1. C program to emulate the UNIX ls –l command.

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <stdlib.h>
int main()
{
        int pid;
        pid = fork();
        if ( pid < 0 )
        {
                printf("\nFork failed\n");
                exit (-1);
        }
        else if ( pid == 0 )
        {
                execlp ( "/bin/ls", "ls", "-l", NULL );
        }
        else
        {
                wait (NULL);
                printf("\nchild complete\n");
                exit (0);
        }
}
```

2. C program that illustrates how to execute two commands concurrently with a command pipe. Ex: - ls –l | sort

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <stdlib.h>
int main()
{
        int pfds[2];
        char buf[30];
        if(pipe(pfds)==-1)
        {
                perror("pipe failed");
                exit(1);
        }
        if(!fork())
        {
                close(1);
                dup(pfds[1];
                system ("ls –l");
        }
        else
        {
                printf("parent reading from pipe \n");
                while(read(pfds[0],buf,80))
```

```
                printf("%s \n" ,buf);
        }
}
```

3. CPU scheduling algorithms-FCFS

```
#include<stdio.h>
#include<conio.h>
int main()
{
        int i,n,wt[20],bt[20],tat[20];
        float wtavg,tatavg;
        printf("\nEnter the no.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+=wt[i];
                tatavg+=tat[i];
        }
        printf("\n\t PROCESS \tBURST TIME \tWAITING TIME \tTURN AROUND TIME\n");
        for(i=0;i<n;i++)
                printf("\n\tP%d \t\t %d \t\t %d \t\t %d\n",i,bt[i],wt[i],tat[i]);
        printf("\nAverage Waiting Time - %f",wtavg/n);
        printf("\nAverage Turn Around Time - %f",tatavg/n);
        return 0;
}
```

4. CPU scheduling algorithms-SJF

```
#include<stdio.h>
#include<conio.h>
int main()
{
        int n,p[20],bt[20],wt[20],tat[20],i,j,temp;
        float wtavg,tatavg;
        printf("Enter the no.of processes:");
        scanf("%d",&n);
        for(i=0;i<n;i++)
        {
                p[i]=i;
                printf("Enter the burst time for process %d - ");
                scanf("%d",&bt[i]);
        }
        for(i=0;i<n;i++)
        {
                for(j=i+1;j<n;j++)
```

```
		{
			if(bt[i]>bt[j])
			{
				temp=bt[i];
				bt[i]=bt[j];
				bt[j]=temp;

				temp=p[i];
				p[i]=p[j];
				p[j]=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+=wt[i];
		tatavg+=tat[i];
	}
	printf("\n\tPROCESS \tBURST TIME\tWAITING TIME\tTURN AROUND TIME\n");
	for(i=0;i<n;i++)
		printf("\n\tP[%d]\t\t %d \t\t %d \t\t %d\n",p[i],bt[i],wt[i],tat[i]);
	printf("\nAverage Waiting Time:%f\n",wtavg/n);
	printf("Average Turn Around Time:%f",tatavg/n);

}
```

5. CPU scheduling algorithms-Round Robin

```
#include<stdio.h>
#include<conio.h>
int main()
{
	int i,j,n,t,max,ct[20],wt[20],tat[20],bt[20],temp;
	float awt,att;
	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",&bt[i]);
		ct[i]=bt[i];
	}
	printf("\nEnter the size of time slice -- ");
	scanf("%d",&t);
	max=bt[0];
	for(i=1;i<n;i++)
		if(max<bt[i])
			max=bt[i];
	for(j=0;j<(max/t)+1;j++)
```

```
		for(i=0;i<n;i++)
			if(bt[i]!=0)
				if(bt[i]<=t)
				{
					tat[i]=temp+bt[i];
					temp=temp+bt[i];
					bt[i]=0;
				}
				else
				{
					bt[i]=bt[i]-t;
					temp=temp+t;
				}
	for(i=0;i<n;i++)
	{
		wt[i]=tat[i]-
		ct[i];
		att+=tat[i];
		awt+=wt[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],wt[i],tat[i]);
}
```

