**Part B: Cn**

**1: CRC CCITT(16bits)**

#include <stdio.h>

#include <stdlib.h>

int message[100];

int b[100];

int j;

int length;

int gp[17]={1,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,1};

void divide(int k)

{

int i,j,count=0;

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

{

if(message[i]==gp[0])

{

for(j=i;j<17+i;j++)

{

message[j]=message[j]^gp[count++];

}

}

count=0;

}

}

int main()

{

int i;

printf("Enter the length of the Data Frame: \n");

scanf("%d",&length);

printf("\n Enter the message (in bits,ie 0's and 1's)");

printf("\n But each bit separated by space or the new line :");

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

{

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

}

for(i=0;i<16;i++)

{

message[length++]=0;

}

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

{

b[i]=message[i];

}

divide(length-16);

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

{

b[i]=b[i]^message[i];

}

printf("\n Data to be Transmitted \n");

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

{

printf("%2d",b[i]);

}

printf("\n Enter the received data \n");

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

{

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

}

divide(length-16);

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

{

if(message[i]!=0)

{

printf("\n Error in received data");

return 0;

}

}

printf("\n data received in error free");

return 0;

}

**2: Distance vector algorithm:**

#include <stdio.h>

#include <stdlib.h>

int nodes,adjacency[10][10],intermediate[10][10],distance[10][10],i,j,k;

void readRoutingTable()

{

printf("\n enter the node \n");

scanf("%d",&nodes);

printf("\n If no direct edge between vertex u and v or ");

printf("\n if cost is unknown then enter 999, enter 0 if its same node ");

printf("\n\n enter the routing table : \n | ");

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

printf("-------");

printf("\n");

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

{

printf("%c | ",'a'+i);

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

{

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

if(distance[i][j]!=999)

adjacency[i][j]=1;

}

}

}

int main()

{

readRoutingTable();

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

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

intermediate[i][j]=i;

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

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

if(adjacency[i][j])

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

if(distance[i][j]+distance[j][k]<distance[i][k])

{

distance[i][k]=distance[i][j]+distance[j][k];

intermediate[i][k]=j;

}

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

{

printf("\n Table for router %c\n",'a'+i);

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

printf("%c:: %d via %c\n",'a'+j,distance[i][j],'a'+intermediate[i][j]);

}

return 0;

}

**3: Client server program:**

**Client:**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<sys/types.h>

#include<sys/socket.h>

#include<netinet/in.h>

#include<netdb.h>

int main(int argc,char \*argv[])

{

int sockfd,newsockfd,portno,len,n;

char buffer[256],c[20000];

struct sockaddr\_in serv,cli;

FILE \*fd;

if(argc<2)

{

printf("Err:no port no.\nusage:\n./client portno\n ex:./client 7777\n");

exit(1);

}

sockfd=socket(AF\_INET,SOCK\_STREAM,0);

bzero((char \*)&serv,sizeof(serv));

portno=atoi(argv[1]);

serv.sin\_family=AF\_INET;

serv.sin\_port=htons(portno);

if(connect(sockfd,(struct sockaddr \*)&serv,sizeof(serv))<0)

{

printf("server not responding..\n\n\n\ti am to terminate\n");

exit(1);

}

printf("Enter the file with complete path\n");

scanf("%s",&buffer);

if(write(sockfd,buffer,strlen(buffer))<0)

printf("Err writing to socket..\n");

bzero(c,2000);

printf("Reading..\n..\n");

if(read(sockfd,c,1999)<0)

printf("error: read error\n");

printf("client: display content of %s\n..\n",buffer);

fputs(c,stdout);

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

return 0;

}

**Server:**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<sys/types.h>

#include<netinet/in.h>

#include<sys/socket.h>

#include<netdb.h>

int main(int argc,char \*argv[]){

int sockfd,newsockfd,portno,len,n;

char buffer[256],c[2000],cc[20000];

struct sockaddr\_in serv,cli;

