**JAWAHARLA**

**L NEHRU TECHNOLOGICAL UNIVERSITY ANANTAPUR**

**B.Tech. III – I Sem. L C**

**3 2**

**PART-A (15A05510) OPERATING SYSTEM LAB**

1. Simulate the following CPU scheduling algorithms
   1. Round Robin b) SJF c) FCFS d) Priority
2. Simulate all file allocation strategies
   1. Sequential b) Indexed c) Linked
3. Simulate MVT and MFT
4. Simulate all File Organization Techniques
   1. Single level directory b) Two level c) Hierarchical d) DAG
5. Simulate Bankers Algorithm for Dead Lock Avoidance
6. Simulate Bankers Algorithm for Dead Lock Prevention
7. Simulate all page replacement algorithms
   1. FIFO b) LRU c) LFU etc
8. Simulate Paging Technique of memory management
9. Control the number of ports opened by the operating system with

a) Semaphore b) monitors

1. Simulate how parent and child processes use shared memory and address space
2. Simulate sleeping barber problem
3. Simulate dining philosopher‘s problem
4. Simulate producer and consumer problem using threads (use java)
5. Simulate little‘s formula to predict next burst time of a process for SJF scheduling algorithm.
6. Develop a code to detect a cycle in wait-for graph
7. Develop a code to convert virtual address to physical address
8. Simulate how operating system allocates frame to process
9. Simulate the prediction of deadlock in operating system when all the processes announce their resource requirement in advance.

**Reference Books:**

1. **Operating System Concepts, Abraham Silberschatz, Peter B. Galvin, Greg Gagne, Eighth edition, John Wiley.**
2. **Operating Systems: Internals and Design Principles, Stallings, Sixth Edition– 2009,Pearson Education**
3. **Modern Operating Systems, Andrew S Tanenbaum, Second Edition, PHI.**
4. **Operating Systems, S.Haldar, A.A.Aravind, Pearson Education.**
5. **Principles of Operating Systems, B.L.Stuart, Cengage learning, India,Edition.2013-2014**
6. **Operating Systems, A.S.Godbole, Second Edition, TMH.**
7. **An Introduction to Operating Systems, P.C.P. Bhatt, PHI.**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Course Title:** | **Operating Systems Lab** | | | | | | **Course Code:** | | **15A05510** |
| **Class & Sem:** | **III B. Tech -I Sem** | | | | | | **Regulations:** | | **R15** |
| **Course Structure:** | **Theory** | **Tutorial** | **Lab** | **Credits** | | **Core/Elective:** | | **Core** | |
| **-** | **-** | **3** | **2** | |
| **Instructor1:** | **G.K. Venkata Narasimha Reddy** | | | | **Instructor 2:** | | **S.L.Sailaja** | | |
| **Academic Year 2017-18** | | | | | | | | | |

**Course Details**

**1. Academic Calendar:**

|  |  |  |
| --- | --- | --- |
| I Spell of instructions | 03-07-2017 to 01-09-2017 | (09 weeks) |
| I Mid-term Examinations | 04-09-2017 to 11-09-2017 | (06 days) |
| II Spell of instructions | 12-09-2017 to 03-11-2017 | (08 weeks) |
| II Mid-term Examinations | 04-11-2017 to 10-11-2017 | (06 days) |
| Preparation and Practicals | 13-11-2017 to 18-11-2017 | (06 days) |
| End Examinations | 20-11-2017 to 02-12-2017 | (02 weeks) |

**2. Prerequisites:** Programming languages such as C,C++, Java.

**3. Course Description:** This lab deals with the development of design aspects of the operating system where the student will be able to solve various synchronization problems, process deadlocks, to write software routines or modules to implement memory management and file organization and allocation techniques.

**4. List of experiments/Tasks :**

1. Simulate the following CPU scheduling algorithms

a) Round Robin b) SJF c) FCFS d) Priority

2. Simulate all file allocation strategies

a) Sequential b) Indexed c) Linked

3. Simulate MVT and MFT

4. Simulate all File Organization Techniques

a) Single level directory b) Two level c) Hierarchical d) DAG

5. Simulate Bankers Algorithm for Dead Lock Avoidance

6. Simulate Bankers Algorithm for Dead Lock Prevention

7. Simulate all page replacement algorithms

a) FIFO b) LRU c) LFU Etc. …

8. Simulate Paging Technique of memory management

9. Control the number of ports opened by the operating system with

a) Semaphore b) monitors

10. Simulate how parent and child processes use shared memory and address space

11. Simulate sleeping barber problem

12. Simulate dining philosopher‘s problem

13. Simulate producer and consumer problem using threads (use java)

14. Simulate little‘s formula to predict next burst time of a process for SJF scheduling algorithm.

15. Develop a code to detect a cycle in wait-for graph

16. Develop a code to convert virtual address to physical address

17. Simulate how operating system allocates frame to process

18. Simulate the prediction of deadlock in operating system when all the processes announce their resource requirement in advance.

**5. Course Outcomes:**

On successful completion of this lab the students will be able to:

1. Implement CPU scheduling algorithms.

2. Implement algorithms for file allocation strategies & file organization techniques.

3. Implement Bankers algorithm for Deadlock avoidance and prevention.

4. Implement algorithms for Paging and Page Replacement techniques of memory management.

5. Implement the concepts of Inter Process Communication.

6. Simulate algorithms for allocation of frames to processes.

**6. Learning Outcomes:**

On successful completion of each experiment the students will be able to:

1. Develop code for CPU scheduling algorithms such as FCFS, Round Robin, SJF, Priority scheduling.

2. Develop code for file allocation strategies such as Sequential, Indexed, Linked allocation.

3. Develop code for MVT and MFT techniques.

4. Develop code for file organization techniques such as single/two-level/ hierarchical/ DAG directory structures.

5. Develop code for Banker's algorithm for Deadlock avoidance and deadlock prevention.

6. Develop code for page replacement algorithms such as FIFO,LRU,LFU.

7. Develop code for paging technique.

8. Develop code for controlling the ports opened by the operating system using semaphores and monitors.

9. Develop code for sharing the memory space for both parent and child processes.

10. Develop code for Sleeping barber problem.

11. Develop code for dining philosopher's problem.

12. Develop code for producer consumer problem.

13. Develop code to predict the next burst time of a process for SJF scheduling.

14. Develop code to detect a cycle in wait-for-graph.

15. Develop code for converting a virtual address to physical address.

16. Develop code for allocation of frames to a process.

17. Develop code to predict deadlocks when all the resources are specified in advance.

**7. Course Delivery:** The course will be delivered through conduct of practical classes

**Program No.: 1(a)**

**Problem Statement:** Write a program to simulate Round Robin CPU scheduling program.

**Source Code:**

#include<stdio.h>

struct robin

{

int b,t,w;

}r[10];

void main()

{

int n,q,i,j,s,wt,t1=0,t2=0,b[10];

float a1,a2;

printf(“enter no.of processes:”);

scanf(“%d”,&n);

s=n;

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

printf(“enter burst time of process%d:”,i+1);

scanf(“%d”,&r[i].b);

b[i]=r[i].b;

r[i].t=r[i].w=0;

}

printf(“enter time quantum:”);

scanf(“%d”,&q);

do

{

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

{

if(r[i].b==0)

continue;

if(r[i].b>q)

{

r[i].b-=q;

wt=q;

r[i].t+=q;

}

else

{

wt=r[i].b;

r[i].t+=r[i].b;

r[i].b=0;

s--;

}

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

if(j!=i)

if(r[j].b>0)

{

r[j].w+=wt;

r[j].t+=wt;

}

}

}while(s>0);

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

{

t1+=r[i].t;

t2+=r[i].w;

}

a1=(float)t1/n;

a2=(float)t2/n;

printf(“Jno. Burst-Time Turn around-time Waiting-Time\n”);

