc

Step 13:Then (\*root)->nc=0; if check(\*root)->nc==0 gap=rx-lx; else gap=(rx-lx)/(\*root)->nc; Step 14:for i=0&&i<(\*root)->nc increment i;

create(&((\*root)->link[i]),lev+1,(\*root)->name, lx+gap\*i,lx+gap\*i+gap,lx+gap\*i+gap/2); then

(\*root)->nc=0;

Step 15: Initialize e display function Initialize i set textstyle(2,0,4); set textjustify(1,1); set fillstyle(1,BLUE);

setcolor(14); step 13:if check root!=NULL Step 16:for i=0&&i<root->nc increment i

line(root->x,root->y,root->link[i]->x,root->link[i]->y);

Step 17: if check root->ftype==1

bar3d(root->x-20,root->y-10,root->x+20,root->y+10,0,0); else fill ellipse(root->x,root-

>y,20,20);

out textxy(root->x,root->y,root->name);

Step 18:for i=0&&i<root->nc increment i display(root->link[i]); Step 19:End

# SOURCE CODE :

#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\n1. Create Directory\t2. Create File\t3. Delete File"); printf("\n4. Search File\t\t5. Display\t6. Exit\t

Enter your choice --"); scanf("%d",&ch); switch(ch)

{

case 1: printf("\nEnter name of directory -- "); scanf("%s", dir[dcnt].dname); dir[dcnt].fcnt=0;

dcnt++;

printf("Directory created"); break;

case 2: 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",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++)

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 ot 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

26

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 – 6

# EXPERIMENT NO 6:

**FILE ORGANIZATION TECHNIQUES**

1. Hierarchical
2. DAG

# HIERARCHICAL DIRECTORY:

The two level directories eliminate name conflicts among users but it is not satisfactory for users but it is not satisfactory for users with a large no of files. To avoid this, create the subdirectory and load the same type of the files into the subdirectory. So, in this method each can have as many directories are needed. This directory structure looks like tree, that’s why it is also said to be tree-level directory structure

# SOURCE CODE:

#include<stdio.h> #include<graphics.h> struct tree\_element

{

char name[20];

int x,y,ftype,lx,rx,nc,level; struct tree\_element \*link[5];

};

typedef struct tree\_element node; void main()

{

int gd=DETECT,gm; node \*root; root=NULL; clrscr();

create(&root,0,"root",0,639,320); clrscr(); initgraph(&gd,&gm,"c:\\tc\\BGI"); display(root);

getch(); closegraph();

}

create(node \*\*root,int lev,char \*dname,int lx,int rx,int x)

{

int i,gap; if(\*root==NULL)

{

(\*root)=(node \*)malloc(sizeof(node)); printf("Enter name of dir/file(under %s) :",dname); fflush(stdin);

gets((\*root)->name);

printf("enter 1 for Dir/2 forfile :"); scanf("%d",&(\*root)->ftype);

(\*root)->level=lev; (\*root)->y=50+lev\*50; (\*root)->x=x;

(\*root)->lx=lx; (\*root)->rx=rx; for(i=0;i<5;i++)

(\*root)->link[i]=NULL; if((\*root)->ftype==1)

{

printf("No of sub directories/files(for %s):",(\*root)->name); scanf("%d",&(\*root)->nc); if((\*root)->nc==0)

gap=rx-lx;

else gap=(rx-lx)/(\*root)->nc; for(i=0;i<(\*root)->nc;i++)

create(&((\*root)->link[i]),lev+1,(\*root)->name,lx+gap\*i,lx+gap\*i+gap,lx+gap\*i+gap/2);

}

else (\*root)->nc=0;

}

}

display(node \*root)

{

int i; settextstyle(2,0,4); settextjustify(1,1); setfillstyle(1,BLUE);

setcolor(14); if(root!=NULL)

{

for(i=0;i<root->nc;i++)

{

line(root->x,root->y,root->link[i]->x,root->link[i]->y);

}

if(root->ftype==1) bar3d(root->x-20,root->y-10,root->x+20,root->y+10,0,0); else fillellipse(root->x,root->y,20,20);

outtextxy(root->x,root->y,root->name); for(i=0;i<root->nc;i++)

{

display(root->link[i]);