FILE \*fd;

if(argc<2){

printf("Error:No port no\n usage:\n/server port\_no\n");

exit(1);

}

sockfd=socket(AF\_INET,SOCK\_STREAM,0);

portno=atoi(argv[1]);

serv.sin\_family=AF\_INET;

serv.sin\_addr.s\_addr=INADDR\_ANY;

serv.sin\_port=htons(portno);

bind(sockfd,(struct sockaddr \*)&serv,sizeof(serv));

listen(sockfd,10);

len=sizeof(cli);

printf("Server:\nWaiting for connection\n");

newsockfd=accept(sockfd,(struct sockaddr \*)&cli,&len);

bzero(buffer,255);

n=read(newsockfd,buffer,255);

printf("\nSERVER RECV:%s\n",buffer);

if((fd=fopen(buffer,"r"))!=NULL){

printf("server:%s found \n opening and reading...\n",buffer);

printf("READING...\n.. WRITING COMPLETE");

fgets(cc,200,fd);

while(!feof(fd)){

fgets(c,2000,fd);

strcat(cc,c);

}

n=write(newsockfd,cc,strlen(cc));

if(n<0){

printf("errioe writing to socket");

printf("transfer complete");

}

}

else{

printf("server file not found");

n=write(newsockfd,"File not found",15);

if(n<0)

printf("erroe:writing to socket");

}

return 0;

}

**4: Rsa algorithm:**

#include <stdio.h>

#include <stdlib.h>

#include<math.h>

int gcd(long m,long n)

{

while(n!=0)

{

long r=m%n;

m=n;

n=r;

}

return m;

}

int rsa(char message[50])

{

long p=0,q=0,n=0,e=0,d=0,phi=0;

long nummes[100]={0};

long encrypted[100]={0},decrypted[100]={0};

long i=0,j=0,nofelem=0;

printf("\n \n enter the value of p and q \n");

scanf("%d%d",&p,&q);

n=p\*q;

phi=(p-1)\*(q-1);

for(i=2;i<phi;i++)

if(gcd(i,phi)==1)break;

e=i;

for(i=2;i<phi;i++)

if((e\*i-1)%phi==0)break;

d=i;

for(i=0;i<strlen(message);i++)

nummes[i]=message[i]-96;

nofelem=strlen(message);

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

{

encrypted[i]=1;

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

encrypted[i]=(encrypted[i]\*nummes[i])%n;

}

printf("\n Encrypted Message \n");

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

{

printf("%ld",encrypted[i]);

printf("%c",(char)(encrypted[i])+96);

}

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

{

decrypted[i]=1;

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

decrypted[i]=(decrypted[i]\*encrypted[i])%n;

}

printf("\n Decrypted Message \n");

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

printf("%c",(char)(decrypted[i])+96);

return 0;

}

int main()

{

char msg[100];

printf("Enter the message to be encrypted\n");

scanf("%s",msg);

rsa(msg);

return 0;

}

**5: leaky bucket algorithm:**

#include <stdio.h>

#include <stdlib.h>

#include<math.h>

void main()

{

int packets[8],i,j,clk,b\_size,o\_rate,i\_rate,p\_sz\_rm=0,p\_sz,p\_time;

// clrscr();

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

{

packets[i]=rand()%10;

if(packets[i]==0) --i;

}

printf("Enter output rate:");

scanf("%d",&o\_rate);

printf("\n Enter bucket size:");

scanf("%d",&b\_size);

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

{

if((packets[i]+p\_sz\_rm)>b\_size)

{

if(packets[i]>b\_size)

printf("\n Incoming packet size:%d greater than bucket capacity\n",packets[i]);

else

printf("Bucket size exceeded\n");

