

INDEX

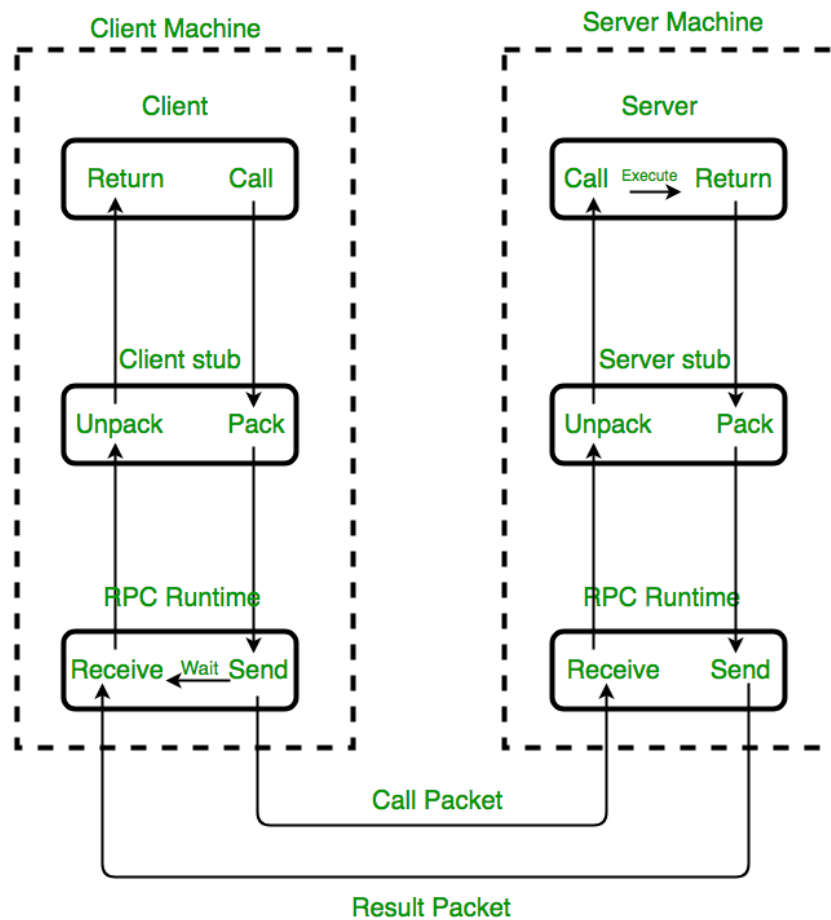
Experiment No.	Experiment Title	Signature
1	Implement Client Server based program using RPC.	
2.	Implement Christian's Algorithm of physical clock synchronization	
3.	Implement Berkeley's Algorithm of physical clock synchronization	
4.	Implement unix system calls(Fork, exit, opendir, readdir).	
5.	Implement Lamport's Logical Clock .	
6.	Implement Vector Logical Clock .	
7.	Implement BSS for causal ordering of messages.	
8.	Implement Bully's Election algorithm.	
9.	Implementation of Ring based Election algorithm.	
10.	Implementation of Huang's Algorithm	

PRACTICAL - 1

Ques. Implement Client Server based program using RPC.

OBJECTIVE: **Remote Procedure Call (RPC)** is a powerful technique for constructing **distributed, client-server based applications**. It is based on extending the conventional local procedure calling so that the **called procedure need not exist in the same address space as the calling procedure**. The two processes may be on the same system, or they may be on different systems with a network connecting them.

Diagram:



Implementation of RPC mechanism

SERVER:

```
#include <string.h>
```

```

#include <arpa/inet.h>
#include <stdio.h>
#include <stdlib.h>

#define MYPOR 3490
#define SIZE_TO_SEND 1000
#define MY_IP "127.0.0.1"

int main(int argc, char *argv[]) {
    int sockfd,sockfd2;
    char tosend = 's'; //a char (1byte) to send to receivers
    char ack;
    struct sockaddr_in my_addr,rcvr_addr;
    struct timeval start,end;
    int sin_size = sizeof(my_addr),i,k,num_packet_sent,optval;
    double t1,t2;

    //open TCP socket,bind and accept RECEIVERS connections
    if( (sockfd = socket(PF_INET, SOCK_STREAM, 0)) ==-1){
        perror("socket error");
        exit(1);
    }
    my_addr.sin_family = AF_INET;
    my_addr.sin_port = htons(3490);
    my_addr.sin_addr.s_addr = inet_addr(MY_IP);
    memset(my_addr.sin_zero, '\0', sizeof(my_addr.sin_zero));
    //allow reuse of port
    optval = 1;
    if (setsockopt(sockfd,SOL_SOCKET,SO_REUSEADDR,&optval,sizeof(int)) == -
1) {
        perror("setsockopt");
        exit(1);
    }
    //bind(socketfd, struct about my address,sizeofmy address);
    if (bind(sockfd, (struct sockaddr *)&my_addr, sizeof my_addr) == -1) {
        perror("bind");
        exit(1);
    }
    listen(sockfd,10);

    sockfd2 = accept(sockfd, (struct sockaddr *)&rcvr_addr, &sin_size);

    //connections OK
    //send 100 packet of size 1 byte and for each send wait for ack
    t1=0.0; t2=0.0;
    printf("Sending 100000 messages 1 byte each and wait for ack.\n");
    for(num_packet_sent=0;num_packet_sent<100000;num_packet_sent++){
        if(gettimeofday(&start,NULL)) {
            printf("time failed\n");
            exit(1);
        }
        send(sockfd2,&tosend,sizeof(char),0);
        optval=recv(sockfd2,&ack,sizeof(char),0);
        if(optval==-1) {
            perror("Receive error");
            exit(1);
        }
    }
}

