**Program No:1**

**FORK SYSTEM CALL**

**AIM:** To implement fork() system call

**ALGORITHM:**

step 1: start

step 2: set r=getpid()

step 3: print “parent process id”,r

step 4: set p=fork()

step 5: if p>0 then print “parent process”

step 6: if p=0 then

(i) print child process

(ii) set q=getpid()

(iii) print”ID of child process”

(vi) print 'q'

(v) else if p=-1 then

(vi) print”error”

step 7: stop

**PROGRAM :**

#include<stdio.h>

int main()

{

int p,q,r;

r=getpid();

printf("parent process id:%d",r);

p=fork();

if(p>0)

{

printf("\n parent process");

}

else if(p==0)

{

printf("\n child process");

q=getpid();

printf("child process id %d",q);

}

else if(p==-1)

{

printf("\n error");

}

return(0);

}

**Program No:2**

**INTERPROCESS COMMUNICATION USING ONE PIPE**

**AIM:** To implement inter process communication using one pipe

**ALGORITHM:**

step 1:start

step 2:call pipe(p)

step 3:f=fork()

step 4:setpid=getpid()

step 5:setppid=getppid()

step 6:if f=0 then

(i) print “i am a child process”

(ii) print “my id is ”

(iii) print “enter child message”

(iv) call fgets(a)

(v) call write(a)

(vi) call sleep(1)

(vii) call read(a)

(viii) print “child reads and print parent message”

(ix) call puts(a)

step 7:if f != 0

(i) call read(a)

(ii) print “parent reads and print message”

(iii) call inputs(a)

(iv) print “enter parent message”

(v) call fgets(a)

(vi) write(a)

step 8:stop

**PROGRAM:**

#include<stdio.h>

main()

{

int p[2],pid,ppid,f;

char a[100];

pipe(p);

f=fork();

pid=getpid();

ppid=getppid();

if(f==0)

{

printf("\n child process id:%d",pid);

printf("\n parent process id:%d",ppid);

printf("\n enter the child message");

fgets(a,sizeof(a),stdin);

write(p[1],a,sizeof(a));

sleep(1);

read(p[0],a,sizeof(a));

printf("\n child reads and prints parent message");

puts(a);

}

else

{

read(p[0],a,sizeof(a));

printf("\n parent reads and prints child message");

puts(a);

printf("\n enter parent message");

fgets(a,sizeof(a),stdin);

write(p[1],a,sizeof(a));

}}

**Program No:3**

**FIBINOCCI SERIES USING ONE PIPE**

**AIM:** To implement fibinocci series using one pipe

**ALGORITHM:**

step 1:start

step 2:call pipe(1)

step 3:call pipe(2)

step 4:if fork() ==0 // child process

close p[0]

close p[1]

write the size n into pipe p[1]

step 5: else //parent process

close p[1]

read the size n from the pipe p[0]

set a=0 and b=1

print a and b

check while(n>2)

set c= a+b

print c

set a=b

set b=c

decrement c

step 6:stop

**PROGRAM:**

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

main()

{

int p[2],pid,a,b,c,n;

if(pipe(p)==-1)

{

printf("\n error in pipe connection");

exit(1);

}

pid=fork();

if(pid>0)

{

printf("\nparent process");

printf("\n fibinocci series");

printf("enter the limit");

scanf("%d",&n);

close(p[0]);

write(p[1],&n,sizeof(n));

close(p[1]);

exit(0);

}

else

{

close(p[1]);

read(p[0],&n,sizeof(n));

printf("\n child process");

a=0;

b=1;

close(p[0]);

printf("\n fibinocci series is:");

printf("\n%d\n%d\n",a,b);

while(n>2)

{

c=a+b;

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

a=b;

b=c;

n--;