}

else

{

p\_sz=packets[i];

p\_sz\_rm+=p\_sz;

printf("\n---------------------------\n");

printf("Incoming packet:%d",p\_sz);

printf("\nTransmission left\n",p\_sz\_rm);

p\_time=rand()%10;

printf("Next packet will come at %d",p\_time);

for(clk=0;clk<p\_time&&p\_sz\_rm>0;++clk)

{

printf("\n Time left %d---No packets to transmit!!\n",p\_time-clk);

// sleep(1);

if(p\_sz\_rm)

{

printf("Transmitted\n");

if(p\_sz\_rm<o\_rate)

p\_sz\_rm=0;

else

p\_sz\_rm-=o\_rate;

printf("Bytes remaining:%d\n",p\_sz\_rm);

}

else

printf("No packets to transmit\n");

}

}

}

getch();

}

**USP LAB:**

**PART A: Os**

1. **Round Robin**

#include <stdio.h>

int main()

{

int n=4,remain=n;

int processNo,elapsedTime,flag=0,timeQuantum=4;

int totalWaitTime=0,totalTurnAroundTime=0;

int arrivalTime[10]={0,2,2,3};

int burstTime[10]={7,3,8,4};

int remainingTime[10]={7,3,8,4};

printf("\n Process|Turnaround time | Waiting time\n");

for(elapsedTime=0,processNo=0;remain!=0;)

{

if(remainingTime[processNo]>0)

{

if(remainingTime[processNo]<=timeQuantum)

{

elapsedTime+=remainingTime[processNo];

remainingTime[processNo]=0;

flag=1;

remain--;

}

else

{

remainingTime[processNo]-=timeQuantum;

elapsedTime+=timeQuantum;

}

}

if(remainingTime[processNo]==0&&flag==1)

{

printf("p[%d]\t|\t%d\t|\t%d\n",processNo,elapsedTime-arrivalTime[processNo],elapsedTime-arrivalTime[processNo]-burstTime[processNo]);

totalTurnAroundTime+=elapsedTime-arrivalTime[processNo];

totalWaitTime+=elapsedTime-arrivalTime[processNo]-burstTime[processNo];

flag=0;

}

if(processNo==n-1)

processNo=0;

else if(arrivalTime[processNo+1]<=elapsedTime)

processNo++;

else

processNo=0;

}

printf("\n Average turnaround time=%f\n",totalTurnAroundTime\*1.0/n);

printf("\nAverage waiting time=%f",totalWaitTime\*1.0/n);

return 0;

}

**2.Fragmentation**

#include<stdio.h>

#include<stdlib.h>

void display(int,int[10]);

void firstfit(int,int[10],int[10],int);

void bestfit(int,int[10],int[10],int);

void worstfit(int,int[10],int[10],int);

int main(int argc,char\*\*argv)

{

int npr,nh=0,i,pr[10],hol[10],ch,temp[10];

do

{

printf("\n\n -------MENU-------\n 1.INPUT" "\n2.FIRST FIT \n3.BEST FIT""\n4.WORST FIT \n 5.EXIT\n\nENTER YOUR CHOICE:");

scanf("%d",&ch);

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

temp[i]=hol[i];

switch(ch)

{

case 1:printf("\n Enter the no of process ");

scanf("%d",&npr);

printf("\n no of hole");

scanf("%d",&nh);

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

{

printf("\n process %d",i+1);

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

}

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

{

printf("\n hole %d",i+1);

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

temp[i]=hol[i];

}

break;

case 2:firstfit(npr,temp,pr,nh);

break;

case 3:bestfit(npr,temp,pr,nh);

break;

case 4:worstfit(npr,temp,pr,nh);

break;

case 5:exit(0);

//break;

default:printf("\n wrong choice ");

}

}

while(ch!=5);

return 0;

}

void firstfit(int npr,int hol[10],int pr[10],int nh)

{

int i,j,k,flag=0;

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

{

flag=0;

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

{

if(pr[i]<hol[j])

{

hol[j]=hol[j]-pr[i];

printf("\n memory is allocated to process %d:(%d)",i+1,pr[i]);

flag=1;

break;