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

printf(“%d %10d %11d %17d\n”,i+1,b[i],r[i].t,r[i].w);

printf(“\nAverage turn around time=%f:”,a1);

printf(“\nAverage Waiting time=%f:”,a2);

}

**Output:**

Enter number of processes: 3

Enter bursttime of process1: 5

Enter bursttime of process1: 4

Enter bursttime of process1: 5

Enter the time quantum: 4

JNo Burst-time Turnaround time waiting time

1 5 13 8

2 4 8 4

3 5 14 9

Average turn around time = 11.66667

Average Waiting time = 7.00000

**Program No.: 1(b)**

**Problem Statement:**Write a program to simulate SJF CPU scheduling algorithm.

**Source Code:**

#include<stdio.h>

void main()

{

int b[10],t[10],w[10],s[10];

int n,i,j,temp,wait=0,ttat=0,twt=0;

float atat,awt;

printf("Enter no.of jobs:");

scanf("%d",&n);

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

{

printf("enter burst time of job %d:",i);

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

s[i]=i;

}

for(i=1;i<=n-1;i++)

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

if(b[i]>b[j+1])

{

temp=b[i];

b[i]=b[j+1];

b[j+1]=temp;

temp=s[i];

s[i]=s[j+1];

s[j+1]=temp;

}

w[1]=0;

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

{

t[i]=b[i]+w[i];

w[i+1]=t[i];

}

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

{

ttat=ttat+t[i];

twt=twt+w[i];

}

printf("\nJobno. Burst-Time Turnaroundtime Watintingtime");

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

printf("\n%3d%11d%12d%18d",s[i],b[i],t[i],w[i]);

printf("\n Total turnaround time:%d Total waiting time:%d", ttat,twt);

atat=(float) ttat/ (float) n;

awt=(float) twt/(float) n;

printf("\nAverage turnaround time=%f",atat);

printf("\nAverage waiting time=%f",awt);

}

**Output:**

Enter number of processes: 3

Enter burst time of process1: 24

Enter burst time of process1: 3

Enter burst time of process1: 3

JNo Burst-time Turnaround time waiting time

2 3 3 0

3 3 6 3

1 24 30 6

Total turnaround time :39

Total Waiting time: 9

Average turn around time = 13.0000

Average Waiting time = 3.00000

**Program No.: 1(c)**

**Problem Statement:**Write a program to simulate FCFS CPU scheduling algorithm.

**Source Code:**

#include<stdio.h>

void main()

{

int b[10],t[10],w[10];

int n,i,j,wait=0,ttat=0,twt=0;

float atat,awt;

printf("Enter no.of jobs:");

scanf("%d",&n);

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

{

printf("enter burst time of job %d:",i);

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

}

w[1]=0;

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

{

t[i]=b[i]+w[i];

w[i+1]=t[i];

}

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

{

ttat=ttat+t[i];

twt=twt+w[i];

}

printf("\nJobno. Burst-Time Turnaroundtime Waitingtime");

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

printf("\n%3d%11d%12d%18d",i,b[i],t[i],w[i]);

printf("\n Total turnaround time:%d Total waiting time:%d", ttat,twt);

atat=(float) ttat/ (float) n;

awt=(float) twt/(float) n;

printf("\nAverage turnaround time=%f",atat);

printf("\nAverage waiting time=%f",awt);