}

}

}

**Program No:4**

**INTERPROCESS COMMUNICATION USING TWO PIPE**

**AIM:** To implement inter process communication using two pipe

**ALGORITHM:**

step 1:start

step 2:call pipe(1)

step 3:call pipe(2)

step 4:if fork() ==0 // child process

close p1[0]

close p2[1]

write the child message into pipe p1[1]

read the parent message from the pipe p2[0]

step 5:else //parent process

close p1[1]

close p2[0]

read the child message from the pipe p1[0]

write the parent message into the pipe p2[1]

step 6:stop

**PROGRAM:**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

main()

{

char buff[100];

int p1[2],p2[2],i;

pipe(p1);

pipe(p2);

if(fork()==0)

{

printf("\nOpening child");

close(p1[0]);

close(p2[1]);

if(write(p1[1],"Hello",7)<0)

printf("\nChild :write error");

if(read(p2[0],buff,sizeof(buff))<0)

printf("\nchild:read error");

else

{

printf("\nreceived msg from parent: parent says:%s\n",buff);

}

printf("\n closing child\n");

}

else

{

close(p1[1]);

close(p2[0]);

printf("\nopening parent");

if(read(p1[0],buff,sizeof(buff))<0)

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

else

printf("\nreceived msg from child: child says:%s",buff);

if(write(p2[1],"Hi!!",4)<0)

{

printf("\nparent:write error");

}

wait(0);

printf("\n Closing parent");

}

exit(0);

}

**Program No:5**

**SORTING OF AN ARRAY USING TWO PIPE**

**AIM:** To implement sorting of an array using two pipe

**ALGORITHM:**

step 1:Start

step2:call pipe(1)

step 3:call pipe(2)

step 4:set pid=fork()

if pid >0 // child process

close p1[0]

close p2[1]

write the array size into pipe p1[1]

write the array into pipe p1[1]

read the sorted from the pipe p2[0]

print the sorted array in child

step 5:else //parent process

close p1[1]

close p2[0]

read the size of array 'n' and array 'a'from the pipe p1[0]

repeat for i=0 to n-1

repeat for j=0 to n-i-1

check if a[j]>a[j+1]

set temp=a[j]

set a[j]=a[j+1]

set a[j+1]=temp

j=j+1

i=i+1

write the sorted array into the pipe p2[1]

step 6:stop

**PROGRAM:**

#include<stdio.h>

void main()

{

int a[100],buff[100];

int pid,p1[2],p2[2],i,j,n,temp;

pipe(p1);

pipe(p2);

pid=fork();

if(pid>0)

{

printf("parent process\n");

close(p1[0]);

close(p2[1]);

printf("enter the limit");

scanf("%d",&n);

write(p1[1],&n,sizeof(n));

printf("enter the array");

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

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

write(p1[1],buff,sizeof(buff));

sleep(1);

read(p2[0],buff,sizeof(buff));

printf("sorted array is");

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

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

}

else

{

read(p1[0],&n,sizeof(n));

read(p1[0],a,sizeof(a));

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

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

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

{

temp=a[j];

a[j]=a[j+1];

a[j+1]=temp;

}

write(p2[1],a,sizeof(a));

}

}

**Program No:6**

**PALINDROME USING TWO PIPE**

**AIM:** To check whether a number is palindrome or not.

**ALGORITHM:**

step 1:Start

step2:call pipe(1)

step 3:call pipe(2)

step 4:set pid=fork()

if pid >0 // child process

close p1[0]

close p2[1]

write the number to pipe p1[1]

read the parent message from the pipe p2[0]

print the number is palindrome or not

step 5:else //parent process

close p1[1]

close p2[0]

read the number from pipe pipe p1[0]

repeat while(num2>0)

set r=num2%10

set x=x\*10+r

set num2=num2/10

check if the number is palindrome or not and write the corresponding message into the pipe p2[1]

step 6:stop

**PROGRAM:**

#include<stdio.h>

void main()

{

int temp,num1,pid,p1[2],p2[2],num2,r,x=0;

char a[10];

pipe(p1);

pipe(p2);

pid=fork();

if(pid>0)