}

else if(pr[i]==hol[j])

{

flag=1;

printf("\n memory is allocated to process %d:(%d)",i+1,pr[i]);

for(k=j;k<nh-1;k++)

{

hol[k]=hol[k+1];

}

nh--;

break;

}

}

if(flag==0)

{

printf("\n Hole is not available..");

break;

}

display(nh,hol);

}

}

void bestfit(int npr,int hol[10],int pr[10],int nh)

{

int i,j,k,flag,min;

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

{

flag=0;

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

{

if(pr[i]<=hol[j])

{

flag=1,min=j;

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

{

if(hol[min]>hol[k]&&(pr[i]<=hol[k]))

min=k;

}

if(pr[i]<hol[min])

{

printf("\n memory is allocated to process %d:(%d)",i+1,pr[i]);

hol[min]=hol[min]-pr[i];

break;

}

else if(pr[i]==hol[min])

{

printf("\n memory is allocated to process %d:(%d)",i+1,pr[i]);

for(k=min;k<nh-1;k++)

hol[k]=hol[k+1];

nh--;

break;

}

}

if(flag==0)

{

printf("\n Hole is not available..");

break;

}

display(nh,hol);

}

}

}

void worstfit(int npr,int hol[10],int pr[10],int nh)

{

int i,j,k,max,flag;

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

flag=0;

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

{

if(pr[i]<=hol[j])

{

printf("\n memory is allocated to process %d:(%d)",i+1,pr[i]);

flag=1;max=j;

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

{

if(hol[max]<hol[k]&&(pr[i]<=hol[k]))

max=k;

}

if(pr[i]<hol[max])

{

//printf("\n cpu is allocated for process %d",i+1);

hol[max]=hol[max]-pr[i];

break;

}

else if(pr[i]==hol[max])

{

//printf("\n cpu is allocated for process %d",i+1);

for(k=max;k<nh-1;k++)

hol[k]=hol[k+1];

nh--;

break;

}

}

}

if(flag==0)

{

printf("\n Hole is not available..");

//break;

}

display(nh,hol);

}

void display(int nh,int hol[10])

{

printf("\n Holes ");

int i=0;

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

{

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

}

}

**3.bankers algorithm**

#include <stdio.h>

#include <stdlib.h>

int main()

{

int process=5, resource=3;

int i, j, k=0, count1=0, count2=0;

int avail[3]={1, 0, 2};

int max[5][3]={0, 0, 4, 2, 0, 1, 1, 3, 7, 8, 4, 2, 1, 5, 7};

int allot[5][3]={0, 0, 2, 1, 0, 0, 1, 3, 5, 6, 3, 2, 1, 4, 3};

int need[5][3];

int complete[5]={0};

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

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

need[i][j]=max[i][j]-allot[i][j];

printf("\nPossible Sequence: \n");

while(count1!=process)

{

count2=count1;

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

{

k=0;

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

{

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

k++;

}

if(k==resource && complete[i]==0)

{

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

complete[i]=1;

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

{

avail[j]=avail[j]+allot[i][j];

}

count1++;

}

}

if(count1==count2)

{

printf("\nStop.....After this...Deadlock\n");

/\*for(i=0;i<process;i++){

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

printf("%d\t",max[i][j]);

printf("\n");}

printf("\nAllocation");

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

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

printf("%d\t",allot[i][j]);

printf("\n");}

printf("\nNeed");

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

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

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

printf("\n");}

printf("\nSafe Sequence exists\n");

printf("\nStop...After this ...Deadlock");

return 0;

}

}

("\nMaximum");

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

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

printf("%d\t",max[i][j]);

printf("\n");}

printf("\nAllocation");

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

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

printf("%d\t",allot[i][j]);

printf("\n");}

printf("\nNeed");printf

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

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

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

printf("\n");}

printf("\nSafe Sequence exists\n");\*/

return 0;