```

```

else{
    if(gettimeofday(&end,NULL)) {
        printf("time failed\n");
        exit(1);
    }
    t1+=start.tv_sec+(start.tv_usec/1000000.0);
    t2+=end.tv_sec+(end.tv_usec/1000000.0);
}
}
//calculate and print mean rtt
printf("RTT = %g ms\n", (t2-t1)/1000000);
printf("close sockets and exit\n");
shutdown(sockfd2,2);
shutdown(sockfd,2);
exit(0);
}

```

CLIENT:

```

#include <string.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <stdlib.h>

#define SENDER_PORT 3490
#define SENDER_IP "127.0.0.1"

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

    int sockfd;
    int rcv_num, loop_count, i;
    char buf;
    struct sockaddr_in sender_addr;

    //open socket and connect
    if( (sockfd = socket(PF_INET, SOCK_STREAM, 0)) == -1){
        perror("socket error"); // do some error checking!
        exit(1);
    }

    sender_addr.sin_family = AF_INET;
    sender_addr.sin_port = htons(3490);
    sender_addr.sin_addr.s_addr = inet_addr(SENDER_IP);
    memset(sender_addr.sin_zero, '\0', sizeof(sender_addr.sin_zero));

    if ((connect(sockfd, (struct sockaddr
*)&sender_addr, sizeof(sender_addr))) == -1){
        perror("connect error"); // do some error checking!
        exit(1);
    }

    //connection established
    printf("Connection to sender established\n");
    //reads 100 packets of 1 byte and sends them back as ack packets
    printf("Receive 100000 packets of 1 byte and send them back\n");
}

```

```

for(i=0;i<100000;i++){
    rcv_num = recv(sockfd,&buf,sizeof(char),0);
    if(rcv_num!=0) {
        //send  ack
        send(sockfd,&buf,sizeof(char),0);
    }
    else{
        perror("Receive error");
        exit(1);
    }
}
printf("\tDone\nClose socket and exit\n");
close(sockfd);
exit(0);
}

```

SERVER:

```

pclose
tfcsed@tfcsed:~/Desktop/IIITU16105/EXP1.2$ gcc server2.c
server2.c: In function 'main':
server2.c:48:6: warning: implicit declaration of function 'gettimeofday' [-Wimpl
icit-function-declaration]
    if(gettimeofday(&start,NULL)) {
    ~~~~~
tfcsed@tfcsed:~/Desktop/IIITU16105/EXP1.2$ ./a.out
Sending 100000 messages 1 byte each and wait for ack.
RTT = 1.3125e-05 ms
close sockets and exit
tfcsed@tfcsed:~/Desktop/IIITU16105/EXP1.2$ |