{

printf("parent process\n");

close(p1[0]);

close(p2[1]);

printf("enter the number");

scanf("%d",&num1);

write(p1[1],&num1,sizeof(num1));

sleep(1);

read(p2[0],&a,sizeof(a));

printf("%s",a);

}

else

{

read(p1[0],&num2,sizeof(num2));

temp=num2;

while(num2>0)

{

r=num2%10;

x=x\*10+r;

num2=num2/10;

}

printf("reversed string %d",x);

if(temp==x)

write(p2[1],"\npalindrome",12);

else

write(p2[1],"\nnot palindrome",16);

}

}

**Program No:7**

**SHARED MEMORY CREATION**

**AIM:** Implement shared memory creation

**ALGORITHM:**

step 1:Start

step 2:Set key\_t key =ftok()

step 3:Set shmid = call shmget()

step 4: Combine the space allocated to the existing space of process

step 5:if shmid != -1 then

print” shared memory created with id”

step 6:else

print”shared memory not created”

step7 :stop

**PROGRAM:**

#include<sys/shm.h>

#include<sys/types.h>

#include<sys/ipc.h>

#include<stdio.h>

main()

{

char \*a;

key\_t key=ftok("shm.c",'A');

int shmid=shmget(key,sizeof(char),IPC\_CREAT|0666);

a=(char \*)shmat(shmid,0,0);

if(shmid!=-1)

printf("Shared memory created with id:%d",shmid);

else

printf("Shared memory not created");

}

**Program No:8**

**INTERPROCESS COMMUNICATION USING MESSAGE QUEUE**

**AIM:** To implement interprocess communication using message queue.

**ALGORITHM:**

**Server:**

step 1:Start

step 2:Declare a structure with members mtype and mtext

step 3:Set id = call msgget()

step 4:Call the function msg rev ()

step 5:Print buf.mtext

step 6: stop

**Client:**

step 1:Start

step 2:Declare a structure with members mtype and mtext

step 3:Set id =call msgget()

step 4: Set message into buf.mtext

step 5:call Msg snd()

step 6:Stop

**PROGRAM:**

**Server:**

#include<sys/stat.h>

#include<sys/msg.h>

#include<stdio.h>

struct

{

long int mtype;

char mtext [100];

}buf;

main()

{

int id;

id=msgget(4589,IPC\_CREAT|0666);

msgrcv(id,&buf,sizeof(buf),1,0);

printf("%s",buf.mtext);

}

**Client:**

#include<sys/stat.h>

#include<sys/msg.h>

struct

{

long int mtype;

char mtext [100];

}buf;

main()

{

int id;

id=msgget(4589,IPC\_CREAT|0666);

buf.mtype=1;

gets (buf.mtext);

msgsnd(id,&buf,sizeof(buf),0);

}

**Program No:9**

**BANKER'S ALGORITHM**

**AIM:** To implement Banker's algorithm

**ALGORITHM:**

step 1: start

step 2: count=0

step 3: Read the no :of process,p

step 4: Read the no :of resources,r

step 5: Read the max matrix for each process, max[i][j]

step 6: Read the allocation matrix for each process ,alloc[i][j]

step 7: Read the available resource matrix ,avail[i][j]

step 8 : Find the need matrix ,need[i][j]

step 9: For each incompleted process pi

check whether need[i][j] < avail[i][j]

if then allocate resources to the process

increment Safe sequence

step 10: If safe sequence ==p then print safe state else unsafe state

Step 11 : stop

**PROGRAM :**

#include<stdio.h>

int main()

{

int max[10][10],need[10][10],alloc[10][10],avail[10],completed[10],safesequence[10];

int p,r,i,j,process,count;

count=0;

printf("\n\nEnter the no of process");

scanf("%d",&p);

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

completed[i]=0;

printf("\n\Enter the no of resources");

scanf("%d",&r);

printf("\n\nEnter the max. matrix for each process");

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

{

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

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

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

}

printf("\n\nEnter the allocation for each process");

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

{

printf("\nFor process %D:",i+1);