}

**Output:**

Enter number of processes: 3

Enter bursttime of process1: 24

Enter bursttime of process1: 3

Enter bursttime of process1: 3

JNo Burst-time Turnaround time waiting time

1 24 24 0

2 3 27 24

3 3 30 27

Total turnaround time: 81

Total Waiting time: 51

Average turn around time = 27.0000

Average Waiting time = 17.00000

**Program No.: 1(d)**

**Problem Statement:Write a program to simulate Priority based CPU scheduling algorithm.**

**Source Code:**

**#include<stdio.h>**

**void main()**

**{**

**int b[10],t[10],w[10],s[10],p[10];**

**int n,i,j,tp,tb,temp,wait=0,ttat=0,twt=0;**

**float atat,awt;**

**printf("Entger no.of jobs:");**

**scanf("%d",&n);**

**for(i=1;i<=n;i++)**

**{**

**printf("enter bust time and priority of job %d:",i);**

**scanf("%d%d",&b[i],&p[i]);**

**s[i]=i;**

**}**

**for(i=1;i<=n-1;i++)**

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

**if(p[i]>=p[j])**

**{**

**tp=p[i];**

**p[i]=p[j];**

**p[j]=tp;**

**if(p[i]==p[j] && s[i]<s[j])**

**continue;**

**tb=b[i];**

**b[i]=b[j];**

**b[j]=tb;**

**temp=s[i];**

**s[i]=s[j];**

**s[j]=temp**;

**}**

**w[1]=0;**

**for(i=1;i<=n;i++)**

**{**

**t[i]=b[i]+w[i];**

**w[i+1]=t[i];**

**}**

**for(i=1;i<=n;i++)**

**{**

**ttat=ttat+t[i];**

**twt=twt+w[i];**

**}**

**printf("\nJobno. Priority Burst-Time Turnaroundtime Watintingtime");**

**for(i=1;i<=n;i++)**

**printf("\n%3d%9d%11d%12d%18d",s[i],p[i],sb[i],t[i],w[i]);**

**printf("\n Total turnaround time:%d Total waiting time:%d", ttat,twt);**

**atat=(float) ttat/ (float) n;**

**awt=(float) twt/(float) n;**

**printf("\nAverage turnaround time=%f",atat);**

**printf("\nAverage waiting time=%f",awt);**

**}**

**Output:**

**Enter number of processes: 3**

**Enter burst time and priority of job1: 5 2**

**Enter burst time and priority of job2: 6 1**

**Enter burst time and priority of job3: 7 3**

**JNo Priority Burst-time Turnaround time waiting time**

**2 1 6 6 0**

**1 2 5 11 6**

**3 3 7 18 11**

**Total turnaround time: 35**

**Total Waiting time: 17**

**Average turnaround time = 11.66667**

**Average Waiting time = 5.66667**

**Program No.: 2(a)**

**Problem Statement:Write a program to simulate Sequential File Allocation Strategy.**

**Source Code:**

**#include<stdio.h>**

**struct fileTable**

**{**

**char name[20];**

**int sb, nob;**

**}ft[30];**

**void main()**

**{**

**int i, j, n;**

**char s[20];**

**printf("Enter no of files :");**

**scanf("%d",&n);**

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

**{**

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

**scanf("%s",ft[i].name);**

**printf("Enter starting block of file %d :",i+1);**

**scanf("%d",&ft[i].sb);**

**printf("Enter no of blocks in file %d :",i+1);**

**scanf("%d",&ft[i].nob);**

**}**

**printf("\nEnter the file name to be searched -- ");**

**scanf("%s",s);**

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

**if(strcmp(s, ft[i].name)==0)**

**break;**

**if(i==n)**

**printf("\nFile Not Found");**

**else**

**{**

**printf("\nFILE NAME START BLOCK NO OF BLOCKS BLOCKS OCCUPIED\n");**

**printf("\n%s\t\t%d\t\t%d\t",ft[i].name,ft[i].sb,ft[i].nob);**

**for(j=0;j<ft[i].nob;j++)**

**printf("%d, ",ft[i].sb+j);**

**}**

**}**

**Output:**

**Enter no of files :3**

**Enter file name 1 :A**

**Enter starting block of file 1 :85**

**Enter no of blocks in file 1 :6**

**Enter file name 2 :B**

**Enter starting block of file 2 :102**

**Enter no of blocks in file 2 :4**

**Enter file name 3 :C**

**Enter starting block of file 3 :60**

**Enter no of blocks in file 3 :4**

**Enter the file name to be searched -- B**

**FILE NAME START BLOCK NO OF BLOCKS BLOCKS OCCUPIED**

**B 102 4 102, 103, 104, 105,**

**Program No.: 2(b)**

**Problem Statement:Write a program to simulate Indexed File Allocation Strategy.**

**Source Code:**

**#include<stdio.h>**

**#include<string.h>**

**struct fileTable**

**{**

**char name[20];**

**int nob, blocks[30];**

**}ft[30];**

**void main()**

**{**

**int i, j, n;**

**char s[20];**

**printf("Enter no of files :");**

**scanf("%d",&n);**

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

**{**

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

**scanf("%s",ft[i].name);**

**printf("Enter no of blocks of this file: ",i+1);**

**scanf("%d",&ft[i].nob);**

**printf("Enter the blocks of the file: ");**

**for(j=0;j<ft[i].nob;j++)**

**scanf("%d",&ft[i].blocks[j]);**

**}**

**printf("Enter the file name to be searched: ");**

**scanf("%s",s);**

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

**if(strcmp(s, ft[i].name)==0)**

**break;**

**if(i==n)**

**printf("File is Not Found.\n");**

**else**

**{**

**printf("FILE\_NAME NO\_OF\_BLOCKS BLOCKS\_OCCUPIED\n");**

**printf("%-10s%-13d",ft[i].name,ft[i].nob);**

**for(j=0;j<ft[i].nob;j++)**

**printf("%d, ",ft[i].blocks[j]);**

**printf("\b\b.\n");**

**}**

**}**

**Output:**

**Enter no of files :2**

**Enter file name 1 :A**

**Enter no of blocks of this file: 4**

**Enter the blocks of the file: 11 12 13 14**

**Enter file name 2 :B**

**Enter no of blocks of this file: 5**

**Enter the blocks of the file: 15 16 17 18 19**

**Enter the file name to be searched: B**

**FILE\_NAME NO\_OF\_BLOCKS BLOCKS\_OCCUPIED**

**B 5 15, 16, 17, 18, 19.**

**Program No.: 2(c)**

**Problem Statement:Write a program to simulate Linked File Allocation Strategy.**

**Source Code:**

**#include<stdio.h>**

**#include<stdlib.h>**

**struct fileTable**

**{**

**char name[20];**

**int nob;**

**struct block \*sb;**

**}ft[30];**

**struct block**

**{**

**int bno;**

**struct block \*next;**

**};**

**void main()**

**{**

**int i, j, n;**

**char s[20];**

**struct block \*temp;**

**printf("Enter no of files :");**

**scanf("%d",&n);**

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

**{**

**printf("\nEnter file name** **%d :",i+1);**

**scanf("%s",ft[i].name);**

**printf("Enter no of blocks in file %d :",i+1);**

**scanf("%d",&ft[i].nob);**

**ft[i].sb=(struct block\*)malloc(sizeof(struct block));**

**temp = ft[i].sb; printf("Enter the blocks of the file :");**

**scanf("%d",&temp->bno); temp->next=NULL;**

**for(j=1;j<ft[i].nob;j++)**

**{**

**temp->next = (struct block\*)malloc(sizeof(struct block));**

**temp = temp->next; scanf("%d",&temp->bno);**

**}**

**temp->next = NULL;**

**}**

**printf("\nEnter the file name to be searched -- ");**

**scanf("%s",s);**

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

**if(strcmp(s, ft[i].name)==0)**

**break;**

**if(i==n)**

**printf("\nFile Not Found");**

**else**

**{**

**printf("\nFILE NAME NO OF BLOCKS BLOCKS OCCUPIED");**

**printf("\n %s\t\t%d\t",ft[i].name,ft[i].nob);**

**temp=ft[i].sb;**

**for(j=0;j<ft[i].nob;j++)**

**{**

**if(j+1 != ft[i].nob)**

**printf("%d ->",temp->bno);**

**else**

**printf("%d.",temp->bno) ;**

**temp = temp->next;**

**}}}**

**Output:**

**Enter no of files : 2**

**Enter file name 1 : A**

**Enter no of blocks in file 1 : 4**

**Enter the blocks of the file : 12 23 9 4**

**Enter file name 2 : G**

**Enter no of blocks in file 2 : 5**

**Enter the blocks of the file : 88 77 66 55 44**

**Enter the file name to be searched -- G**

**FILE NAME NO OF BLOCKS BLOCKS OCCUPIED**

**G 5 88 -> 77 -> 66 -> 55 -> 44**

**Program No.: 3(a)**

**Problem Statement:Write a program to simulate MVT.**

**Source Code:**

**#include<stdio.h>**

**main()**

**{**

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