```

CLIENT:

```

tfcsed@tfcsed:~/Desktop/IIITU16105/EXP1.2$ gcc client2.c
client2.c: In function 'main':
client2.c:48:2: warning: implicit declaration of function 'close'; did you mean
'pclose'? [-Wimplicit-function-declaration]
    close(sockfd);
    ~~~~~
pclose
tfcsed@tfcsed:~/Desktop/IIITU16105/EXP1.2$ ./a.out
Connection to sender established
Receive 100000 packets of 1 byte and send then back
Done
Close socket and exit
tfcsed@tfcsed:~/Desktop/IIITU16105/EXP1.2$ |

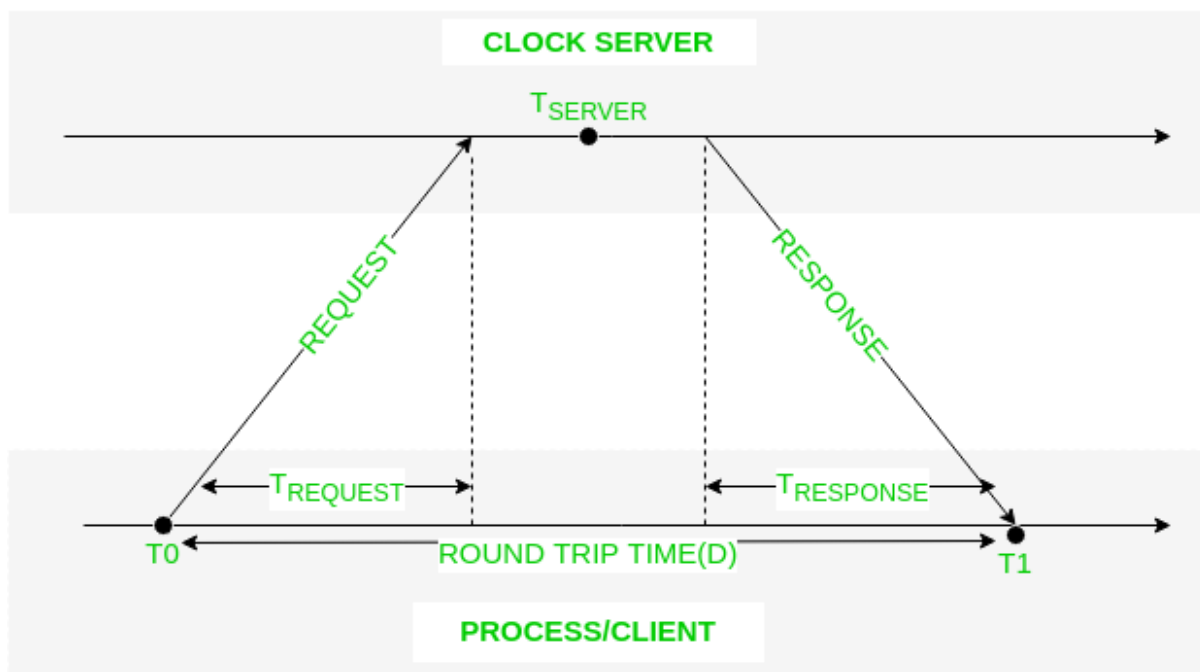
```

PRACTICAL - 2

Ques. Implement Christian's Algorithm of physical clock synchronization.

OBJECTIVE: Christian's Algorithm is a clock synchronization algorithm is used to synchronize time with a time server by client processes. This algorithm works well with low-latency networks where Round Trip Time is short as compared to accuracy while redundancy prone distributed systems/ applications do not go hand in hand with this algorithm. Here Round Trip Time refers to the time duration between start of a Request and end of corresponding Response.

Diagram:



SERVER:

```
import socket
import datetime

def initiateClockServer():

    s = socket.socket()
    print("Socket successfully created")

    port = 8000
```

```

s.bind(('', port))

s.listen(5)
print("Socket is listening...")

while True:

    connection, address = s.accept()
    print('Server connected to', address)

    connection.send(str(datetime.datetime.now()).encode())

    connection.close()

if __name__ == '__main__':

    initiateClockServer()

```

CLIENT:

```

import socket
import datetime
from dateutil import parser
from timeit import default_timer as timer

def synchronizeTime():

    s = socket.socket()

    port = 8000

    s.connect(('127.0.0.1', port))

    request_time = timer()

    server_time = parser.parse(s.recv(1024).decode())
    response_time = timer()
    actual_time = datetime.datetime.now()

    print("Time returned by server: " + str(server_time))

    process_delay_latency = response_time - request_time

    print("Process Delay latency: " + str(process_delay_latency) + "
seconds")

    print("Actual clock time at client side: " + str(actual_time))

    client_time = server_time + datetime.timedelta(seconds =
(process_delay_latency) / 2)

    print("Synchronized process client time: " + str(client_time))

```

```
error = actual_time - client_time
print("Synchronization error : "
      + str(error.total_seconds()) + " seconds")
```

```
s.close()
```

```
if __name__ == '__main__':
```

```
    synchronizeTime()
```

SERVER:

```
tfcscd@tfcscd:~/Desktop/IIITU16105/EXP2.1$ python server.py
File "server.py", line 25
    connection, address = s.accept()
    ^
IndentationError: expected an indented block
tfcscd@tfcscd:~/Desktop/IIITU16105/EXP2.1$ python server.py
Socket successfully created
Socket is listening...
('Server connected to', ('127.0.0.1', 42054))
```