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

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

}

printf("\n\nEnter the available resource");

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];

do

{

printf("\nmax matrix:\t allocation matrix:\n");

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

{

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

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

printf("\t\t");

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

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

printf("\n");

}

process=-1;

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

{

if(completed[i]==0)

{

process=i;

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

{

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

{

process=-1;

break;

}

}

}

if(process!=-1)

{

printf("\nProcess %d runs to completion",process+1);

safesequence[count]=process+1;

count++;

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

{

avail[j]+=alloc[process][j];

alloc[process][j]=0;

max[process][j]=0;

completed[process]=1;

}

}

}

}while(count!=p&&process!=-1);

if(count==p)

{

printf("\nThe system is in safe state!\n");

printf("safe sequence:<");

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

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

printf(">\n");

}

else

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

return(0);

}

**Program No:10**

**DINING PHILOSOPHER PROBLEM**

**AIM:** Implement dining philosopher problem

**ALGORITHM:**

The algorithm is the for the process,

step 1: start

step 2: intialize the thread variables ,thread attributes ,semaphores as required

step 3: create the threads representing philosopher.

step 4: wait until the thread function execution.

Step 5: stop

The algorithm for the philosopher is,

step 1: Start

step 2: wait for the left fork

step 3: wait for the right fork

step 4:start eating

Step 5:release the left fork

Step 6:release the right fork

Step 7: Stop

**PROGRAM:**

#include<stdio.h>

#include<pthread.h>

#include<semaphore.h>

#define cls() printf("\033[H\033[J")

#define EATINGTIME 1

void \*philosopher1();

void \*philosopher2();

void \*philosopher3();

void \*philosopher4();

void \*philosopher5();

sem\_t sem15,sem12,sem23,sem34,sem45;

int end=0;

main()

{ char a[2];

pthread\_t t1,t2,t3,t4,t5;

pthread\_attr\_t at1;

pthread\_attr\_init(&at1);

pthread\_attr\_setdetachstate(&at1,PTHREAD\_CREATE\_DETACHED);

sem\_init(&sem15,0,1);

sem\_init(&sem12,0,1);

sem\_init(&sem23,0,1);

sem\_init(&sem34,0,1);

sem\_init(&sem45,0,1);

cls();

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

printf("........");

printf("\n\n\nDINING PHILOSOPHER PROBLEM");

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

printf("\n\n\n\t No. OF PHILOSOPHER:5");

printf("\n\n\t\tNo. OF FORK:5\n\n");

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

printf("\n\n\nPRESS ENTER TO CONTINUE....");

fgets(a,sizeof(a),stdin);

cls();

pthread\_create(&t1,&at1,philosopher1,NULL);

pthread\_create(&t2,&at1,philosopher2,NULL);

pthread\_create(&t3,&at1,philosopher3,NULL);

pthread\_create(&t4,&at1,philosopher4,NULL);

pthread\_create(&t5,&at1,philosopher5,NULL);

while(end!=5)

{

}

}

void \*philosopher1()

{

int i=0;

printf("\n\t\t PHILOSOPHER\_1 THINKING");

while(i<EATINGTIME)

{

sleep(1);

sem\_wait(&sem15);

sem\_wait(&sem12);

printf("\n\n\t \*PHILOSOPHER\_1 EATING\n");

sleep(1);

sem\_post(&sem15);

sem\_post(&sem12);

printf("\n\t\tPHILOSOPHER\_1 THINKING\n");

i++;

}

end++;

}

void \*philosopher2()

{

int i=0;

printf("\n\t\t PHILOSOPHER\_2 THINKING");

while(i<EATINGTIME)

{

sleep(1);

sem\_wait(&sem12);

sem\_wait(&sem23);

printf("\n\n\t \*PHILOSOPHER\_2 EATING\n");

sleep(1);

sem\_post(&sem12);

sem\_post(&sem23);

printf("\n\t\tPHILOSOPHER\_2 THINKING\n");

i++;

}

end++;

}

void \*philosopher3()