}

}

}

# OUTPUT:

Enter Name of dir/file (under root): ROOT Enter 1 for Dir / 2 For File : 1

No of subdirectories / files (for ROOT) :2 Enter Name of dir/file (under ROOT):USER 1 Enter 1 for Dir /2 for file:1

No of subdirectories /files (for USER 1):1

Enter Name of dir/file (under USER 1):SUBDIR

Enter 1 for Dir /2 for file:1

No of subdirectories /files (for SUBDIR):2 Enter Name of dir/file (under USER 1):

JAVA Enter 1 for Dir /2 for file:1

No of subdirectories /files (for JAVA): 0 Enter Name of dir/file (under SUBDIR):VB Enter 1 for Dir /2 for file:1

No of subdirectories /files (for VB): 0

Enter Name of dir/file (under ROOT):USER2 Enter 1 for Dir /2 for file:1

No of subdirectories /files (for USER2):2 Enter Name of dir/file (under ROOT):A Enter 1 for Dir /2 for file:2

Enter Name of dir/file (under USER2):SUBDIR 2 Enter 1 for Dir /2 for file:1

No of subdirectories /files (for SUBDIR 2):2

Enter Name of dir/file (under SUBDIR2):PPL Enter 1 for Dir /2 for file:1

No of subdirectories /files (for PPL):2 Enter Name of dir/file (under PPL):B Enter 1 for Dir /2 for file:2

Enter Name of dir/file (under PPL):C Enter 1 for Dir /2 for file:2

Enter Name of dir/file (under SUBDIR):AI Enter 1 for Dir /2 for file:1

No of subdirectories /files (for AI): 2 Enter Name of dir/file (under AI):D Enter 1 for Dir /2 for file:2

Enter Name of dir/file (under AI):E Enter 1 for Dir /2 for file:2

# GENERAL GRAPH DIRECTORY:

When we add links to an existing tree structured directory, the tree structure is destroyed, resulting in a simple graph structure. This structure is used to traversing is easy and file sharing also possible.

# SOURCE CODE:

#include<stdio.h> #include<conio.h> #include<graphics.h> #include<string.h> struct tree\_element

{

char name[20];

int x,y,ftype,lx,rx,nc,level; struct tree\_element \*link[5];

};

typedef struct tree\_element node; typedef struct

{

char from[20]; char to[20];

}link;

link L[10]; int nofl; node \* root; void main()

{

int gd=DETECT,gm; root=NULL; clrscr(); create(&root,0,"root",0,639,320); read\_links();

clrscr(); initgraph(&gd,&gm,"c:\\tc\\BGI"); draw\_link\_lines(); display(root); getch(); closegraph();

}

read\_links()

{

int i;

printf("how many links"); scanf("%d",&nofl); for(i=0;i<nofl;i++)

{

printf("File/dir:");

fflush(stdin); gets(L[i].from); printf("user name:"); fflush(stdin); gets(L[i].to);

}

}

draw\_link\_lines()

{

int i,x1,y1,x2,y2; for(i=0;i<nofl;i++)

{

search(root,L[i].from,&x1,&y1); search(root,L[i].to,&x2,&y2); setcolor(LIGHTGREEN); setlinestyle(3,0,1); line(x1,y1,x2,y2); setcolor(YELLOW); setlinestyle(0,0,1);

}

}

search(node \*root,char \*s,int \*x,int \*y)

{

int i; if(root!=NULL)

{

if(strcmpi(root->name,s)==0)

{

\*x=root->x;

\*y=root->y; return;

}

else{

for(i=0;i<root->nc;i++) search(root->link[i],s,x,y);

}

}

}

create(node \*\*root,int lev,char \*dname,int lx,int rx,int x)

{

int i,gap; if(\*root==NULL)

{

(\*root)=(node \*)malloc(sizeof(node)); printf("enter name of dir/file(under %s):",dname); fflush(stdin);

gets((\*root)->name);

printf("enter 1 for dir/ 2 for file:"); scanf("%d",&(\*root)->ftype);

(\*root)->level=lev; (\*root)->y=50+lev\*50; (\*root)->x=x; (\*root)->lx=lx; (\*root)->rx=rx;

for(i=0;i<5;i++)

(\*root)->link[i]=NULL; if((\*root)->ftype==1)