CLIENT:

```
tfcscd@tfcscd:~/Desktop/IIITU16105/EXP2.1$ python client.py
Traceback (most recent call last):
  File "client.py", line 3, in <module>
    from dateutil import parser
ImportError: No module named dateutil
tfcscd@tfcscd:~/Desktop/IIITU16105/EXP2.1$ python client.py
Time returned by server: 2019-10-18 22:30:41.473222
Process Delay latency: 0.000967025756836 seconds
Actual clock time at client side: 2019-10-18 22:30:41.474064
Synchronized process client time: 2019-10-18 22:30:41.473706
Synchronization error : 0.000358 seconds
tfcscd@tfcscd:~/Desktop/IIITU16105/EXP2.1$ |
```

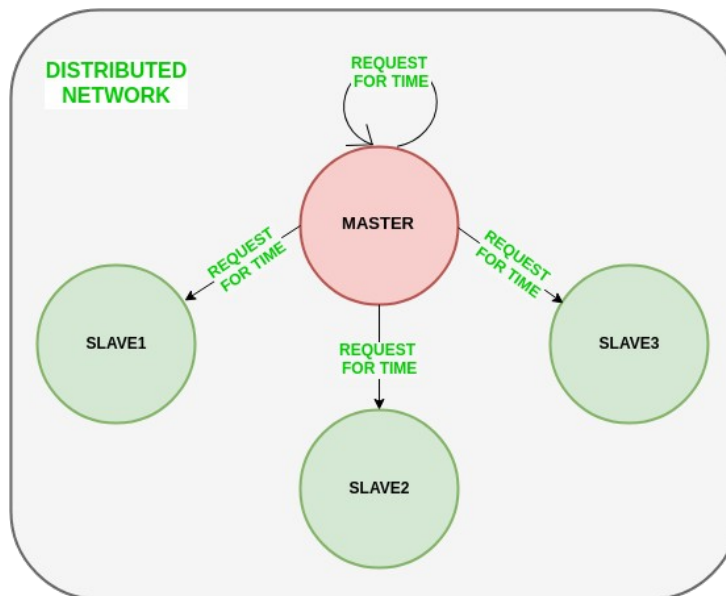

PRACTICAL - 3

Ques. Implement Berkeley's Algorithm of physical clock synchronization.

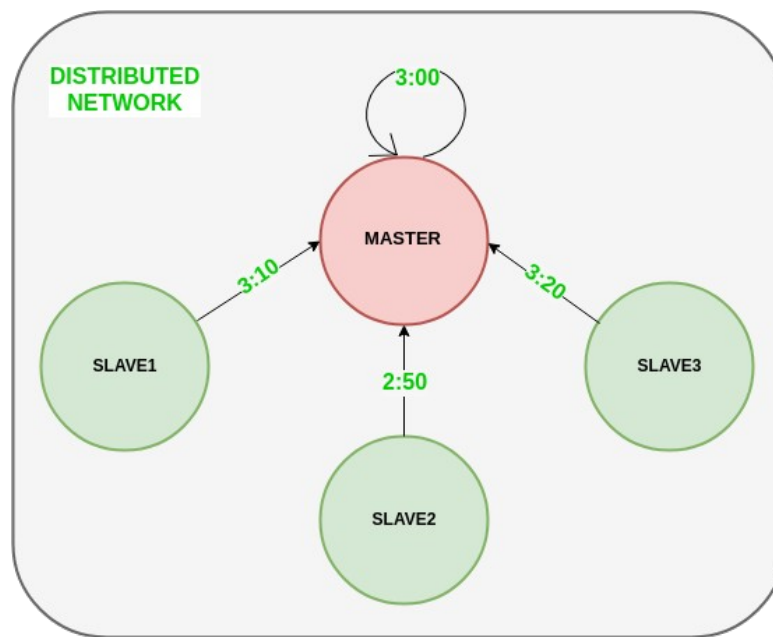
Objective: Berkeley's Algorithm is a clock synchronization technique used in distributed systems. The algorithm assumes that each machine node in the network either doesn't have an accurate time source or doesn't possess an UTC server.

Berkeley's Diagram:

Master sends request to slave nodes.



Slave nodes send back time given by their system clocks.