{

int i=0;

printf("\n\t\t PHILOSOPHER\_3 THINKING");

while(i<EATINGTIME)

{

sleep(1);

sem\_wait(&sem23);

sem\_wait(&sem34);

printf("\n\n\t \*PHILOSOPHER\_3 EATING\n");

sleep(1);

sem\_post(&sem23);

sem\_post(&sem34);

printf("\n\t\tPHILOSOPHER\_3 THINKING\n");

i++;

}

end++;

}

void \*philosopher4()

{

int i=0;

printf("\n\t\t PHILOSOPHER\_4 THINKING");

while(i<EATINGTIME)

{

sleep(1);

sem\_wait(&sem34);

sem\_wait(&sem45);

printf("\n\n\t \*PHILOSOPHER\_4 EATING\n");

sleep(1);

sem\_post(&sem34);

sem\_post(&sem45);

printf("\n\t\tPHILOSOPHER\_4 THINKING\n");

i++;

}

end++;

}

void \*philosopher5()

{

int i=0;

printf("\n\t\t PHILOSOPHER\_5 THINKING");

while(i<EATINGTIME)

{

sleep(1);

sem\_wait(&sem45);

sem\_wait(&sem15);

printf("\n\n\t \*PHILOSOPHER\_5 EATING\n");

sleep(1);

sem\_post(&sem45);

sem\_post(&sem15);

printf("\n\t\tPHILOSOPHER\_5 THINKING\n");

i++;

}

end++;

}

**Program No:11**

**REMOTE PROCEDURE CALL**

**AIM:** Implement Remote procedure call

**ALGORITHM:**

**sum.x**

step 1: start

step 2: declare globally used variables

step 3: intialize program number,version number and procedure number.

**RPC client**

step 1: start

step 2: create client

step 3: calling function at server.

Step 4: stop

**RPC server**

step 1: start

step 2: receiving request from client

step 3: execute the request

step 4: return the result

step 5: stop

**PROGRAM:**

**Sum.x**

struct operands

{

int x;

int y;

};

program sum

{

version sum\_ver

{

int ADD(operands)=1;

}=1;

}=8545555;

**Rpc client:**

#include<stdio.h>

#include<stdlib.h>

#include "sum.h"

int add(CLIENT \*clnt,int x,int y)

{

operands ops;

int \*result;

ops.x=x;

ops.y=y;

result=add\_1(&ops,clnt);

return(\*result);

}

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

{

CLIENT \*clnt;

int x,y;

clnt=clnt\_create(argv[1],sum,sum\_ver,"UDP");

x=atoi(argv[2]);

y=atoi(argv[3]);

printf("\n\n%d+%d=%d\n\n",x,y,add(clnt,x,y));

}

**Rpc server**

#include<stdio.h>

#include<rpc/rpc.h>

#include "sum.h"

int \*add\_1\_svc(operands \*argp,struct svc\_req \*reqstp)

{

static int result;

printf("got request:adding %d,%d\n",argp->x,argp->y);

result=argp->x+argp->y;

return(&result);

}

**Program No:12**

**TCP PROTOCOL**

**AIM:** To implement TCP protocol

**ALGORITHM:**

**SERVER**

Step 1: Start

Step 2: Create socket

Step 3: Bind it with socket address structure

Step 4: Accept connection

Step 5: Read/Write message to client

Step 6: Close the socket

Step 7:Stop

**CLIENT**

Step 1: Start

Step 2: Create a stream socket

Step 3: Receive/Send message to the server

Step 4: Close the socket

Step 5:Stop

**PROGRAM :**

**Tcp server**

#include<stdio.h>

#include<sys/types.h>

#include<netdb.h>

#include<netinet/in.h>

#include<sys/socket.h>

#include<unistd.h>

int main()

{

int clntsocket,serversocket;

char msg[50],msg1[50];

struct sockaddr\_in serveraddr,clntaddr;

socklen\_t len;

bzero(msg,sizeof(msg));