{

printf("no of sub directories /files (for %s):",(\*root)->name); scanf("%d",&(\*root)->nc);

if((\*root)->nc==0) gap=rx-lx;

else**{**

gap=(rx-lx)/(\*root)->nc; for(i=0;i<(\*root)->nc;i++) create( & ( (\*root)->link[i] ) , lev+1 ,

(\*root)->name,lx+gap\*i,lx+gap\*i+gap,lx+gap\*i+gap/2);

}

else

(\*root)->nc=0;

}

}

/\* displays the constructed tree in graphics mode \*/ display(node \*root)

{

int i; settextstyle(2,0,4); settextjustify(1,1); setfillstyle(1,BLUE); setcolor(14); if(root

!=NULL)

{

for(i=0;i<root->nc;i++)

{

line(root->x,root->y,root->link[i]->x,root->link[i]->y);

}

if(root->ftype==1) bar3d(root->x-20,root->y-10,root-

>x+20,root->y+10,0,0);

else

fillellipse(root->x,root->y,20,20); outtextxy(root->x,root->y,root-

>name); for(i=0;i<root->nc;i++)

{

display(root->link[i]);

}}}

# OUTPUT:

Enter Name of dir/file (under root): ROOT Enter 1 for Dir / 2 For File : 1

No of subdirectories / files (for ROOT) :2 Enter Name of dir/file (under ROOT): USER 1 Enter 1 for Dir /2 for file:1

No of subdirectories /files (for USER 1): 2 Enter Name of dir/file (under USER1): VB Enter 1 for Dir /2 for file:1

No of subdirectories /files (for VB): 2 Enter Name of dir/file (under VB): A Enter 1 for Dir /2 for file:2

Enter Name of dir/file (under VB): B Enter 1 for Dir /2 for file:2

Enter Name of dir/file (under USER1): C Enter 1 for Dir /2 for file:2

Enter Name of dir/file (under ROOT): USER2 Enter 1 for Dir /2 for file:1

No of subdirectories /files (for USER2): 1 Enter Name of dir/file (under USER2):JAVA Enter 1 for Dir /2 for file:1

No of subdirectories /files (for JAVA):2 Enter Name of dir/file (under JAVA):D Enter 1 for Dir /2 for file:2

Enter Name of dir/file (under JAVA):HTML Enter 1 for Dir /2 for file:1

No of subdirectories /files (for HTML):0 How many links:2

File/Dir: B

User Name: USER 2 File/Dir: HTML User Name: USER1

# EXPERIMENT NO 7:

**DEAD LOCK AVOIDANCE**

**AIM:** Simulate bankers algorithm for Dead Lock Avoidance (Banker‘s Algorithm)

# DESCRIPTION:

Deadlock is a situation where in two or more competing actions are waiting f or the other to finish, and thus neither ever does. When a new process enters a system, it must declare the maximum number of instances of each resource type it needed. This number may exceed the total number of resources in the system. When the user request a set of resources, the system must determine whether the allocation of each resources will leave the system in safe state. If it will the resources are allocation; otherwise the process must wait until some other process release the resources.

Data structures

n-Number of process, m-number of resource types.

Available: Available[j]=k, k – instance of resource type Rj is available. Max: If max[i, j]=k, Pi may request at most k instances resource Rj.

Allocation: If Allocation [i, j]=k, Pi allocated to k instances of resource Rj Need: If Need[I, j]=k, Pi may need k more instances of resource type Rj, Need[I, j]=Max[I, j]- Allocation[I, j]; *Safety Algorithm*

1. Work and Finish be the vector of length m and n respectively, Work=Available and Finish[i] =False.
2. Find an i such that both
3. Finish[i] =False Need<=Work If no such I exists go to step 4.
4. work= work + Allocation, Finish[i] =True;
5. if Finish[1]=True for all I, then the system is in safe state. Resource request algorithm

Let Request i be request vector for the process Pi, If request i=[j]=k, then process Pi wants k instances of resource type Rj.

1. if Request<=Need I go to step 2. Otherwise raise an error condition.
2. if Request<=Available go to step 3. Otherwise Pi must since the resources are available.
3. Have the system pretend to have allocated the requested resources to process Pi by modifying the state as follows;

Available=Available-Request I; Allocation I

=Allocation +Request I; Need i=Need i- Request I;

If the resulting resource allocation state is safe, the transaction is completed and process Pi is allocated its resources. However if the state is unsafe, the Pi must wait for Request i and the old resource-allocation state is restored.