SERVER:

```
from functools import reduce
from dateutil import parser
import threading
import datetime
import socket
import time

client_data = {}

''' nested thread function used to receive
    clock time from a connected client '''
def startRecieveingClockTime(connector, address):

    while True:
        clock_time_string = connector.recv(1024).decode()
        clock_time = parser.parse(clock_time_string)
        clock_time_diff = datetime.datetime.now() - clock_time

        client_data[address] = {
            "clock_time"      : clock_time,
            "time_difference" : clock_time_diff,
            "connector"       : connector
        }

        print("Client Data updated with: " + str(address))
        time.sleep(5)

''' master thread function used to open portal for
    accepting clients over given port '''
def startConnecting(master_server):

    while True:

        master_slave_connector, addr = master_server.accept()
        slave_address = str(addr[0]) + ":" + str(addr[1])

        print(slave_address + " got connected successfully")

        current_thread = threading.Thread( target =
startRecieveingClockTime, args = (master_slave_connector, slave_address, ))
        current_thread.start()
```

```

def getAverageClockDiff():

    current_client_data = client_data.copy()

    time_difference_list = list(client['time_difference'] for client_addr,
client in client_data.items())

    sum_of_clock_difference =
sum(time_difference_list,datetime.timedelta(0, 0))

    average_clock_difference = sum_of_clock_difference / len(client_data)

    return average_clock_difference

''' master sync thread function used to generate
cycles of clock synchronization in the network '''
def synchronizeAllClocks():

    while True:

        print("New synchroniztion cycle started.")
        print("Number of clients to be synchronized: " +
str(len(client_data)))

        if len(client_data) > 0:

            average_clock_difference = getAverageClockDiff()

            for client_addr, client in client_data.items():
                try:
                    synchronized_time =
datetime.datetime.now() + average_clock_difference

                    client['connector'].send(str(

synchronized_time).encode())

                except Exception as e:
                    print("Something went wrong while " +
"sending synchronized time " + "through " + str(client_addr))

            else :
                print("No client data." + " Synchronization not
applicable.")

                print("\n\n")

                time.sleep(5)

def initiateClockServer(port = 8080):

    master_server = socket.socket()
    master_server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

```

```

    print("Socket at master node created successfully\n")

    master_server.bind(('', port))

    master_server.listen(10)
    print("Clock server started...\n")

    print("Starting to make connections...\n")
    master_thread = threading.Thread( target = startConnecting,args =
(master_server, ))
    master_thread.start()

    print("Starting synchronization parallely...\n")
    sync_thread = threading.Thread( target = synchronizeAllClocks,args =
())
    sync_thread.start()

if __name__ == '__main__':

    initiateClockServer(port = 8080)

```

CLIENT:

```

from timeit import default_timer as timer
from dateutil import parser
import threading
import datetime
import socket
import time

def startSendingTime(slave_client):

    while True:
        slave_client.send(str(datetime.datetime.now()).encode())

        print("Recent time sent successfully")
        time.sleep(5)

def startReceivingTime(slave_client):

    while True:
        Synchronized_time =
parser.parse( slave_client.recv(1024).decode())

        print("Synchronized time at the client is: " +
str(Synchronized_time))

def initiateSlaveClient(port = 8080):

    slave_client = socket.socket()

    slave_client.connect(('127.0.0.1', port))

```

```

print("Starting to receive time from server\n")
send_time_thread = threading.Thread(
    target = startSendingTime,
    args = (slave_client, ))
send_time_thread.start()

print("Starting to recieving " + "synchronized time from server\n")
receive_time_thread = threading.Thread(
    target = startReceivingTime,
    args = (slave_client, ))
receive_time_thread.start()

if __name__ == '__main__':

    initiateSlaveClient(port = 8080)

```

SERVER:

```

tfcsed@tfcsed:~/Desktop/IIITU16105/EXP2.2$ python server.py
Socket at master node created successfully

Clock server started...

Starting to make connections...

Starting synchronization parallelly...

New synchronization cycle started.
Number of clients to be synchronized: 0
No client data. Synchronization not applicable.

New synchronization cycle started.
Number of clients to be synchronized: 0
No client data. Synchronization not applicable.

New synchronization cycle started.

```

CLIENT:

```

tfcsed@tfcsed:~/Desktop/IIITU16105/EXP2.2$ python client.py
Starting to receive time from server

Starting to recieving synchronized time from server

Recent time sent successfully
Synchronized time at the client is: 2019-10-18 23:00:49.547118
Recent time sent successfully
Synchronized time at the client is: 2019-10-18 23:00:54.540077
Recent time sent successfully
Synchronized time at the client is: 2019-10-18 23:00:59.549053
Recent time sent successfully
Synchronized time at the client is: 2019-10-18 23:01:04.553827
Recent time sent successfully
^[[20-Synchronized time at the client is: 2019-10-18 23:01:09.561371