**char ch = 'y';**

**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);**

**}**

**Output:**

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

**Enter memory required for process1 (in Bytes)--450**

**Memory is allocated for process1**

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

**Enter memory required for process2 (in Bytes)--250**

**Memory is allocated for process2**

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

**Enter memory required for process3 (in Bytes)--128**

**Memory is allocated for process3**

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

**Enter memory required for process4 (in Bytes)--100**

**Memory is allocated for process4**

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

**Total memory available --1000**

**PROCESS MEMORY ALLOCATED**

**1 450**

**2 250**

**3 128**

**4 100**

**Total Memory Allocated is 928**

**Total External Fragmentation is 72**

**Program No.: 3(b)**

**Problem Statement:Write a program to simulate MFT.**

**Source Code:**

**#include<stdio.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);**

**}**

**Output:**

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

**Enter the block size (in Bytes) --300**

**Enter the number of processes --5**

**Enter memory required for process1(in Bytes)--280**

**Enter memory required for process2(in Bytes)--185**

**Enter memory required for process3(in Bytes)--350**

**Enter memory required for process4(in Bytes)--300**

**Enter memory required for process5(in Bytes)--100**

**No. of Blocks available in memory-- 3**

**PROCESS MEMORY REQUIRED ALLOCATED INTERNAL FRAGMENTATION**

**1 280 YES 20**

**2 185 YES 115**

**3 350 NO --**

**4 300 YES 0**

**Memory is Full, Remaining Processes cannot be accommodated**

**Total Internal Fragmentation is 135**

**Total External Fragmentation is 100**

**Program No.: 4(a)**

**Problem Statement:Write a program to simulate Single level directory organization technique.**

**Source Code:**

**#include<stdio.h>**

**#include<string.h>**

**#include<stdlib.h>**

**struct**

**{**

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

**int fcnt;**

**}dir;**

**void main()**

**{**

**int i,ch,n;**

**char f[30];**

**dir.fcnt = 0;**

**printf("Enter name of directory: ");**

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

**printf("1. Create File\n2. Delete File\n3. Search File\n4. Display Files\n5. Exit\n");**

**while(1)**

**{**

**printf("Enter your choice: ");**

**scanf("%d",&ch);**

**switch(ch)**

**{**

**case 1: printf("Enter the name of the file: ");**

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

**++dir.fcnt;**

**break;**

**case 2: printf("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.\n",f);**

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

**break;**

**}**

**}**

**if(i==dir.fcnt)**

**printf("File %s not found.\n",f);**

**else**

**dir.fcnt--;**

**break;**

**case 3: printf("Enter the name of the file: ");**

**scanf("%s",f);**

**n=dir.fcnt;**

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

**{**

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

**{**

**printf("%s is found.\n", f);**

**break;**

**}**

**}**

**if(i==n)**

**printf("%s is not found.\n",f);**

**break;**

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

**printf("Directory Empty.\n");**

**else**

{

**printf("The Files are: ");**

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

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

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

**}**

**break;**

**default: exit(0);**

**}**

**}**

**}**

**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**

**Enter your choice: 1**

**Enter the name of the file: B**

**Enter your choice: 1**

**Enter the name of the file: C**

**Enter your choice: 4**

**The Files are: A B C**

**Enter your choice: 3**

**Enter the name of the file: c1**

**c1 is not found.**

**Enter your choice: 3**

**Enter the name of the file: A**

**A is found.**

**Enter your choice: 2**

**Enter the name of the file: A**

**File A is deleted.**

**Enter your choice: 4**

**The Files are: C B**

**Enter your choice: 5**

**Program No.: 4(b)**

**Problem Statement:Write a program to simulate Two level directory organization technique.**

**Source Code:**

**#include<stdio.h>**

**#include<string.h>**

**#include<stdlib.h>**

**struct**

**{**

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

**int fcnt;**

**}dir[10];**

**void main()**

**{**

**int i,ch,dcnt,k;**

**char f[30], d[30];**

**dcnt=0;**

**printf("1. Create Directory\n2. Create File\n3. Delete File\n");**

**printf("4. Search File\n5. Display\n6. Exit\n");**

**while(1)**

**{**

**printf("Enter your Choice: ");**

**scanf("%d",&ch);**

**switch(ch)**

**{**

**case 1: printf("Enter name of directory: ");**

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

**dir[dcnt].fcnt=0;**

**dcnt++;**

**printf("Directory created\n");**

**break;**

**case 2: printf("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\n");**

**break;**

**}**

**if(i==dcnt)**

**printf("Directory \"%s\" is not found\n",d);**

**break;**

**case 3: printf("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",f);**

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

**{**

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

**{**

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

**dir[i].fcnt--;**

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

**goto jmp;**

**}**

**}**

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

**goto jmp;**

**}**

**}**

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

**jmp : break;**

**case 4: printf("Enter 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.\n",f);**

**goto jmp1;**

**}**

**}**

**printf("File %s not found.\n",f);**

**goto jmp1;**

**}**

**}**

**printf("Directory %s is not found.\n",d);**

**jmp1:**

**break;**

**case 5: if(dcnt==0)**

**printf("No Directory's.\n");**

**else**

**{**

**printf("Directory\tFiles\n");**

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

**{**

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

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

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