# ALGORITHM:

1. Start the program.
2. Get the values of resources and processes.
3. Get the avail value.
4. After allocation find the need value.
5. Check whether its possible to allocate.
6. If it is possible then the system is in safestate.
7. Else system is not in safety state.
8. If the new request comes then check that the system is in safety.
9. or not if we allow the request.
10. stop the program.
11. End

# SOURCE CODE :

#include<stdio.h> #include<conio.h> #include<string.h>

void main()

{

int alloc[10][10],max[10][10]; int avail[10],work[10],total[10]; int i,j,k,n,need[10][10];

int m;

int count=0,c=0; char finish[10]; clrscr(); printf("Enter the no. of processes and resources:"); scanf("%d%d",&n,&m);

for(i=0;i<=n;i++) finish[i]='n'; printf("Enter the claim matrix:\n"); for(i=0;i<n;i++)

for(j=0;j<m;j++) scanf("%d",&max[i][j]);

printf("Enter the allocation matrix:\n"); for(i=0;i<n;i++)

for(j=0;j<m;j++) scanf("%d",&alloc[i][j]); printf("Resource vector:"); for(i=0;i<m;i++) scanf("%d",&total[i]); for(i=0;i<m;i++) avail[i]=0; for(i=0;i<n;i++) for(j=0;j<m;j++) avail[j]+=alloc[i][j];

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

work[i]=avail[i]; for(j=0;j<m;j++) work[j]=total[j]-work[j]; for(i=0;i<n;i++) for(j=0;j<m;j++) need[i][j]=max[i][j]-alloc[i][j]; A:for(i=0;i<n;i++)

{ c=0;

for(j=0;j<m;j++) if((need[i][j]<=work[j])&&(finish[i]=='n')) c++;

if(c==m)

{

printf("All the resources can be allocated to Process %d", i+1); printf("\n\nAvailable resources are:"); for(k=0;k<m;k++)

{

work[k]+=alloc[i][k];

printf("%4d",work[k]);

}

printf("\n");

finish[i]='y';

printf("\nProcess %d executed?:%c \n",i+1,finish[i]); count++;

}}

if(count!=n) goto A; else

printf("\n System is in safe mode"); printf("\n The given state is safe state"); getch();

}

# OUTPUT

Enter the no. of processes and resources: 4 3 Enter the claim matrix: 3 2 2

6 1 3

3 1 4

4 2 2

Enter the allocation matrix:

1 0 0

6 1 2

2 1 1

0 0 2

Resource vector:9 3 6

All the resources can be allocated to Process 2 Available resources are: 6 2 3 Process 2 executed?:y

All the resources can be allocated to Process 3 Avaiable resources are: 8 3 4 Prcess 3 executed?:y

All the resources can be allocated to Process 4 Avilable resources are: 8 3 6 Prcess 4 executed?:y

All the resources can be allocated to Process 1 Aailable resources are: 9 3 6 Prcess 1 executed?:y System is in safe mode

The given state is safe state

# EXPERIMENT NO 8:

**DEAD LOCKPREVENTION**

**AIM:** To implement deadlock prevention technique

# Banker‘s Algorithm:

When a new process enters a system, it must declare the maximum number of instances of each resource type it needed. This number may exceed the total number of resources in the system. When the user request a set of resources, the system must determine whether the allocation of each resources will leave the system in safe state. If it will the resources are allocation; otherwise the process must wait until some other process release the resources.

# DESCRIPTION:

Data structures

n-Number of process, m-number of resource types.

Available: Available[j]=k, k – instance of resource type Rj is available. Max: If max[i, j]=k, Pi may request at most k instances resource Rj.