6. CPU scheduling algorithms-Priority

```
#include<stdio.h>
main()
{
	int p[20],bt[20],pri[20], wt[20],tat[20],i, k, n, temp; float wtavg,
	tatavg;
	printf("Enter the number of processes - ");
	scanf("%d",&n);
	for(i=0;i<n;i++)
	{
		p[i] = i;
		printf("Enter the Burst Time & Priority of Process %d - ",i);
		scanf("%d %d",&bt[i],&pri[i]);
	}
	for(i=0;i<n;i++)
		for(k=i+1;k<n;k++)
			if(pri[i] > pri[k])
			{
				temp=p[i];
				p[i]=p[k];
				p[k]=temp;
				temp=bt[i];
				bt[i]=bt[k];
				bt[k]=temp;
				temp=pri[i];
```

```
                                pri[i]=pri[k];
                                pri[k]=temp;
                        }
        wtavg = wt[0] = 0;
        tatavg = tat[0] = bt[0];
        for(i=1;i<n;i++)
        {
                wt[i] = wt[i-1] + bt[i-1];
                tat[i] = tat[i-1] + bt[i];
                wtavg = wtavg + wt[i];
                tatavg = tatavg + tat[i];
        }
        printf("\nPROCESS\t\tPRIORITY\tBURST TIME\tWAITING TIME\tTURNAROUNDTIME");
        for(i=0;i<n;i++)
                printf("\n%d \t\t %d \t\t %d \t\t %d \t\t %d ",p[i],pri[i],bt[i],wt[i],tat[i]);
        printf("\nAverage Waiting Time is - %f",wtavg/n);
        printf("\nAverage Turnaround Time is - %f",tatavg/n);
}
```

7. Multiprogramming with a fixed number of tasks (MFT)

```
#include<stdio.h>
#include<conio.h>
main()
{
        int ms, bs, nob, ef,n,
        mp[10],tif=0; int i,p=0;
        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\tMEMORYREQUIRED\tALLOCATED\tINTERNALFRAGMENTATION");
        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);
}
```

8. Multiprogramming with a variable number of tasks (MVT)

```
#include<stdio.h>
#include<conio.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);
}
```

9. Simulate Bankers Algorithm for Dead Lock Avoidance

```
#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];
```

```
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");
}
```

10. Simulate Bankers Algorithm for Dead Lock Detection.

```
#include<stdio.h>
#include<conio.h>
int max[100][100];
int alloc[100][100];
int need[100][100];
int avail[100];
int n,r;
void input();
void show();
void cal();
int main()
{
        int i,j;
        printf("*Deadlock Detection*\n");
        input();
        show();
        cal();
   return 0;
}
void input()
{
        int i,j;
        printf("Enter the no of Processes: ");
        scanf("%d",&n);
        printf("Enter the no of resource instances: ");
        scanf("%d",&r);
        printf("Enter the Max Matrix:\n");
        for(i=0;i<n;i++)
        {
                for(j=0;j<r;j++)
                {
                        scanf("%d",&max[i][j]);
                }
        }
        printf("Enter the Allocation Matrix:\n");
        for(i=0;i<n;i++)
        {
                for(j=0;j<r;j++)
                {
                        scanf("%d",&alloc[i][j]);
                }
        }
        printf("Enter the available Resources:\n");
        for(j=0;j<r;j++)
        {
                scanf("%d",&avail[j]);
        }
```

```
}
void show()
{
	int i,j;
	printf("Process\t Allocation\t Max\t Available\t");
	for(i=0;i<n;i++)
	{
		printf("\nP%d\t ",i+1);
		for(j=0;j<r;j++)
		{
			printf("%d ",alloc[i][j]);
		}
		printf("\t");
		for(j=0;j<r;j++)
		{
			printf("%d ",max[i][j]);
		}
		printf("\t");
		if(i==0)
		{
			for(j=0;j<r;j++)
				printf("%d ",avail[j]);
		}
	}
}
void cal()
{
	int finish[100],temp,need[100][100],flag=1,k,c1=0;
	int dead[100];
	int safe[100],i,j;
	for(i=0;i<n;i++)
	{
		finish[i]=0;
	}
	for(i=0;i<n;i++)
	{
		for(j=0;j<r;j++)
		{
			need[i][j]=max[i][j]-alloc[i][j];
		}
	}
	while(flag)
	{
		flag=0;
		for(i=0;i<n;i++)
		{
			int c=0;
			for(j=0;j<r;j++)
			{
				if((finish[i]==0)&&(need[i][j]<=avail[j]))
				{
```

```
                                        c++;
                                        if(c==r)
                                        {
                                                for(k=0;k<r        ;k++)
                                                {
                                                        avail[k]+=alloc[i][j];
                                                        finish[i]=1;
                                                        flag=1;
                                                }
                                                if(finish[i]==1)
                                                {
                                                        i=n;
                                                }
                                        }
                                }
                        }
                }
        }
        j=0;
        flag=0;
        for(i=0;i<n;i++)
        {
                if(finish[i]==0)
                {
                        dead[j]=i;
                        j++;
                        flag=1;
                }
        }
        if(flag==1)
        {
                printf("\n\nSystem is in Deadlock and the Deadlock process are:\n");
                for(i=0;i<n;i++)
                {
                        printf("P%d\t",dead[i]);
                }
        }
        else
        {
                printf("\nNo Deadlock Occur");
        }
}
```