**printf("\b\b.\n");**

**}**

**}**

**break;**

**default: exit(0);**

**}**

**}**

**}**

**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**

**Enter your Choice: 1**

**Enter name of directory: dir2**

**Directory created**

**Enter your Choice: 2**

**Enter name of the directory: dir1**

**Enter name of the file: f1**

**File created**

**Enter your Choice: 2**

**Enter name of the directory: dir1**

**Enter name of the file: f2**

**File created**

**Enter your Choice: 2**

**Enter name of the directory: dir2**

**Enter name of the file: f3**

**File created**

**Enter your Choice: 2**

**Enter name of the directory: dir2**

**Enter name of the file: f4**

**File created**

**Enter your Choice: 5**

**Directory Files**

**dir1 f1, f2.**

**dir2 f3, f4.**

**Enter your Choice: 4**

**Enter name of the directory: dir1**

**Enter the name of the file: f2**

**File f2 is found.**

**Enter your Choice: 3**

**Enter name of the directory: dir2**

**Enter name of the file: f4**

**File f4 is deleted**

**Enter your Choice: 5**

**Directory Files**

**dir1 f1, f2.**

**dir2 f3.**

**Enter your Choice: 6**

**Program No.: 4(b)**

**Problem Statement:***Write a program to simulate Two level directory organization technique.*

**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**

**Program No.: 5**

**Problem Statement:Write a program to simulate Bankers Algorithm for Dead Lock** **Avoidance.**

**Source Code:**

**#include<stdio.h>**

**#include<stdlib.h>**

**#define true 1**

**#define false 0**

**int avilable[10], allocation[10][10], max[10][10];**

**int need[10][10], work[10], finish[10], maxres[10], safe[10], req[10],m,n;**

**int find()**

**{**

**int i,j;**

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

**{**

**if(finish[i]==false)**

**{**

**for(j=0;j<m;j++)**

**if(need[i][j]>work[j])**

**break;**

**if(j==m)**

**{**

**finish[i]=true;**

**return i;**

**}**

**}**

**}**

**return -1;**

**}**

**void issafe()**

**{**

**int i,j,k=0,cnt=n;**

**for(j=0;j<m;j++)**

**work[j]=avilable[j];**

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

**finish[i]=false;**

**while(cnt>0)**

**{**

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

**{**

**i=find();**

**if(i==-1)**

**{**

**printf("The system is in unsafe state.\n");**

**return;**

**}**

**for(j=0;j<m;j++)**

**work[j]+=allocation[i][j];**

**safe[k++]=i;**

**cnt--;**

**}**

**}**

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

**if(finish[i]==false)**

**{**

**printf("\nThe system is in unsafe state ");**

**exit(0);**

*}*

*printf("\nThe system is in safe state,safe sequence:");*

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

*printf("P%d, ",safe[i]);*

*printf("\b\b.\n");*

*}*

*void main()*

*{*

*int i,j,sum;*

*char ch;*

*printf("Enter the number of processes & resources: ");*

*scanf("%d%d",&n,&m);*

*printf("Enter the maximum instance of resources: ");*

*for(j=0;j<m;j++)*

*{*

**scanf("%d",&maxres[j]);**

**avilable[j]=maxres[j];**

**}**

**printf("Enter the allocated matrix.\n");**

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

**{**

**for(j=0;j<m;j++)**

**scanf("%d",&allocation[i][j]);**

**}**

**printf("Enter the Max matrix.\n");**

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

**for(j=0;j<m;j++)**

**{**

**scanf("%d",&max[i][j]);**

**need[i][j]=max[i][j]-allocation[i][j];**

**}**

**printf("The Allocation Matrix as Follows.\n");**

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

**{**

**for(j=0;j<m;j++)**

**printf("%d ",need[i][j]);**

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

**}**

**for(j=0;j<m;j++)**

**{**

**sum=0;**

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

**sum+=allocation[i][j];**

**avilable[j]-=sum;**

**}**

**issafe();**

**}**

**Output:**

**Enter the maximum instance of resources: 10 5 7**

**Enter the allocated matrix.**

**0 1 0**

**2 0 0**

**3 0 2**

**2 1 1**

**0 0 2**

**Enter the Max matrix.**

**7 5 3**

**3 2 2**

**9 0 2**

**2 2 2**

**4 3 3**

**The Allocation Matrix as Follows.**

**7 4 3**

**1 2 2**

**6 0 0**

**0 1 1**

**4 3 1**

**The system is in safe state, safe sequence: P1, P3, P0, P2, P4.**

**Program No.: 6**

**Problem Statement:Write a program to simulate Bankers Algorithm for Dead Lock Prevention.**

**Source Code:**

**#include<stdio.h>**

**int max[10][10], alloc[10][10], need[10][10];**

**int avail[10],i,j,p,r,finish[10]={0},flag=0;**

**int main()**

**{**

**printf("\n\nSIMULATION OF DEADLOCK PREVENTION");**

**printf("Enter no. of processes, resources");**

**scanf("%d%d",&p,&r);printf("Enter allocation matrix");**

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

**for(j=0;j<r;j++)**

**scanf("%d",&alloc[i][j]);**

**printf("enter max matrix");**

**for(i=0;i<p;i++) /\*reading the maximum matrix and availale matrix\*/**

**for(j=0;j<r;j++)**

**scanf("%d",&max[i][j]);**

**printf("enter available matrix");**

**for(i=0;i<r;i++)**

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

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

**for(j=0;j<r;j++)**

**need[i][j]=max[i][j]-alloc[i][j];**

**fun(); /\*calling function\*/**

**if(flag==0)**

**{**

**if(finish[i]!=1)**

**{**

**printf("\n\n Failing :Mutual exclusion");**

**for(j=0;j<r;j++)**

**{ /\*checking for mutual exclusion\*/**

**if(avail[j]<need[i][j])**

**avail[j]=need[i][j];**

**}fun();**

**printf("\n By allocating required resources to process %d dead lock is prevented ",i);**

**printf("\n\n lack of preemption");**

**for(j=0;j<r;j++)**

**{**

**if(avail[j]<need[i][j])**

**avail[j]=need[i][j];**

**alloc[i][j]=0;**

**}**

**fun( );**

**printf("\n\n daed lock is prevented by allocating needed resources");**

**printf(" \n \n failing:Hold and Wait condition ");**

**for(j=0;j<r;j++)**

**{**

**if(avail[j]<need[i][j])**

**avail[j]=need[i][j];**

**}**

**fun( );**

**printf("\n AVOIDING ANY ONE OF THE CONDITION, U CAN PREVENT DEADLOCK");**

**}**

**}**

**getch( ); return 0;**

**}**

**fun()**

**{**

**while(1)**

**{**

**for(flag=0,i=0;i<p;i++)**

**{**

**if(finish[i]==0)**

**{**

**for(j=0;j<r;j++)**

**{**

**if(need[i][j]<=avail[j])**

**continue;**

**else**

**break;**

**}**

**if(j==r)**

**{**

**for(j=0;j<r;j++)**

**avail[j]+=alloc[i][j];**

**flag=1;**

**finish[i]=1;**

**}**

**}**

**}**

**if(flag==0)**

**break;**

**}return 0;**

**}**

**OUTPUT:**

**SIMULATION OF DEADLOCK PREVENTION**

**Enter no. of processes, resources 3, 2**

**Enter allocation matrix**

**2 4 5**

**3 4 5**

**Enter max matrix**

**4 3 4**

**5 6 1**

**Enter available matrix2**

**Failing: Mutual Exclusion**

**By allocating required resources to process dead is prevented**

**Lack of no preemption deadlock is prevented by allocating needed resources**

**Failing: Hold and Wait condition**

**Program No.: 7(a)**

**Problem Statement:**Write a program to simulate FIFO page replacement algorithm.

**Source Code:**

**#include<stdio.h>**

**void main()**

**{**

**int i, j, k, f, pf=0, count=0, rs[25], m[10], n;**

**printf("Enter the Number of Pages: ");**

**scanf("%d",&n);**

**printf("Enter %d Page Numbers: ",n);**

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

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

**printf("Enter the Frame Size: ");**

**scanf("%d",&f);**

**for(i=0;i<f;i++)**

**m[i]=-1;**

**printf("The FIFO Page Replacement Process as follows.\n");**

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

**{**

**for(k=0;k<f;k++)**

**{**

**if(m[k]==rs[i])**

**break;**

**}**

**if(k==f)**

**{**

**m[count++]=rs[i];**

**pf++;**

**}**

**for(j=0;j<f;j++)**

**printf("\t%d",m[j]);**

**if(k==f)**

**printf("\tPF No. %d",pf);**

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