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

printf("\n SERVER PROCESS\n");

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

serveraddr.sin\_family=AF\_INET;

serveraddr.sin\_port=htons(5015);

serveraddr.sin\_addr.s\_addr=192.168.20.1;

bind(serversocket,(struct sockaddr \*)&serveraddr,sizeof(serveraddr));

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

len=sizeof(clntaddr);

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

listen(serversocket,5);

printf("\nWaiting for client connectivity\n");

clntsocket=accept(serversocket,(struct sockaddr\*)&clntaddr,&len);

printf("\nClient connectivity is received\n");

printf("\nReading message from client\n");

read(clntsocket,msg,sizeof(msg));

printf("\nClient message is \t%s",msg);

printf("\nsending acknowledgement to client\n");

write(clntsocket,"your msg is received",sizeof("your msg is received"));

close(clntsocket);

close(serversocket);

}

**Tcp client**

#include<stdio.h>

#include<sys/types.h>

#include<netdb.h>

#include<netinet/in.h>

#include<sys/socket.h>

#include<unistd.h>

int main()

{

int clntsocket;

struct sockaddr\_in serveraddr;

socklen\_t len;

struct hostent \*server;

char msg[50],msg1[50];

bzero(msg,sizeof(msg));

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

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

len=sizeof(serveraddr);

serveraddr.sin\_family=AF\_INET;

serveraddr.sin\_port=htons(5015);

server=gethostbyname("192.168.0.2");

bcopy((char \*)server->h\_addr,(char \*)&serveraddr.sin\_addr.s\_addr,sizeof(server->h\_addr));

connect(clntsocket,(struct sockaddr \*)&serveraddr,sizeof(serveraddr));

printf("\nConnection process\n");

printf("\nsending message to server\n");

write(clntsocket,"i am client",sizeof("i am client"));

printf("\nreceiving acknowledgement from server\n");

read(clntsocket,msg1,sizeof(msg1));

printf("\nacknowledgement is %s",msg1);

close(clntsocket);

return(0);

}

**Program No:13**

**UDP PROTOCOL**

**AIM:** To implement UDP protocol

**ALGORITHM:**

**SERVER**

Step 1: Start

Step 2: Create a datagram socket

Step 3: Bind the socket to the operating system

Step 4: Receiving/sending process

step 5: Close the socket

Step 6: Stop

**CLIENT**

Step 1: Start

Step 2: Create a datagram socket

Step 3: Receive/Send message to the server

Step 4: close the socket

Step 5: Stop

**PROGRAM**

**Udp server:**

#include<sys/socket.h>

#include<netinet/in.h>

#include<arpa/inet.h>

#include<stdio.h>

main()

{

struct sockaddr\_in server,client;

int s,n;

char a[10],b[50];

s=socket(AF\_INET,SOCK\_DGRAM,0);

server.sin\_port=1240;

server.sin\_family=AF\_INET;

server.sin\_addr.s\_addr=inet\_addr("127.0.0.1");

bind(s,(struct sockaddr\*)&server,sizeof(server));

n=sizeof(client);

while(1)

{

printf("\nreceived from client");

recvfrom(s,a,sizeof(a),0,(struct sockaddr\*)&client,&n);

printf("%s",a);

printf("\n send to client");

scanf("%s",b);

if(strcmp(b,"close")==0)

break;

sendto(s,b,sizeof(b),0,(struct sockaddr\*)&client,n);

}

}

**Udp client:**

#include<sys/socket.h>

#include<netinet/in.h>

#include<arpa/inet.h>

#include<stdio.h>

main()

{

struct sockaddr\_in server;

int s,n;

char a[10],b[50];

s=socket(AF\_INET,SOCK\_DGRAM,0);

server.sin\_port=1240;

server.sin\_family=AF\_INET;

server.sin\_addr.s\_addr=inet\_addr("127.0.0.1");

while(1)