Allocation: If Allocation [i, j]=k, Pi allocated to k instances of resource Rj Need: If Need[I, j]=k, Pi may need k more instances of resource type Rj,

Need[I, j]=Max[I, j]-Allocation[I, j];

*Safety Algorithm*

Work and Finish be the vector of length m and n respectively, Work=Available and Finish[i] =False. Find an i such that both

Finish[i] =False Need<=Work If no such I exists go to step 4.

work=work+Allocation, Finish[i] =True;

if Finish[1]=True for all I, then the system is in safe state

# ALGORITHM:

1. Start the program.
2. Get the values of resources and processes.
3. Get the avail value.
4. After allocation find the need value.
5. Check whether its possible to allocate.
6. If it is possible then the system is in safestate.
7. Else system is not in safety state
8. Stop the process.

# SOURCE CODE :

#include<stdio.h> #include<conio.h> void main()

{

char job[10][10];

int time[10],avail,tem[10],temp[10]; int safe[10]; int ind=1,i,j,q,n,t; clrscr();

printf("Enter no of jobs: "); scanf("%d",&n); for(i=0;i<n;i++)

{

printf("Enter name and time: "); scanf("%s%d",&job[i],&time[i]);

}

printf("Enter the available resources:"); scanf("%d",&avail); for(i=0;i<n;i++)

{

temp[i]=time[i]; tem[i]=i;

}

for(i=0;i<n;i++) for(j=i+1;j<n;j++)

{

if(temp[i]>temp[j])

{

t=temp[i]; temp[i]=temp[j]; temp[j]=t; t=tem[i]; tem[i]=tem[j]; tem[j]=t;

}

}

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

{

q=tem[i]; if(time[q]<=avail)

{

safe[ind]=tem[i]; avail=avail-tem[q];

//printf("%s",job[safe[ind]]); ind++;

}

else

{

printf("No safe sequence\n");

}

}

printf("Safe sequence is:"); for(i=1;i<ind; i++) printf(" %s %d\n",job[safe[i]],time[safe[i]]); getch();

}

# OUTPUT:

Enter no of jobs:4

Enter name and time: A 1 Enter name and time: B 4 Enter name and time: C 2 Enter name and time: D 3

Enter the available resources: 20 Safe sequence is: A 1, C 2, D 3, B 4.

# EXPERIMENT NO.9

**PAGE REPLACEMENT ALGORITHMS**

1. **FIRST IN FIRST OUT:**

**AIM:** To implement FIFO page replacement technique.

# DESCRIPTION:

* The FIFO page-replacement algorithm is easy to understand and program. However, its performance is not always good.
* On the one hand, the page replaced may be an initialization module that was useda long time ago and is no longer needed.
* On the other hand, it could contain a heavily used variable that was initialized early and is in constant use.

# ALGORITHM:

1. Start the process
2. Read number of pages n
3. Read number of pages no
4. Read page numbers into an array a[i]
5. Initialize avail[i]=0 .to check page hit
6. Replace the page with circular queue, while re-placing check page availability in the frame Place avail[i]=1 if page is placed in theframe Count page faults
7. Print the results.
8. Stop the process.

# SOURCE CODE :

#include<stdio.h> #include<conio.h> int fr[3];

void main()

{

void display();

int i,j,page[12]={2,3,2,1,5,2,4,5,3,2,5,2};

int flag1=0,flag2=0,pf=0,frsize=3,top=0; clrscr(); for(i=0;i<3;i++)

{

fr[i]=-1;

}

for(j=0;j<12;j++)

{

flag1=0; flag2=0; for(i=0;i<12;i++)

{

if(fr[i]==page[j])

{

flag1=1; flag2=1; break;

}

}

if(flag1==0)

{

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

{

if(fr[i]==-1)

{

fr[i]=page[j]; flag2=1; break;

}

}

}

if(flag2==0)

{

fr[top]=page[j]; top++; pf++;

if(top>=frsize) top=0;

}

display();

}

printf("Number of page faults : %d ",pf); getch();

}

void display()

{

int i; printf("**\n**"); for(i=0;i<3;i++) printf("%d**\t**",fr[i]);

}

# OUTPUT:

2 -1 -1

2 3 -1

2 3 -1

2 3 1

5 3 1

5 2 1

5 2 4

5 2 4

3 2 4

3 2 4

3 5 4

3 5 2

Number of page faults: 6

# LEAST RECENTLY USED

**AIM:** To implement LRU page replacement technique.

# ALGORITHM:

1. Start the process
2. Declare the size
3. Get the number of pages to be inserted
4. Get the value
5. Declare counter and stack
6. Select the least recently used page by counter value
7. Stack them according the selection.
8. Display the values
9. Stop the process