**if(count==f)**

**count=0;**

**}**

**printf("The number of Page Faults using FIFO is %d",pf);**

**}**

**Output:**

**Enter the Number of Pages: 20**

**Enter 20 Page Numbers: 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1**

**Enter the Frame Size: 3**

**The FIFO Page Replacement Process as follows.**

**7 -1 -1 PF No. 1**

**7 0 -1 PF No. 2**

**7 0 1 PF No. 3**

**2 0 1 PF No. 4**

**2 0 1**

**2 3 1 PF No. 5**

**2 3 0 PF No. 6**

**4 3 0 PF No. 7**

**4 2 0 PF No. 8**

**4 2 3 PF No. 9**

**0 2 3 PF No. 10**

**0 2 3**

**0 2 3**

**0 1 3 PF No. 11**

**0 1 2 PF No. 12**

**0 1 2**

**0 1 2**

**7 1 2 PF No. 13**

**7 0 2 PF No. 14**

**7 0 1 PF No. 15**

**The number of Page Faults using FIFO is 15**

**Program No.: 7(b)**

**Problem Statement:Write a program to simulate LRU page replacement algorithm.**

**Source Code:**

**#include<stdio.h>**

**void main()**

**{**

**int i, j , k, min, rs[25], m[10], count[10], flag[25], n, f, pf=0, next=1;**

**printf("Enter the Number of Pages: ");**

**scanf("%d",&n);**

**printf("Enter Page Numbers: ");**

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

**{**

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

**flag[i]=0;**

**}**

**printf("Enter the Frame Size: ");**

**scanf("%d",&f);**

**for(i=0;i<f;i++)**

**{**

**count[i]=0;**

**m[i]=-1;**

**}**

**printf("The LRU Page Replacement process is as follows.\n");**

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

**{**

**for(j=0;j<f;j++)**

**{**

**if(m[j]==rs[i])**

**{**

**flag[i]=1;**

**count[j]=next;**

**next++;**

**}**

**}**

**if(flag[i]==0)**

**{**

**if(i<f)**

**{**

**m[i]=rs[i];**

**count[i]=next;**

**next++;**

**}**

**else**

**{**

**min=0;**

**for(j=1;j<f;j++)**

**if(count[min] > count[j])**

**min=j;**

**m[min]=rs[i];**

**count[min]=next;**

**next++;**

**}**

**pf++;**

**}**

**for(j=0;j<f;j++)**

**printf("%d\t", m[j]);**

**if(flag[i]==0)**

**printf("PF No. --> %d" , pf);**

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

**}**

**printf("The number of page faults using LRU is %d\n",pf);**

**}**

**Output:**

**Enter the Frame Size: 3**

**The LRU Page Replacement process is as follows.**

**7 -1 -1 PF No. --> 1**

**7 0 -1 PF No. --> 2**

**7 0 1 PF No. --> 3**

**2 0 1 PF No. --> 4**

**2 0 1**

**2 0 3 PF No. --> 5**

**2 0 3**

**4 0 3 PF No. --> 6**

**4 0 2 PF No. --> 7**

**4 3 2 PF No. --> 8**

**0 3 2 PF No. --> 9**

**0 3 2**

**0 3 2**

**1 3 2 PF No. --> 10**

**1 3 2**

**1 0 2 PF No. --> 11**

**1 0 2**

**1 0 7 PF No. --> 12**

**1 0 7**

**1 0 7**

**The number of page faults using LRU is 12**

**Program No.: 7(c)**

**Problem Statement:Write a program to simulate LFU page replacement algorithm.**

**Source Code:**

**#include<stdio.h>**

**void main()**

**{**

**int rs[50], i, j, k, m, f, cntr[20], a[20], min, pf=0;**

**printf("Enter number of pages: ");**

**scanf("%d",&m);**

**printf("Enter %d page references: ",m);**

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

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

**printf("Enter Frame Size: ");**

**scanf("%d",&f);**

**for(i=0;i<f;i++)**

**{**

**cntr[i]=0;**

**a[i]=-1;**

**}**

**printf("The LFU Page Replacement Process as Follows:\n");**

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

**{**

**for(j=0;j<f;j++)**

**if(rs[i]==a[j])**

**{**

**cntr[j]++;**

**break;**

**}**

**if(j==f)**

**{**

**min = 0;**

**for(k=1;k<f;k++)**

**if(cntr[k]<cntr[min])**

**min=k;**

**a[min]=rs[i];**

**cntr[min]=1;**

**pf++;**

**}**

**for(j=0;j<f;j++)**

**printf("\t%d",a[j]);**

**if(j==f)**

**printf("\tPF No --> %d\n",pf);**

**}**

**printf("Total number of page faults is %d.\n",pf);**

**}**

**Output:**

**Enter number of pages: 10**

**Enter 10 page references: 1 2 3 4 5 2 5 1 4 3**

**Enter Frame Size: 3**

**The LFU Page Replacement Process as Follows:**

**1 -1 -1 PF No --> 1**

**1 2 -1 PF No --> 2**

**1 2 3 PF No --> 3**

**4 2 3 PF No --> 4**

**5 2 3 PF No --> 5**

**5 2 3 PF No --> 5**

**5 2 3 PF No --> 5**

**5 2 1 PF No --> 6**

**5 2 4 PF No --> 7**

**5 2 3 PF No --> 8**

**Total number of page faults is 8.**

**Program No.: 8**

**Problem Statement:Write a program to simulate paging technique of memory management.**

**Source Code:**

#**include<stdio.h>**

**#include<stdlib.h>**

**#define max 20**

**void main()**

**{**

**int frag[max],loc=0,a[10],t[10],b[10],n;**

**int s=0,i,j,k,oc,big,flag[10], temp,m,g,ch=0,lowest=1000;**

**int static ab[max],ff[max];**

**printf("\nEnter the number of memory partitions: ");**

**scanf("%d",&m);**

**printf("\nEnter the partition size: ");**

**for(i=1;i<=m;i++)**

**{**

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

**t[i]=a[i];**

**}**

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

**flag[i]=0;**

**printf("\n\t Memory Partitions\n");**

**printf("\n\t\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");**

**for(i=1;i<=m;i++)**

**printf("\t\t%d----%d\n\n",i,a[i]);**

**printf("\nEnter no.of jobs: ");**

**scanf("%d",&n);**

**g=n;**

**printf("\nEnter jobs size (at end give digit -5)\n");**

**i=0;**

**while(s!=-5)**

**{**

**i++;**

**scanf("%d",&s);**

**b[i]=s;**

**}**

**printf("1.first fit\n2.worstfit\n3.bestfit\n4.exit\n");**

**do{**

**printf("\nEnter ur choice:");**

**scanf("%d",&ch);**

**switch(ch)**

**{**

**case 1:**

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

**flag[i]=0;**

**for(i=1;i<=m;i++)**

**{**

**a[i]= t[i];**

**}**

**for(j=1;j<=g;j++)**

**{**

**k=0;**

**for(i=1;i<=m;i++)**

**{**

**if((b[j]<=a[i])&&(flag[i]==0))**

**{**

**printf("job %d of size %d is fitted in %d Remaining capacity is %d \n \n",j,b[j],i,a[i]-b[j]);**

**flag[i]=1;**

**a[i]-=b[j];**

**k=1;**

**break;**

**}**

**}**

**if(k==0)**

**printf("\n insufficient memory for job %d",j);**

**}**

**break;**

**case 2:**

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

**flag[i]=0;**

**for(i=1;i<=m;i++)**

**{**

**a[i]= t[i];**

**}**

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

**{**

**big=a[1];**

**loc=1;**

**for(i=2;i<=m;i++)**

**{**

**if((a[i]>big)&&(flag[i]==0))**

**{**

**big=a[i];**

**loc=i;**

**}**

**}**

**if(b[j]>a[loc])**

**printf("\njob %d is not fitted\n",j,i);**

**else**

**if(flag[loc]==0)**

**{**

**printf("\nJob %d is fitted in %d (%d) Remaining capacity is %d \n",j,loc,a[loc],a[loc]-b[j]);**

**a[loc]-=b[j];**

**flag[loc]=1;**

**}**

**}**

**break;**

**case 3:**

**for(i=1;i<=m;i++)**

**{**

**a[i]= t[i];**

**}**

**for(i=1;i<=n;i++)**

**{**

**for(j=1;j<=m;j++)**

**{**

**if(ab[j]!=1)**

**{**

**temp=a[j]-b[i];**

**if(temp>=0)**

**if(lowest>temp)**

**{**

**ff[i]=j;**

**lowest=temp;**

**}**

**}**

**}**

**frag[i]=lowest;**

**ab[ff[i]]=1;**

**lowest=1000;**

**}**

**for(i=1;i<=n;i++)**

**{**

**if(frag[i]<1000)**

**printf("\njob %d of size %d fitted in partition %d and remaining capacity\t%d",i,b[i],ff[i],a[ff[i]]-b[i]);**

**else**

**printf("\ninsufficient memory for job %d",i);**

**}**

**break;**

**case 4: exit(0);**

**}**

**} while(ch!