}

}

return 0;

}

**4.word processor**

**Excel.c**

#include<stdio.h>

#include<sys/ipc.h>

#include<sys/sem.h>

#include<stdio.h>

#include<errno.h>

#include<unistd.h>

#include<stdlib.h>

#define PERMS 0666

#define PRINTERKEY 654321L

int main()

{

int semid=semget(PRINTERKEY,1,IPC\_EXCL);

setbuf(stdout,NULL);

struct sembuf semBufVar;

semBufVar.sem\_num=0;

semBufVar.sem\_op=-1;

semBufVar.sem\_flg=SEM\_UNDO;

printf("\n excel process requested printer resource on-");

system("date");

if(semop(semid,&semBufVar,1)!=-1)

{

printf("\n Printer allocated to excel on");

system("date");

printf("\n excel process Using resource ");

sleep(10);

}

semBufVar.sem\_op=+1;

printf("\n\n Printer resource releases by excel process on -");

system("date");

semop(semid,&semBufVar,1);

return 0;

}

**Print.c**

#include<stdio.h>

#include<sys/ipc.h>

#include<sys/sem.h>

#define PERMS 0666

#define PRINTERKEY 654321L

int main()

{

int semid=semget(PRINTERKEY,1,IPC\_CREAT|PERMS);

int status=semctl(semid,0,SETVAL,1);

return 0;

}

**Word.c**

#include<stdio.h>

#include<sys/ipc.h>

#include<sys/sem.h>

#include<stdio.h>

#include<errno.h>

#include<unistd.h>

#include<stdlib.h>

#define PERMS 0666

#define PRINTERKEY 654321L

int main()

{

int semid=semget(PRINTERKEY,1,IPC\_EXCL);

setbuf(stdout,NULL);

struct sembuf semBufVar;

semBufVar.sem\_num=0;

semBufVar.sem\_op=-1;

semBufVar.sem\_flg=SEM\_UNDO;

printf("\n Word process requested printer resource on-");

system("date");

if(semop(semid,&semBufVar,1)!=-1)

{

printf("\n Printer allocated to word process on");

system("date");

printf("\n Word process Using resource ");

sleep(10);

}

semBufVar.sem\_op=+1;

printf("\n\n Printer resource releases by word process on -");

system("date");

semop(semid,&semBufVar,1);

return 0;

}

**5.pagefaults**

#include <stdio.h>

#include <stdlib.h>

int referenceString[]={1,2,3,4,1,2,5,1,2,3,4,5};

int lengthOfReferenceString=sizeof(referenceString)/sizeof(int);

int pagePresentInFrames;

int pageFaults,i,j,k,m;

void printPagesInFrames(int frame[],int numberOfFrames)

{

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

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

}

int findIndexOfLeastRecentlyUsed(int lruCounter[],int numberOfFrames)

{

int lruValue=-1;

int indexOfLRU=0;

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

if(lruCounter[m]==-1)

return i;

else if(lruCounter[m]>lruValue)

{

lruValue=lruCounter[m];

indexOfLRU=m;

}

return indexOfLRU;

}

void fifoPageReplacement(int numberOfFrames)

{

int frame[5]={-1,-1,-1,-1,-1};

pageFaults=0;

j=0;

printf("\n FIFO page replacement using %d Frames ,initial Frames=",numberOfFrames);

printPagesInFrames(frame,numberOfFrames);

printf("\n Page in reference String Pages in Frames");

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

{

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

pagePresentInFrames=0;

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

if(frame[k]==referenceString[i])

pagePresentInFrames=1;

if(pagePresentInFrames==0)

{

frame[j]=referenceString[i];

j=(j+1)%numberOfFrames;

pageFaults++;

printPagesInFrames(frame,numberOfFrames);

}

}

printf("\n Page Faults are =%d\n",pageFaults);

}

void lruPageReplacement(int numberOfFrames)