# SOURCE CODE :

#include<stdio.h> #include<conio.h> int fr[3]; void main()

{

void display();

int p[12]={2,3,2,1,5,2,4,5,3,2,5,2},i,j,fs[3];

int index,k,l,flag1=0,flag2=0,pf=0,frsize=3; clrscr(); for(i=0;i<3;i++)

{

fr[i]=-1;

}

for(j=0;j<12;j++)

{

flag1=0,flag2=0; for(i=0;i<3;i++)

{

if(fr[i]==p[j])

{

flag1=1; flag2=1; break;

}

}

if(flag1==0)

{

for(i=0;i<3;i++)

{

if(fr[i]==-1)

{

fr[i]=p[j]; flag2=1; break;

}

}

}

if(flag2==0)

{

for(i=0;i<3;i++) fs[i]=0;

for(k=j-1,l=1;l<=frsize-1;l++,k--)

{

for(i=0;i<3;i++)

{

if(fr[i]==p[k]) fs[i]=1;

}

}

for(i=0;i<3;i++)

{

if(fs[i]==0) index=i;

}

fr[index]=p[j]; pf++;

}

display();

}

printf("**\n** no of page faults :%d",pf); getch();

}

void display()

{

int i; printf("**\n**");

for(i=0;i<3;i++) printf("**\t**%d",fr[i]);

}

# OUTPUT:

2 -1 -1

2 3 -1

2 3 -1

2 3 1

2 5 1

2 5 1

2 5 4

2 5 4

3 5 4

3 5 2

3 5 2

3 5 2

no of page faults : 4

# LFU: LEAST FREQUENTLY USED

**AIM:** To implement LFU page replacement technique.

# ALGORTHIM:

1. Start Program
2. Read Number Of Pages And Frames
3. Read Each Page Value
4. Search For Page In The Frames
5. If Not Available Allocate Free Frame
6. If No Frames Is Free Repalce The Page With The Page That Is LeastlyUsed
7. Print Page Number Of Page Faults
8. Stop process.

# SOURCE CODE :

/\* Program to simulate optimal page replacement \*/ #include<stdio.h>

#include<conio.h> int fr[3], n, m; void display(); void main()

{

int i,j,page[20],fs[10];

int max,found=0,lg[3],index,k,l,flag1=0,flag2=0,pf=0; float pr; clrscr();

printf("Enter length of the reference string: "); scanf("%d",&n);

printf("Enter the reference string: "); for(i=0;i<n;i++) scanf("%d",&page[i]);

printf("Enter no of frames: "); scanf("%d",&m); for(i=0;i<m;i++)

fr[i]=-1; pf=m;

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

{

flag1=0; flag2=0; for(i=0;i<m;i++)

{

if(fr[i]==page[j])

{

flag1=1; flag2=1; break;

}

}

if(flag1==0)

{

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

{

if(fr[i]==-1)

{

fr[i]=page[j]; flag2=1; break;

}

}

}

if(flag2==0)

{

for(i=0;i<m;i++) lg[i]=0; for(i=0;i<m;i++)

{

for(k=j+1;k<=n;k++)

{

if(fr[i]==page[k])

{

lg[i]=k-j; break;

}

}

}

found=0; for(i=0;i<m;i++)

{

if(lg[i]==0)

{

index=i; found = 1;

break;

}

}

if(found==0)

{

max=lg[0]; index=0; for(i=0;i<m;i++)

{

if(max<lg[i])

{

max=lg[i]; index=i;

}

}

}

fr[index]=page[j]; pf++;

}

display();

}

printf("Number of page faults : %d\n", pf); pr=(float)pf/n\*100;

printf("Page fault rate = %f \n", pr); getch();

}

void display()

{

int i; for(i=0;i<m;i++) printf("%d\t",fr[i]); printf("\n");

}

# OUTPUT:

Enter length of the reference string: 12

Enter the reference string: 1 2 3 4 1 2 5 1 2 3 4 5 Enter no of frames: 3

1 -1 -1

1 2 -1

1 2 3

1 2 4

1 2 4

1 2 4

1 2 5

1 2 5

1 2 5

3 2 5

4 2 5

4 2 5

Number of page faults : 7 Page fault rate = 58.333332

# EXPERIMENT NO 10: PAGING

**AIM:** Simulate Paging technique for memory management.

# ALGORITHM:

Step 1: Read all the necessary input from the keyboard.