=4);**

**}**

**Output:**

**Enter the number of memory partitions: 3**

**Enter the partition size:**

**5**

**7**

**3**

**Memory Partitions**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**1----5**

**2----7**

**3----3**

**Enter no.of jobs: 3**

**Enter jobs size (at end give digit -5)**

**6**

**3**

**8**

**-5**

**1.first fit**

**2.worstfit**

**3.bestfit**

**4.exit**

**Enter ur choice:1**

**job 1 of size 6 is fitted in 2 Remaining capacity is 1**

**job 2 of size 3 is fitted in 1 Remaining capacity is 2**

**insufficient memory for job 3**

**Enter ur choice:2**

**Job 1 is fitted in 2 (7) Remaining capacity is 1**

**Job 2 is fitted in 1 (5) Remaining capacity is 2**

**job 3 is not fitted**

**Enter ur choice:3**

**job 1 of size 6 fitted in partition 2 and remaining capacity 1**

**job 2 of size 3 fitted in partition 3 and remaining capacity 0**

**insufficient memory for job 3**

**Enter ur choice:4**

**Program No.: 10**

**Problem Statement:** **Simulate how parent and child processes use shared memory and address space.**

**Source Code:**

**//SHMServer.C**

**#include <sys/types.h>**

**#include <sys/ipc.h>**

**#include <sys/shm.h>**

**#include <stdio.h>**

**#include <stdlib.h>**

**#define MAXSIZE 27**

**void die(char \*s)**

**{**

**perror(s);**

**exit(1);**

**}**

**int main()**

**{**

**char c;**

**int shmid;**

**key\_t key;**

**char \*shm, \*s;**

**key = 5678;**

**if ((shmid = shmget(key, MAXSIZE, IPC\_CREAT | 0666)) < 0)**

**die("shmget");**

**if ((shm = shmat(shmid, NULL, 0)) == (char \*) -1)**

**die("shmat");**

**/\***

**\* \* Put some things into the memory for the**

**\* other process to read.**

**\* \*/**

**s = shm;**

**for (c = 'a'; c <= 'z'; c++)**

**\*s++ = c;**

**/\***

**\* Wait until the other process**

**\* changes the first character of our memory**

**\* to '\*', indicating that it has read what**

**\* we put there.**

**\*/**

**while (\*shm != '\*')**

**sleep(1);**

**exit(0);**

**}**

**//SHMClient.C**

**#include <sys/types.h>**

**#include <sys/ipc.h>**

**#include <sys/shm.h>**

**#include <stdio.h>**

**#include <stdlib.h>**

**#define MAXSIZE 27**

**void die(char \*s)**

**{**

**perror(s);**

**exit(1);**

**}**

**int main()**

**{**

**int shmid;**

**key\_t key;**

c**har \*shm, \*s;**

**key = 5678;**

**if ((shmid = shmget(key, MAXSIZE, 0666)) < 0)**

**die("shmget");**

**if ((shm = shmat(shmid, NULL, 0)) == (char \*) -1)**

**die("shmat");**

**//Now read what the server put in the memory.**

**for (s = shm; \*s != '\0'; s++)**

**putchar(\*s);**

**putchar('\n');**

**/\***

**\*Change the first character of the**

**\*segment to '\*', indicating we have read**

**\*the segment.**

**\*/**

**\*shm = '\*';**

**exit(0);**

**}**

**OUTPUT:**

**After running the Server you can see the attached Shared Memory**

**vgupta80@linux unixprog> ipcs -m**

**------ Shared Memory Segments --------**

**key shmid owner perms bytes nattch status**

**0x0000162e 65537 vgupta80 666 27 1**

**After running the client the memory is freed.**

**------ Shared Memory Segments --------**

**key shmid owner perms bytes nattch status**

**0x0000162e 65537 vgupta80 666 27 0**

**Program No.: 12**

**Problem Statement:Simulate dining philosophers problem**

**Source Code:**

**#include<stdio.h>**

**#include<process.h**

**int tph, philname[20], status[20], howhung, hu[20], cho;**

**main()**

**{**

**int i;**

**printf("\n\nDINING PHILOSOPHERS PROBLEM");**

**printf("\nEnter the total no. of philosophers: ");**

**scanf("%d",&tph);**

**for(i=0;i<tph;i++)**

**{**

**philname[i] = (i+1);**

**status[i]=1;**

**}**

**printf("How many are hungry : ");**

**scanf("%d", &howhung);**

**if(howhung==tph)**

**{**

**printf("\nAll are hungry..\nDead lock stage will occur");**

**printf("\nExiting..");**

**}**

**else**

**{**

**for(i=0;i<howhung;i++)**

**{**

**printf("Enter philosopher %d position: ",(i+1));**

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

**status[hu[i]]=2;**

**}**

**do**

**{**

**printf("1.One can eat at a time\t2.Two can eat at a time\t3.Exit\nEnter your choice:");**

**scanf("%d", &cho);**

**switch(cho)**

**{**

**case 1: one();**

**break;**

**case 2: two();**

**break;**

**case 3: exit(0);**

**default: printf("\nInvalid option..");**

**}**

**46**

**}while(1);**

**}**

**}**

**one()**

**{**

**int pos=0, x, i;**

**printf("\nAllow one philosopher to eat at any time\n");**

**for(i=0;i<howhung; i++, pos++)**

**{**

**printf("\nP %d is granted to eat", philname[hu[pos]]);**

**for(x=pos;x<howhung;x++)**

**printf("\nP %d is waiting", philname[hu[x]]);**

**}**

**}**

**two()**

**{**

**int i, j, s=0, t, r, x;**

**printf("\n Allow two philosophers to eat at same time\n");**

**for(i=0;i<howhung;i++)**

**{**

**for(j=i+1;j<howhung;j++)**

**{**

**if(abs(hu[i]-hu[j])>=1&& abs(hu[i]-hu[j])!=4)**

**{**

**printf(**"**\n\ncombination %d \n", (s+1));**

**t=hu[i];**

**r=hu[j];**

**s++;**

**printf("\nP %d and P %d are granted to eat", philname[hu[i]],**

**philname[hu[j]]);**

**for(x=0;x<howhung;x++)**

**{**

**if((hu[x]!=t)&&(hu[x]!=r))**

**printf("\nP %d is waiting", philname[hu[x]]);**

**}**

**}**

**}**

**}**

**}**

**Sample Output:**

**DINING PHILOSOPHERS PROBLEM**

**Enter the total no. of philosophers: 5**

**How many are hungry : 3**

**Enter philosopher 1 position: 2**

**Enter philosopher 2 position: 4**

**Enter philosopher 3 position: 5**

**OUTPUT**

**1.One can eat at a time 2.Two can eat at a time 3.Exit**

**Enter your choice: 1**

**Allow one philosopher to eat at any time**

**P 3 is granted to eat**

**P 3 is waiting**

**P 5 is waiting**

**P 0 is waiting**

**P 5 is granted to eat**

**P 5 is waiting**

**P 0 is waiting**

**P 0 is granted to eat**

**P 0 is waiting**

**1.One can eat at a time 2.Two can eat at a time 3.Exit**

**Enter your choice: 2**

**Allow two philosophers to eat at same time**

**combination 1 P 3 and P 5 are granted to eat**

**P 0 is waiting**

**combination 2 P 3 and P 0 are granted to eat**

**P 5 is waiting**

**combination 3 P 5 and P 0 are granted to eat**

**P 3 is waiting**

**1.One can eat at a time 2.Two can eat at a time 3.Exit**

**Enter your choice: 3**

**Program No.: 13**

**Problem Statement:** **Simulate producer and consumer problem using threads (use java)**

**Source Code:**

**public** **class** Q {

**boolean** valueSet = **false**;

**int** n;

**synchronized** **int** get() {

**if** (!valueSet)

**try** {

wait();

} **catch** (InterruptedException e) {

System.out.println("Exception is:" + e);

}

System.out.println("got:" + n);

valueSet = **false**;

notify();

**return** n;

}

**synchronized** **void** put(**int** n) {

**if** (valueSet)

**try** {

wait();

} **catch** (InterruptedException e) {

System.out.println("\n Exception in put:" + e);

}

**this**.n = n;

valueSet = **true**;

System.out.println("\n put:" + n);

notify();

}

}

**public** **class** Producer **implements** Runnable {

Q q;

Producer(Q q) {

**this**.q = q;

**new** Thread(**this**, "Producer").start();

}

**public** **void** run() {

**int** i = 0;

**while** (**true**)

q.put(i++);

}

}

**public** **class** Consumer **implements** Runnable {

Q q;

Consumer(Q q) {

**this**.q = q;

**new** Thread(**this**, "Consumer").start();

}

**public** **void** run() {

**while** (**true**)

q.get();

}

}

**public** **class** ProducerConsumer {

**public** **static** **void** main(String args[]) {

Q q = **new** Q();

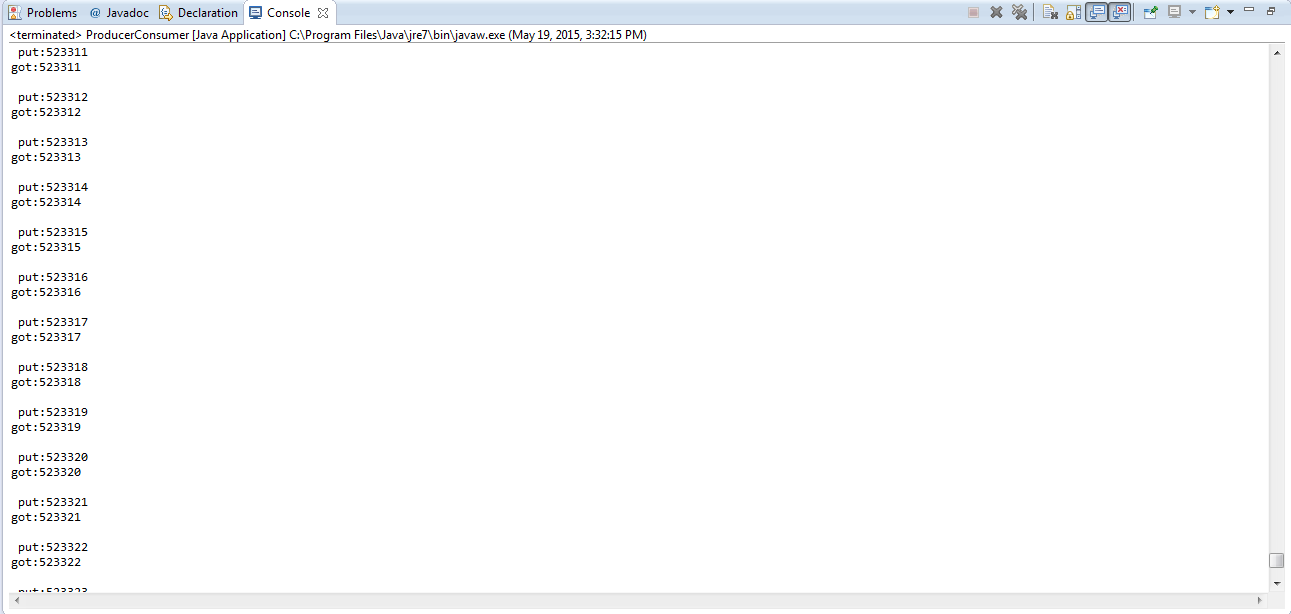
**new** Producer(q);