{

int lruCounter[5]={-1,-1,-1,-1,-1};

int frame[5]={-1,-1,-1,-1,-1};

pageFaults=0;

printf("\n LRU page replacement using %d Frames ,initial Frames=",numberOfFrames);

printPagesInFrames(frame,numberOfFrames);

printf("\n Page in reference String Pages in Frames");

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

{

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

pagePresentInFrames=0;

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

if(frame[k]==referenceString[i])

{

pagePresentInFrames=1;

lruCounter[k]=0;

}

else if(lruCounter[k]!=-1)

lruCounter[k]++;

if(pagePresentInFrames==0)

{

j=findIndexOfLeastRecentlyUsed(lruCounter,numberOfFrames);

frame[j]=referenceString[i];

lruCounter[j]=0;

pageFaults++;

printPagesInFrames(frame,numberOfFrames);

}

}

printf("\n page Faultsa are =%d\n",pageFaults);

}

int main()

{

printf("Reference String=");

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

printf("%d",referenceString[i]);

fifoPageReplacement(4);

lruPageReplacement(4);

return 0;

}

**Part B:Usp**

**1.compile n run time**

#define \_POSIX\_SOURCE

#define \_POSIX\_C\_SOURCE 199309L

#include<stdio.h>

#include<unistd.h>

#include<iostream>

using namespace std;

int main()

{

int c,res;

cout<<" enter for the compile time 1,run time 2"<<endl;

cin>>c ;

switch(c)

{

case 1:

#ifdef \_POSIX\_CHILD\_MAX

cout<<" the no of child processes "<<\_POSIX\_CHILD\_MAX<<endl;

#endif

#ifdef \_POSIX\_PATH\_MAX

cout<<" Max PAth length "<<\_POSIX\_PATH\_MAX<<endl;

#endif

#ifdef \_POSIX\_NAME\_MAX

cout<<" the maximun no of char in a file name "<<\_POSIX\_NAME\_MAX

<<endl;

#endif

#ifdef \_POSIX\_OPEN\_MAX

cout<<" the max no of open files per process "<< \_POSIX\_OPEN\_MAX

<<endl;

#endif

break;

case 2:

if(res=sysconf(\_SC\_CLK\_TCK))

cout<<" the no OF CLOCK TICKS ARE "<<res <<endl;

if(res=pathconf("/",\_PC\_PATH\_MAX))

cout<<" the no OF MAX LENGTH "<<res<<endl;

if(res=pathconf("/",\_PC\_PATH\_MAX))

cout<<" the no OF char is file name"<<res<<endl;

if(res=sysconf(\_SC\_OPEN\_MAX))

cout<<" the no of open files per process "<<res<<endl;

}

return 0;

}

**2.race condition**

#include <stdio.h>

#include <stdlib.h>

main()

{

pid\_t pid;

pid=fork();

if(pid<0)

{

printf("Fork Error \n");

}

if(pid==0)

{

charatatime("i am child process \n");

}else

{

charatatime("i am parent process \n");

}

}

Static void charatatime(char \*str)

{

char \*ptr;

int c;

setbuf(stdout,NULL);

for(ptr=str;(c=\*ptr++)!=0;)

putc(c,stdout);

}

**3.resource leak(zombie)**

#ifdef HAVE\_CONFIG\_H

#include <config.h>

#endif

#include <stdio.h>

#include <stdlib.h>

int main()

{

pid\_t pid;

if((pid=fork())<0)

perror("forkerror");

else if(pid==0)

\_exit(0);

sleep(4);

system("ps -o pid,ppid,state,tty,command");

//tty- Print the file name of the terminal connected to standard input.

//ps-The ps command reports information on current running processes, outputting to standard output.