{

printf("\nclient");

scanf("%s",b);

if(strcmp(b,"close")==0)

break;

sendto(s,b,sizeof(b),0,(struct sockaddr\*)&server,sizeof(server));

printf("\nclient2 ");

recvfrom(s,a,sizeof(a),0,NULL,NULL);

printf("%s",a);

}

}

**Program No:14**

**SORTING OF AN ARRAY USING TCP PROTOCOL**

**AIM:** To sort an array using TCP protocol

**ALGORITHM:**

**SERVER**

Step 1: Start

Step 2: Create socket

Step 3: Bind it with socket address structure

Step 4: Accept connection

Step 5: Read/Write message to client

Step 6: Close the socket

Step 7:Stop

**CLIENT**

Step 1: Start

Step 2: Create a stream socket

Step 3: Receive/Send message to the server

Step 4: Close the socket

Step 5:Stop

**PROGRAM:**

**Server:**

#include<stdio.h>

#include<sys/types.h>

#include<netdb.h>

#include<netinet/in.h>

#include<sys/socket.h>

#include<unistd.h>

int main()

{

int clntsocket,serversocket,a[50],i,temp,j,n;

char msg[50];

struct sockaddr\_in serveraddr,clntaddr;

socklen\_t len;

bzero((char \*)msg,sizeof(msg));

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

printf("\nSERVER PROCESS\n");

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

serveraddr.sin\_family=AF\_INET;

serveraddr.sin\_port=htons(5015);

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

bind(serversocket,(struct sockaddr\*)&serveraddr,sizeof(serveraddr));

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

len=sizeof(clntaddr);

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

listen(serversocket,5);

printf("\nwaitng for client connectivity\n");

clntsocket=accept(serversocket,(struct sockaddr\*)&clntaddr,&len);

printf("\nclient connectivity is received\n");

printf("\nreading array size from client\n");

read(clntsocket,&n,sizeof(n));

printf("\narray size is:%d\n",n);

printf("\nreading array from client\n");

read(clntsocket,&a,sizeof(a));

printf("\narray is:\n");

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

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

printf("\nsending acknowledgement to client\n");

write(clntsocket,"your message is received",sizeof("your message is received"));

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

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

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

{

temp=a[j];

a[j]=a[j+1];

a[j+1]=temp;

}

printf("\nsorted array is:\n");

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

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

close(clntsocket);

close(serversocket);

}

**Client:**

#include<stdio.h>

#include<sys/types.h>

#include<netdb.h>

#include<netinet/in.h>

#include<sys/socket.h>

#include<unistd.h>

int main()

{

int clntsocket,i,a[10],n;

struct sockaddr\_in serveraddr;

socklen\_t len;

struct hostent \*server;

char msg[50];

bzero(msg,sizeof(msg));

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

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

len=sizeof(serveraddr);

serveraddr.sin\_family=AF\_INET;

serveraddr.sin\_port=htons(5015);

server=gethostbyname("127.0.0.1");

bcopy((char \*)server->h\_addr,(char \*)&serveraddr.sin\_addr.s\_addr,sizeof(server->h\_addr));

connect(clntsocket,(struct sockaddr\*)&serveraddr,sizeof(serveraddr));

printf("\nconnection process\n");

printf("\nenter array size: \n");

scanf("%d",&n);

printf("\nsending array size to server\n");

write(clntsocket,&n,sizeof(n));

printf("\nenter 5 numbers: \n");

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

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

printf("\nsending array to server\n");

write(clntsocket,&a,sizeof(a));

printf("\nreceiving acknowledgement from server\n");

read(clntsocket,msg,sizeof(msg));

printf("\nacknowledgement is %s\n",msg);

close(clntsocket);

return(0);