Step 2: Pages - Logical memory is broken into fixed - sized blocks. Step 3: Frames – Physical memory is broken into fixed – sized blocks. Step 4: Calculate the physical address using the following

Physical address = ( Frame number \* Frame size ) + offset Step 5: Display the physical address.

Step 6: Stop the process.

# SOURCE CODE :

/\* Memory Allocation with Paging Technique \*/ #include <stdio.h>

#include <conio.h> struct pstruct

{

int fno; int pbit;

}ptable[10];

int pmsize,lmsize,psize,frame,page,ftable[20],frameno; void info()

{

printf("\n\nMEMORY MANAGEMENT USING PAGING\n\n"); printf("\n\nEnter the Size of Physical memory: "); scanf("%d",&pmsize);

printf("\n\nEnter the size of Logical memory: "); scanf("%d",&lmsize);

printf("\n\nEnter the partition size: "); scanf("%d",&psize);

frame = (int) pmsize/psize; page = (int) lmsize/psize;

printf("\nThe physical memory is divided into %d no.of frames\n",frame); printf("\nThe Logical memory is divided into %d no.of pages",page);

}

void assign()

{

int i;

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

{

ptable[i].fno = -1;

ptable[i].pbit= -1;

}

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

ftable[i] = 32555; for (i=0;i<page;i++)

{

printf("\n\nEnter the Frame number where page %d must be placed: ",i); scanf("%d",&frameno);

ftable[frameno] = i; if(ptable[i].pbit == -1)

{

ptable[i].fno = frameno; ptable[i].pbit = 1;

}

}

getch();

clrscr();

printf("\n\nPAGE TABLE\n\n"); printf("PageAddress FrameNo. PresenceBit\n\n"); for (i=0;i<page;i++)

printf("%d\t\t%d\t\t%d\n",i,ptable[i].fno,ptable[i].pbit); printf("\n\n\n\tFRAME TABLE\n\n");

printf("FrameAddress PageNo\n\n"); for(i=0;i<frame;i++)

printf("%d\t\t%d\n",i,ftable[i]);

}

void cphyaddr()

{

int laddr,paddr,disp,phyaddr,baddr; getch();

clrscr();

printf("\n\n\n\tProcess to create the Physical Address\n\n"); printf("\nEnter the Base Address: ");

scanf("%d",&baddr);

printf("\nEnter theLogical Address: "); scanf("%d",&laddr);

paddr = laddr / psize; disp = laddr % psize;

if(ptable[paddr].pbit == 1 )

phyaddr = baddr + (ptable[paddr].fno\*psize) + disp;

printf("\nThe Physical Address where the instruction present: %d",phyaddr);

}

void main()

{

clrscr();

info();

assign(); cphyaddr(); getch();

}

# OUTPUT:

MEMORY MANAGEMENT USING PAGING

Enter the Size of Physical memory: 16 Enter the size of Logical memory: 8 Enter the partition size: 2

The physical memory is divided into 8 no.of frames The Logical memory is divided into 4 no.of pages

Enter the Frame number where page 0 must be placed: 5 Enter the Frame number where page 1 must be placed: 6 Enter the Frame number where page 2 must be placed: 7 Enter the Frame number where page 3 must be placed: 2 PAGE TABLE

|  |  |  |
| --- | --- | --- |
| PageAddress FrameNo. | | PresenceBit |
| 0 | 5 | 1 |
| 1 | 6 | 1 |
| 2 | 7 | 1 |
| 3  FRAME TABLE  FrameAddress | 2 | 1 |
| PageNo |
| 0 | | 32555 |
| 1 | | 32555 |
| 2 | | 3 |
| 3 | | 32555 |
| 4 | | 32555 |
| 5 | | 0 |
| 6 | | 1 |
| 7 | | 2 |

Process to create the Physical Address Enter the Base Address: 1000 Enter theLogical Address: 3

The Physical Address where the instruction present: 1013

# EXPERIMENT NO 11:

**AIM:** Simulate disk scheduling algorithms- Scan,C-Scan.

# SCAN DISK SCHEDULING ALGORITHM:

**DESCRIPTION:**

It is also called as Elevator Algorithm. In this algorithm, the disk arm moves into a particular direction till the end, satisfying all the requests coming in its path,and then it turns backand moves in the reverse direction satisfying requests coming in its path.