\_exit(0);

}

**4.locked**

#ifdef HAVE\_CONFIG\_H

#include <config.h>

#endif

#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

#include <stdlib.h>

#include <fcntl.h>

int main(int argc, char \*argv[])

{

char temp[1000];

setbuf(stdout,temp);

struct flock fvar;

int fdesc;

char buf;

int rc;

off\_t offset;

pid\_t pid=fork();

fdesc=open(argv[1],O\_RDWR);

fvar.l\_type=F\_WRLCK;

fvar.l\_whence=SEEK\_END;

fvar.l\_start=-100;

fvar.l\_len=100;

if(fcntl(fdesc,F\_SETLK,&fvar)==-1)

{

printf("\n--------------------------------\n");

printf("\n Unable to get lock as file has been locked by:\n");

while(fcntl(fdesc,F\_GETLK,&fvar)==-1 && fvar.l\_type!=F\_UNLCK)

{

printf("\n File : %s is looked by process with pid : %u",argv[1],fvar.l\_pid);

printf(" from %ld th byte in folder for %ld",fvar.l\_start,fvar.l\_len);

printf("number of bytes,for %s \n\n",(fvar.l\_type==F\_WRLCK?"write" : "read"));

if(!fvar.l\_len)

break;

fvar.l\_start+=fvar.l\_len;

fvar.l\_len=0;

}

}

else

{

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

printf("\n\n File: %s was not locked and acquiring of Exclusive Lock was",argv[1]);

printf("successful By Process ID : %u \n",getpid());

offset=lseek(fdesc,-50,SEEK\_END);

printf("\n\n Last 50 bytes of file : %s =\n",argv[1]);

while((rc=read(fdesc,&buf,1))>0)

printf("%c",buf);

fvar.l\_type=F\_UNLCK;

fvar.l\_whence=SEEK\_END;

fvar.l\_start=-100;

fvar.l\_len=100;

if(fcntl(fdesc,F\_SETLKW,&fvar)!=-1)

printf("\n File Unlocked Succesfull\n\n");

}

return 0;

}

**5.inter process communication(client server)**

**Client:**

#include<stdio.h>

#include<unistd.h>

#include<sys/stat.h>

#include<fcntl.h>

#include<string.h>

#define FIFO1 "fifo1"

#define FIFO2 "fifo2"

#define PERMS 0666

char fname[256];

int main()

{

int readfd,writefd;

ssize\_t n;

char buff[512];

printf("Trying to Connect to Server..\n");

writefd=open(FIFO1,O\_WRONLY,0);

readfd=open(FIFO2,O\_RDONLY,0);

printf("Connected...\n");

printf("Enter the filename to request from server:");

scanf("%s",fname);

write(writefd,fname,strlen(fname));

printf("Waiting for Server to reply..\n");

while((n=read(readfd,buff,512))>0) //Read 515 bytes of data contents which is pointed by readfd into FIFO2 buff

write(1,buff,n);

close(readfd);

close(writefd);

return 0;

}

**Server:**

#include<stdio.h>

#include<unistd.h>

#include<sys/stat.h>

#include<fcntl.h>

#include<string.h>

#define FIFO1 "fifo1"

#define FIFO2 "fifo2"

#define PERMS 0666

char fname[256];

int main()

{

int readfd,writefd,fd;

ssize\_t n;

char buff[512];

if(mkfifo(FIFO1,PERMS)<0)

printf("Cant Create FIFO Files\n");

if(mkfifo(FIFO2,PERMS)<0)

printf("Cant Create FIFO Files\n");

printf("Waiting for connection Request...\n");

readfd=open(FIFO1,O\_RDONLY,0);

writefd=open(FIFO2,O\_WRONLY,0);

printf("Connection Established...\n");

read(readfd,fname,255);

printf("Client has requested file %s\n",fname);

if((fd=open(fname,O\_RDWR))<0)

{

strcpy(buff,"File does not exist..\n");

write(writefd,buff,strlen(buff));

}

else

{

while((n=read(fd,buff,512))>0)

write(writefd,buff,n);

}

close(readfd);unlink(FIFO1);

close(writefd);unlink(FIFO2);

}