```

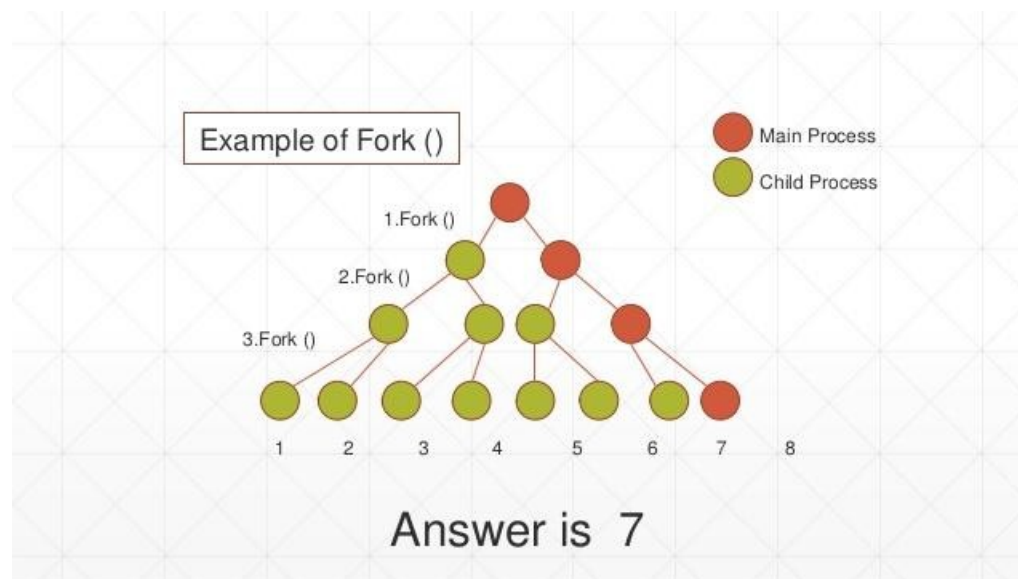
PRACTICAL - 4

Ques. Implement unix system calls(Fork, exit, opendir, readdir).

Objective: Fork system call is used for creating a new process, which is called *child process*, which runs concurrently with the process that makes the fork() call (parent process). After a new child process is created, both processes will execute the next instruction following the fork() system call. A child process uses the same pc(program counter), same CPU registers, same open files which use in the parent process.

The exec() family of functions **replaces** the current process image with a new process image. It loads the program into the current process space and runs it from the entry point.

Diagram:



FORK():

```
#include<bits/stdc++.h>
```

```
#include<unistd.h>
```

```
using namespace std;
```

```
int main()
```

```
{
```

```
cout<<"OS Fork1\n";
```

```
fork();
```

```

fork();

cout<<"OS Fork2"<<endl;

return 0;

}

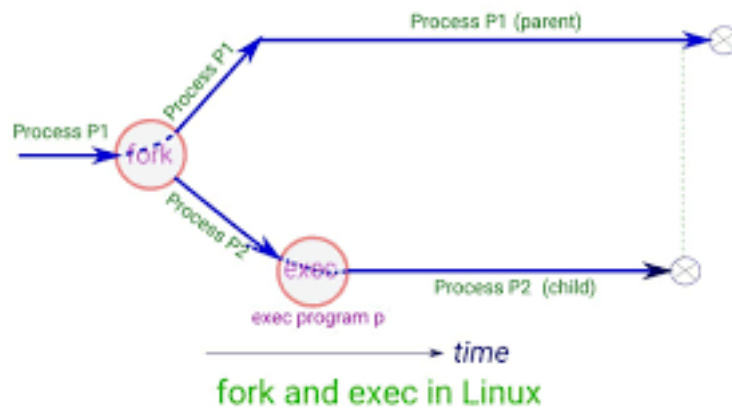
```

```

guest-1e7x0h@kartik-X556UF: ~ 80x24
~$ g++ fork.cpp
~$ ./a.out
OS Fork1
OS Fork2
OS Fork2
OS Fork2
OS Fork2
~$

```

Diagram:



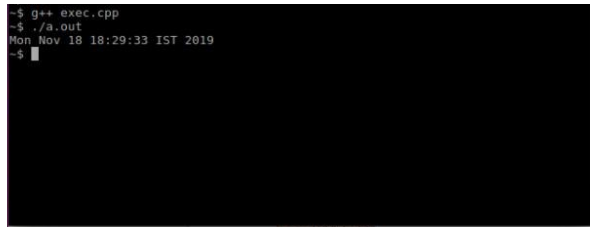
EXEC()

```

//exec program
#include<bits/stdc++.h>
#include<unistd.h>
using namespace std;
int main()
{int x = 5;

```

```
execl("/bin/date", "date", (char*)0);  
cout<<"x : "<<x<<endl;  
return 0;  
}
```



```
~$ g++ exec.cpp  
~$ ./a.out  
Mon Nov 18 18:29:33 IST 2019  
~$
```

REaddir() & OPENDIR()

```
#include <sys/types.h>  
#include <dirent.h>  
#include <stdio.h>  
#include <libgen.h>  
#include <stdlib.h>  
#include <string.h>  
  
int main(){  
    DIR *dir;  
    struct dirent *dp;  
    if((dir = opendir(".")) == NULL){  
        printf ("Cannot open .");  
        exit(1);  
    }  
    while ((dp = readdir(dir)) != NULL) {  
        printf("%s \n",dp->d_name);  
    }  
}
```