**new** Consumer(q);

}

}

**Output:**



**Program No.: 16 & 17**

**Problem Statement: Simulate how operating system allocates frame to process**

**Source Code:**

**/\*Conversion of Virtual to Physical address\*/**

**#include<stdio.h>**

**#include<stdlib.h>**

**#include<time.h>**

**struct MainMem**

**{**

**int start\_add;**

**int pn;**

**};**

**int main()**

**{**

**int s,i,totPages,pno,offset,n,r,arr[200]={0},phy\_add,fno[100];**

**struct MainMem mm[100];**

**srand((unsigned)time(NULL));**

**printf("Logical Address To Physical Address\n");**

**printf("Enter  the Size of File : ");**

**scanf("%d",&n);**

**printf("Enter the Page Size : ");**

**scanf("%d",&s);**

**totPages=n/s;**

**for(i=0;i<totPages;i++)**

**{**

**r=rand()%totPages;**

**if(arr[r] == 1)**

**{**

**i--;**

**continue;**

**}**

**arr[r]=1;**

**mm[i].pn=r;**

**mm[i].start\_add=i\*s;**

**fno[r]=i;**

**}**

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

**printf("The Structure of Main Memory\n");**

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

**printf("Frame\tPage\nNumber\tNumber\n------\t------\n");**

**for(i=0; i < totPages; i++)**

**{**

**printf("%d**\**t%d\n",i,mm[i].pn);**

**}**

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

**printf("Enter The Logical Address\nPage Number : ");**

**scanf("%d",&pno);**

**printf("Offset : ");**

**scanf("%d",&offset);**

**if(pno >= totPages || offset > s)**

**{**

**printf("Invalid Input\n");**

**return 0;**

**}**

**phy\_add = mm[fno[pno]].start\_add + offset;**

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

**printf("Physical Address : %d\n",phy\_add);**

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

**return 0;**

**}**

**Sample Output:**

**Logical Address To Physical Address**

**Enter  the Size of File : 100**

**Enter the Page Size : 10**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**The Structure of Main Memory**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Frame   Page**

**Number  Number**

**------  ------**

**0       1**

**1       0**

**2       7**

**3       3**

**4       4**

**5       6**

**6       8**

**7       2**

**8       9**

**9       5**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Enter The Logical Address**

**Page Number : 7**

**Offset : 7**

\***\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Physical Address : 27**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Program No.: 18**

**Problem Statement: Simulate the prediction of deadlock in operating system when all the processes announce their resource requirement in advance.**

**Source Code:**

**#include<stdio.h>;**

**void main()**

**{**

**int found,flag,l,p[4][5],tp,c[4][5],i,j,k=1,m[5],r[5],a[5],temp[5],sum=0;**

**printf("Enter total no of processes");**

**scanf("%d",&tp);**

**printf("enter claim matrix");**

**for(i=1;i<=4;i++)**

**for(j=1;j<=5;j++)**

**{**

**scanf("%d",&c[i][j]);**

**}**

**printf("Enter allocation matrix");**

**for(i=1;i<=4;i++)**

**for(j=1;j<=5;j++)**

**{**

**scanf("%d",&p[i][j]);**

**}**

**printf("Enter resource vector:\n");**

**for(i=1;i<=5;i++)**

**{**

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

**}**

**printf("Enter availability vector:\n");**

**for(i=1;i<=5;i++)**

**{**

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

**temp[i]=a[i];**

**}**

**for(i=1;i<=4;i++)**

**{**

**sum=0;**

**for(j=1;j<=5;j++)**

**{**

**sum+=p[i][j];**

**}**

**if(sum==0)**

**{**

**m[k]=i;**

**k++;**

**}**

**}**

**for(i=1;i<=4;i++)**

**{**

**for(l=1;l<k;l++)**

**if(i!=m[l])**

**{**

**flag=1;**

**for(j=1;j<=5;j++)**

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

**{**

**flag=0;**

**break;**

**}**

**}**

**if(flag==1)**

**{**

**m[k]=i;**

**k++;**

**for(j=1;j<=5;j++)**

**temp[j]+=p[i][j];**

**}**

**}**

**printf("Deadlock causing processes are:");**

**for(j=1;j<=tp;j++)**

**{**

**found=0;**

**for(i=1;i<k;i++)**

**{**

**if(j==m[i])**

**found=1;**

**}**

**if(found==0)**

**printf("P%d\t",j);**

**}**

**}**

**Sample Output:**

***Enter total no. of processes : 4  
Enter claim matrix :  
0 1 0 0 1  
0 0 1 0 1  
0 0 0 0 1***  
1 **0 1 0 1  
Enter allocation matrix :  
1 0 1 1 0  
1 1 0 0 0  
0 0 0 1 0  
0 0 0 0 0  
Enter resource vector :  
2 1 1 2 1  
Enter the availability vector :  
0 0 0 0 1**

**OUTPUT :  
Deadlock causing processes are : P1 P2**

**ADDITIONAL TASKS**

**TASK 1: Write a programs to simulate UNIX commands like ls, grep, etc.**

**AIM:**

To write a program to simulate UNIX commands like ls, grep, etc.

**ALGORITHM:**

1. Start the program.

2. Read the input through command line.

3. Open the specified file.

4. Options (c & i) are performed.

5. Stop the program.

**PROGRAM:**

#include<stdio.h>

#include<string.h>

main(int ag,char\* arg[])

{

char buf[200],line[200];

int i,j,n,fd1,count=0,opt;

if(ag==4)

{

fd1=open(arg[3],0);

if(strcmp(arg[1],"-c")==0)

opt=2;

if(strcmp(arg[1],"-i")==0)

opt=3;

}

else if(ag==3)

{ fd1=open(arg[2],0); opt=1;

}

if(fd1==-1)

printf("error in opening");

j=0;

switch(opt)

{

case 1:

while((n=read(fd1,buf,sizeof(line)))>0)

{

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

{

if(buf[i]!='\n') line[j]=buf[i];

else

{

line[j]='\n';

if(strstr(line,arg[1])!=0)

write(1,line,j+1); } } }

break;

case 2:

while((n=read(fd1,buf,sizeof(line)))>0)

{

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

{

if(buf[i]!='\n') line[j]=buf[i];

else

{

line[j]='\n';

if(strstr(line,arg[2])!=0)

count=count+1;

j=-1;

}

}

}

printf("%d \n",count);

break;

case 3:

while((n=read(fd1,buf,sizeof(line)))>0)

{

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

{

if(buf[i]!='\n') line[j]=buf[i];

else

{

line[j]='\n';

if(strcasestr(line,arg[2])!=0)

write(1,line,j+1);

j=-1;

}

}

}

break;

}

close(fd1);

}

**SAMPLE OUTPUT:**

[root@localhost ~]# cat tst sd

dsaASD[root@localhost ~]# ./a.out -i a tst

aA[root@localhost ~]# ./a.out -c a tst 1[root@localhost ~]# ./a.out -c A tst 1[root@localhost ~]# ./a.out -c sd tst 1[root@localhost ~]# ./a.out -c s tst 2

**TASK 2:** Implementation of process management using the following system

calls of UNIX operating system: fork, exec, getpid, exit, wait, close.

**AIM:**

To write a program for implementing process management using the following system calls of UNIX operating system: fork, exec, getpid, exit, wait, close.

**ALGORITHM:**

1. Start the program.

2. Read the input from the command line.

3. Use fork() system call to create process, getppid() system call used to get the parent process ID and getpid() system call used to get the current process ID

4. execvp() system call used to execute that command given on that command line argument

5. execlp() system call used to execute specified command.

6. Open the directory at specified in command line input.

7. Display the directory contents.

8. Stop the program.

**PROGRAM:**

#include<stdio.h>

main(int arc,char\*ar[])

{

int pid; char s[100];

pid=fork();

if(pid<0)

printf("error");

else if(pid>0)

{

wait(NULL);

printf("\n Parent Process:\n");

printf("\n\tParent Process id:%d\t\n",getpid());

execlp("cat","cat",ar[1],(char\*)0);

error("can’t execute cat %s,",ar[1]);

}

else

{

printf("\nChild process:");

printf("\n\tChildprocess parent id:\t %d",getppid());

sprintf(s,"\n\tChild process id :\t%d",getpid());

write(1,s,strlen(s));

printf(" ");

printf(" ");

printf(" ");

execvp(ar[2],&ar[2]);

error("can’t execute %s",ar[2]);

}

}

**OUTPUT:**

[root@localhost ~]# ./a.out tst date

Child process:

Child process id :

3137 Sat Apr 10 02:45:32 IST 2010

Parent Process:

Parent Process id:3136

sd

dsaASD[root@localhost ~]# cat tst

sd

dsaASD

**RESULT:**

Thus the program for process management was written and successfully executed.