}

**Program No:15**

**STOP & WAIT USING PROTOCOL**

**AIM:** To implement Stop& Wait using TCP protocol

**ALGORITHM:**

**SENDER**

Step 1: Start

Step 2: Create socket

Step 3: Bind it with socket address structure

Step 4: Accept connection

Step 5: Sending frame to the receiver

Step 6: Receiving the acknowledgement

If the acknowledgement is “ACK” or “ack” then send the next frame

otherwise send the previous frame once again

Step 7: Close the socket

Step 8:Stop

**RECEIVER:**

Step 1: Start

Step 2: Create a stream socket

Step 3: receiving frame from the send

Step 4: Send the acknowledgement (ACK or NACK) to sender

Step 5: Close the socket

Step 6:Stop

**PROGRAM:**

**Sender:**

#include"stdio.h"

#include"sys/types.h"

#include"netinet/in.h"

#include"string.h"

#include"sys/socket.h"

#include"stdlib.h"

#include"unistd.h"

main()

{

int sd,i,r,bi,nsd,port,frame,prev\_frame=0,count=0;

char ack[5],buff[30];

struct sockaddr\_in ser,cli;

if((sd=socket(AF\_INET,SOCK\_STREAM,IPPROTO\_TCP))==-1)

{

printf("\nsocket problem");

return 0;

}

printf("\n socket created\n");

bzero((char \*)&cli,sizeof(ser));

printf("ENTER PORT NUMBER\n");

scanf("%d",&port);

printf("\n port address is %d\n:",port);

ser.sin\_family=AF\_INET;

ser.sin\_port=htons(port);

ser.sin\_addr.s\_addr=htonl(INADDR\_ANY);

bi=bind(sd,(struct sockaddr\*)&ser,sizeof(ser));

if(bi==-1)

{

printf("\n bind error,port busy, plz change port in client and server");

return 0;

}

i=sizeof(cli);

listen(sd,5);

nsd=accept(sd,((struct sockaddr\*)&cli),&i);

if(nsd==-1)

{

printf("\ncheck the description parameter\n");

return 0;

}

printf("\n connection accepted\n");

while(count<5)

{

ph:

printf("\n sending FRAME %d to the receiver.......\n",prev\_frame);

snprintf(buff,sizeof(buff),"%d",prev\_frame);

send(nsd,buff,30,0);

r=recv(nsd,ack,5,0);

if(strcmp(ack,"ack")==0||strcmp(ack,"ACK")==0)

{

count++;

if(prev\_frame==0)

prev\_frame=1;

else

prev\_frame=0;

}

else if(strcmp(ack,"nak")==0||strcmp(ack,"NAK")==0)

{

printf("\n NAK :so again send the previous frame....\n");

goto ph;

}

}

printf("\n bye");

send(nsd,"EOF",4,0);

close(sd);

close(nsd);

return 0;

}

**Receiver:**

#include"stdio.h"

#include"sys/types.h"

#include"netinet/in.h"

#include"string.h"

#include"sys/socket.h"

#include"stdlib.h"

#include"unistd.h"

int main()

{

int sd,con,port,i;

char content[30],ack[3];

struct sockaddr\_in cli;

if((sd=socket(AF\_INET,SOCK\_STREAM,IPPROTO\_TCP))==-1)

{

printf("\nsocket problem");

return 0;

}

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

cli.sin\_family=AF\_INET;

printf("ENTER PORT NUMBER:\n");

scanf("%d",&port);

cli.sin\_port=htons(port);

cli.sin\_addr.s\_addr=htonl(INADDR\_ANY);

con=connect(sd,(struct sockaddr\*)&cli,sizeof(cli));

if(con==-1)

{

printf("\n connection error");

return 0;

}

i=recv(sd,content,30,0);

while(strcmp(content,"EOF")!=0)

{

printf("received from sender:frame %s \n",content);

ph:

printf("\n acknowledgement(ACK/NAK):");

scanf("%s",ack);

if(!(strcmp(ack,"ack")==0||strcmp(ack,"nak")==0||strcmp(ack,"ACK")||strcmp(ack,"NAK")==0))

{

printf("\n not a valid acknowledgement.....use ACK or NAK....\n");

goto ph;

}

send(sd,ack,5,0);

i=recv(sd,content,30,0);

}

printf("\n bye");

close(sd);

return 0;

}