# SOURCE CODE:

#include<stdio.h> main()

{

int t[20], d[20], h, i, j, n, temp, k, atr[20], tot, p, sum=0; clrscr();

printf("enter the no of tracks to be traveresed"); scanf("%d'",&n);

printf("enter the position of head"); scanf("%d",&h);

t[0]=0;t[1]=h;

printf("enter the tracks"); for(i=2;i<n+2;i++) scanf("%d",&t[i]); for(i=0;i<n+2;i++)

{

for(j=0;j<(n+2)-i-1;j++)

{if(t[j]>t[j+1])

{

temp=t[j]; t[j]=t[j+1]; t[j+1]=temp;

}}}

for(i=0;i<n+2;i++) if(t[i]==h)

j=i;k=i; p=0;

while(t[j]!=0)

{

atr[p]=t[j]; j--;

p++;

}

atr[p]=t[j]; for(p=k+1;p<n+2;p++,k++) atr[p]=t[k+1]; for(j=0;j<n+1;j++)

{

if(atr[j]>atr[j+1])

d[j]=atr[j]-atr[j+1]; else

d[j]=atr[j+1]-atr[j]; sum+=d[j];

}

printf("\nAverage header movements:%f",(float)sum/n); getch();

}

}

# INPUT

Enter no.of tracks:9

Enter track position:55 58 60 70 18 90 15 01 60 184

|  |  |
| --- | --- |
| **OUTPUT** |  |
| Tracks traversed | Difference between tracks |
| 150 | 50 |
| 160 | 10 |
| 184 | 24 |
| 90 | 94 |
| 70 | 20 |
| 60 | 10 |
| 58 | 2 |
| 55 | 3 |
| 18 | 37 |

Average header movements: 27.77

# C-SCAN DISK SCHEDULING ALGORITHM

**Description:**

In C-SCAN algorithm, the arm of the disk moves in a particular direction servicing requests until it reaches the last cylinder, then it jumps to the last cylinder of the opposite direction without servicing any request then it turns back and start moving in that direction servicing the remaining requests.

# SOURCE CODE:

#include<stdio.h> main()

{

int t[20], d[20], h, i, j, n, temp, k, atr[20], tot, p, sum=0; clrscr();

printf("enter the no of tracks to be traveresed"); scanf("%d'",&n);

printf("enter the position of head"); scanf("%d",&h);

t[0]=0;t[1]=h;

printf("enter total tracks"); scanf("%d",&tot);

t[2]=tot-1;

printf("enter the tracks"); for(i=3;i<=n+2;i++) scanf("%d",&t[i]); for(i=0;i<=n+2;i++) for(j=0;j<=(n+2)-i-1;j++)

if(t[j]>t[j+1])

{

temp=t[j]; t[j]=t[j+1]; t[j+1]=temp;

}

for(i=0;i<=n+2;i++) if(t[i]==h) j=i;break;

p=0;

while(t[j]!=tot-1)

{

atr[p]=t[j]; j++;

p++;

}

atr[p]=t[j]; p++;

i=0;

while(p!=(n+3) && t[i]!=t[h])

{

atr[p]=t[i]; i++;

p++;

}

for(j=0;j<n+2;j++)

{

if(atr[j]>atr[j+1])

d[j]=atr[j]-atr[j+1]; else

d[j]=atr[j+1]-atr[j]; sum+=d[j]

}

printf("total header movements%d",sum); printf("avg is %f",(float)sum/n);

getch();

}

# INPUT

Enter the track position : 55 58 60 70 18 90 15 01 60 184 Enter starting position : 100

# OUTPUT

Tracks traversed Difference Between tracks

|  |  |
| --- | --- |
| 150 | 50 |
| 160 | 10 |
| 184 | 24 |
| 18 | 240 |
| 55 | 37 |
| 58 | 3 |
| 60 | 2 |
| 70 | 10 |
| 90 | 20 |

Average seek time : 35.7777779