11. #FIFO PAGE REPLACEMENT

```
#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;
```

```
	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+frsize);
}
void display()
{
	int i;
	printf("\n");
	for(i=0;i<3;i++)
		printf("%d\t",fr[i]);
}
```

12. #LRU PAGE REPLACEMENT

```
#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;
	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("\nNo of page faults :%d",pf+frsize);
}
void display()
{
        int i;
        printf("\n");
        for(i=0;i<3;i++)
                printf("\t%d",fr[i]);
}
```

13. #LFU PAGE REPLACEMENT

```
#include<stdio.h>
int main()
{
        int f,p;
        int pages[50],frame[10],hit=0,count[50],time[50];
        int i,j,page,flag,least,minTime,temp;
        printf("Enter no of frames : ");
        scanf("%d",&f);
        printf("Enter no of pages : ");
        scanf("%d",&p);
        for(i=0;i<f;i++)
        {
                frame[i]=-1;
        }
        for(i=0;i<50;i++)
        {
                count[i]=0;
        }
        printf("Enter page no : \n");
        for(i=0;i<p;i++)
        {
                scanf("%d",&pages[i]);
        }
        printf("\n");
        for(i=0;i<p;i++)
        {
                count[pages[i]]++;
                time[pages[i]]=i;
                flag=1;
                least=frame[0];
                for(j=0;j<f;j++)
                {
                        if(frame[j]==-1 || frame[j]==pages[i])
                        {
                                if(frame[j]!=-1)
                                {
                                        hit++;
```

```
                        }
                        flag=0;
                        frame[j]=pages[i];
                        break;
                }
                if(count[least]>count[frame[j]])
                {
                        least=frame[j];
                }
        }
        if(flag)
        {
                minTime=50;
                for(j=0;j<f;j++)
                {
                        if(count[frame[j]]==count[least] && time[frame[j]]<minTime)
                        {
                                temp=j; minTime=time[frame[j]];
                        }
                }
                count[frame[temp]]=0;
                frame[temp]=pages[i];
        }
        for(j=0;j<f;j++)
        {
                printf("%d ",frame[j]);
        }
        printf("\n");
    }
    printf("Page hit = %d",hit);
    return 0;
}
```

14. #FAS_Sequenced

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int f[50],i,j,c,st,len;
    for(i=0;i<50;i++)
        f[i]=0;
    X:
    printf("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("%d -> %d\n",j,f[j]);
        }
        else
        {
```

```
                        printf("Block already allocated");
                        break;
                }
        if(j==(st+len))
                printf("The file is allocated to disk\n");
        printf("\nDo u want to enter more files?(1-yes/0-no)");
        scanf("%d",&c);
        if(c==1) goto X;
        else    exit(0);
}
```

15. #FAS_Indexed

```
#include<stdio.h>
#include<stdlib.h>
int     f[50],i,k,j,inde[50],n,c,count=0,p;
main()
{
        for(i=0;i<50;i++)
                f[i]=0;
        x:
        printf("Enter index block:");
        scanf("%d",&p);
        if(f[p]==0)
        {
                f[p]=1;
                printf("Enter no.of files on index %d:",p);
                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\n");
                        goto x;
                }
        for(j=0;j<n;j++)
                f[inde[j]]=1;
        printf("\nallocated");
        printf("\nfile indexed");
        for(k=0;k<n;k++)
                printf("\n %d->%d:%d",p,inde[k],f[inde[k]]);
        printf("\nEnter 1 to enter more files and 0 to exit:");
        scanf("%d",&c);
        if(c==1) goto x;
        else exit(0);
}
```

16. #FAS_Linked