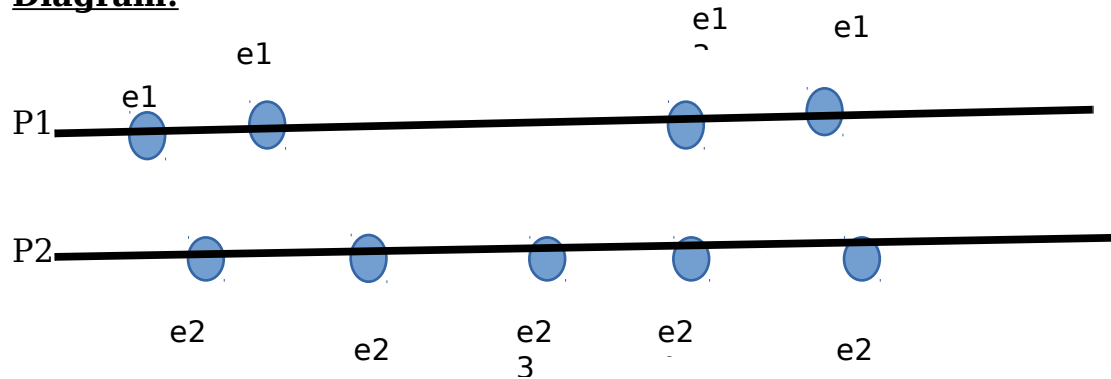
```
~$ g++ rddir.cpp
~$ ./a.out
.
..
a.out
rddir.cpp
exec.cpp
fork.cpp
.pki
.bash_history
.bashrc
.mozilla
.gnome2_private
.gnome2
.gconf
.ltcauthority
.local
.gnupg
Videos
Pictures
Music
Documents
Public
Templates
```

PRACTICAL - 5

Ques. Implement Lamport's Logical Clock .

Objective: Lamport Logical Clock to find the ordering of events in a Distributed System. In Lamport Algorithm each event has its own timestamp which depends on the occurring of events in the order the message has been sent by which event of a process to whichever event.

Diagram:



```
#include<stdio.h>
int max1(int a, int b)    //to find the maximum timestamp between two events
{
    if (a>b)
        return a;
    else
        return b;
}

int main()
{
    int i,j,k,p1[20],p2[20],e1,e2,dep[20][20];
    printf("enter the events : ");
    scanf("%d %d",&e1,&e2);
    for(i=0;i<e1;i++)
        p1[i]=i+1;
    for(i=0;i<e2;i++)
        p2[i]=i+1;
    printf("enter the dependency matrix:\n");
    printf("\t enter 1 if e1->e2 \n\t enter -1, if e2->e1 \n\t else enter 0 \n\n");
    for(i=0;i<e2;i++)
        printf("\te2%d",i+1);
    for(i=0;i<e1;i++)
    {
        printf("\n e1%d \t",i+1);
        for(j=0;j<e2;j++)
            scanf("%d",&dep[i][j]);
    }

    for(i=0;i<e1;i++)
```

```

{
    for(j=0;j<e2;j++)
    {
        if(dep[i][j]==1)    //change the timestamp if dependency
exist
        {
            p2[j]=max1(p2[j],p1[i]+1);
            for(k=j;k<e2;k++)
                p2[k+1]=p2[k]+1;
        }
        if(dep[i][j]==-1)    //change the timestamp if dependency
exist
        {
            p1[i]=max1(p1[i],p2[j]+1);
            for(k=i;k<e1;k++)
                p2[k+1]=p1[k]+1;
        }
    }
}
printf("P1 : ");    //to print the outcome of Lamport Logical Clock
for(i=0;i<e1;i++)
{
    printf("%d",p1[i]);
}
printf("\n P2 : ");
for(j=0;j<e2;j++)
    printf("%d",p2[j]);

return 0 ;
}

```

OUTPUT:

```

tfcsed@tfcsed:~$ ./a.out
enter the events : 3 3
enter the dependency matrix:
    enter 1 if e1->e2
    enter -1, if e2->e1
    else enter 0

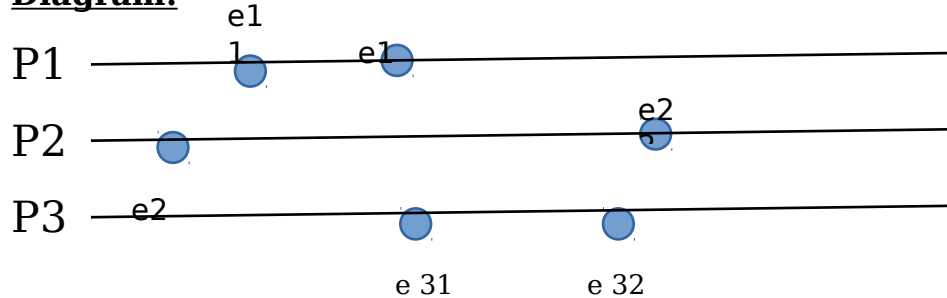
    e21    e22    e23
e11    0      0      0
e12    1      1      0
e13    0      1      1
P1 : 123
P2 : 345tfcsed@tfcsed:~$

```

PRACTICAL - 6

Ques. Implement Vector Logical Clock .