```
#include<stdio.h>
main()
{
	int f[50],p,i,j,k,a,st,len,n,c;
	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("\nEnter 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("\nIf u want to enter one more file? (yes-1/no-0)");
	scanf("%d",&c);
	if(c==1)
		goto X;
	else
		exit(0);
}
```

17. C program that illustrates two processes communicating using sharedmemory.

#Program-1

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/types.h>
#include<string.h>
int main()
{
	int i;
	void *shared_memory;
```

```
        char buff[100];
        int shmid;
        shmid=shmget((key_t)7777,1024,0666 | IPC_CREAT);
        printf("Key of Share memory is %d\n",shmid);
        shared_memory=shmat(shmid,NULL,0);
        printf("Process attached at %p\n",shared_memory);
        pritnf("Enter some data to write into shared memory:\n");
        read(0,buff,100);
        strcpy(shared_memory,buff);
        printf("You write : %s\n",(char*)shared_memory);
}
#Program-2

#include<stdio.h>
#include<stdlib.h>
#include<sys/shm.h>
#include<string.h>
int main()
{
        int i;
        void *shared_memory;
        char buff[100];
        int shmid;
        shmid=shmget((key_t)7777,1024,0666 | IPC_CREAT);
        printf("Key of Share memory is %d\n",shmid);
        shared_memory=shmat(shmid,NULL,0);
        printf("Process attached at %p\n",shared_memory);
        printf("Data read from shared memory is: %s\n",(char*)shared_memory);
}
18. C program to simulate producer and consumer problem using semaphores.
#include<stdio.h>
#include<stdlib.h>
int mutex=1,full=0,empty=5,x=0;
int main()
{
        int n;
        void producer();
        void consumer();
        int wait(int);
        int signal(int);
        printf("\n1.PRODUCER\n2.CONSUMER\n3.EXIT\n");
        while(1)
        {
                printf("\nEnter your choice:\n");
                scanf("%d",&n);
                switch(n)
                {
                        case 1:
                                if((mutex==1)&&(empty!=0))
                                        producer();
                                else
```

```
                                        printf("BUFFER IS FULL");
                                        break;
                                case 2:
                                        if((mutex==1)&&(full!=0))
                                                consumer();
                                        else
                                                printf("BUFFER IS EMPTY");
                                                break;
                                case 3:
                                        exit(0);
                                        break;
                        }
                }
}
int wait(int s)
{
        return(--s);
}
int signal(int s)
{
        return(++s);
}
void producer()
{
        mutex=wait(mutex);
        full=signal(full);
        empty=wait(empty);
        x++;
        printf("\nProducer produces the item%d",x);
        mutex=signal(mutex);
}
void consumer()
{
        mutex=wait(mutex);
        full=wait(full);
        empty=signal(empty);
        printf("\nConsumer consumes item%d",x);
        x--;
        mutex=signal(mutex);
}
```

19. C program that makes a copy of a file using standard I/O.

```
#Copy of a file using standard I/O

#include<stdio.h>
int main(int argc,char *argv[])
{
   FILE *fp1,*fp2;
   int ch;
   fp1=fopen(argv[1],"r");
   fp2=fopen(argv[2],"w");
   while((ch=fgetc(fp1))!=-1)
```

```
    fputc(ch,fp2);
}
```

20.C program that makes a copy of a file using system calls.

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include<stdlib.h>
int main(int argc,char *argv[])
{
        int f1,f2;
        char buff[50];
        long int n;
        if(((f1 =open(argv[1],O_RDONLY))==-1 || ((f2=open(arg [2],O_CREAT | O_WRONLY |
O_TRUNC,100))==1)))
        {
                perror("Problem in file\n");
                exit(1);
        }
        while((n=read(f1,buff,50))>0)
                if(write(f2,buff,n)!=n)
                {
                        perror("Problem in writing");
                        exit(3);
                }
        if(n==-1)
        {
                perror("Problem in reading");
        }
        close(f2);
        exit(0);
}
```

21. #Multiprogramming-Memory management-Implementation of fork (), wait (), exec() and exit (), System calls

#Fork & Wait SystemCalls

```
#include<stdio.h>
#include<unistd.h>
main()
{
   int pid,status,childpid;
   printf(" parent process with PID is %d \n ",getpid());
   pid=fork();
   if(pid!=0)
   {
      sleep(1);
      printf(" parent process with PID %d and PPID %d \n ",getpid(),getppid());
      childpid=wait(&status);
      printf("child process PID %d terminated with exit code %d\n",childpid,status>>8);
   }
   else
   {
```

```
      printf(" child process with PID %d and PPId %d \n ",getpid(),getppid());
      sleep(5);
      exit(42);
   }
   printf("PID %d terminates \n ",getpid());
}
```

OUTPUT:

```
parent process with PID is 7242
 child process with PID 7243 and PPId 7242
 parent process with PID 7242 and PPID 6984
 child process PID 7243 terminated with exit code 42
PID 7242 terminates
```

#execvp system call

```
#include<stdio.h>
main( int argc,char *argv[])
{
   if(fork()==0)
   {
      execvp(argv[1],&argv[1]);
      fprintf(stderr,"couldnot execute %s \n",argv[1]);
   }
}
```

#execl system call

```
#include<stdio.h>
main()
{
printf("i am process %d and iam about exec an ls -1\n",getpid());
execl("/bin/ls","ls","-1",NULL);
printf("this line should never be executed\n");
}
```

OUTPUT:

```
i am process 7189 and iam about exec an ls -1
a.out
armstrong.sh
banker.c
class
d1
d2
```