Diagram:



```
def vector_compare(vector1,vector2):
    vector = [max(value) for value in zip(vector1,vector2)]
    return vector

P = {1:{}, 2:{}, 3:{}}
inc = 0
e1 = int(input("Enter the no. of events in Process 1:"))
e1 = [i for i in range(1, e1 + 1)]
P[1] = {key: [inc + key, 0, 0] for key in e1}
e2 = int(input("Enter the no. of events in Process 2:"))
e2 = [i for i in range(1, e2 + 1)]
P[2] = {key: [0, inc + key, 0] for key in e2}
e3 = int(input("Enter the no. of events in Process 3:"))
e3 = [i for i in range(1, e3 + 1)]
P[3] = {key: [0, 0, inc + key] for key in e3}
comm = int(input("Enter the no of communication lines:"))
while inc < comm:
    sent = int(input("Enter the sending process number:"))
    recv = int(input("Enter the receiving process number:"))
    sent_event_no = int(input("Enter the sending event number:"))
    recv_event_no = int(input("Enter the receiving event number:"))
    if sent <= 3 and recv <= 3:
        print ("P{} --> P{}".format(sent,recv))
        new_vector = vector_compare(P[sent][sent_event_no],P[recv]
[recv_event_no])
        P[recv][recv_event_no] = new_vector
        print ("New vector value for P{} for event{} is:
{}".format(recv,recv_event_no,P[recv][recv_event_no]))
        if (recv_event_no+1) in P[recv]:
            P[recv][recv_event_no+1] = vector_compare(P[recv]
[recv_event_no],P[recv][recv_event_no+1])
        else:
            print ("Enter the sent/recv within existing process")
    inc += 1
```

OUTPUT:

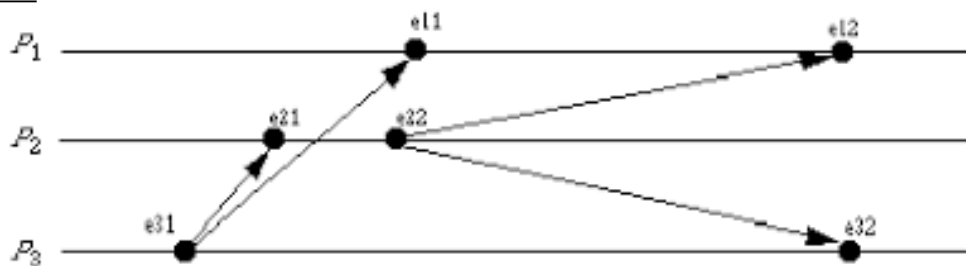
```
tfcsed@tfcsed:~/Desktop/IIITU16105/EXP4$ python vector.py
Enter the no. of events in Process 1:5
Enter the no. of events in Process 2:4
Enter the no. of events in Process 3:4
Enter the no of communication lines:4
Enter the sending process number:3
Enter the receiving process number:2
Enter the sending event number:1
Enter the receiving event number:1
P3 --> P2
New vector value for P2 for event1 is: [0, 1, 1]
Enter the sending process number:1
Enter the receiving process number:2
Enter the sending event number:2
Enter the receiving event number:2
P1 --> P2
New vector value for P2 for event2 is: [2, 2, 1]
Enter the sending process number:2
Enter the receiving process number:1
Enter the sending event number:3
Enter the receiving event number:4
P2 --> P1
New vector value for P1 for event4 is: [4, 3, 1]
Enter the sending process number:1
Enter the receiving process number:3
Enter the sending event number:5
Enter the receiving event number:3
P1 --> P3
New vector value for P3 for event3 is: [5, 3, 3]
tfcsed@tfcsed:~/Desktop/IIITU16105/EXP4$
```

PRACTICAL - 7

Ques. Implement BSS for causal ordering of messages.

OBJECTIVE: The goal of this protocol is to preserve ordering in the sending of messages.

Diagram:



```
#include <stdio.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <pthread.h>

#define MAXPENDING 100

int totalProcesses;

int processId;

unsigned int ports[100];

int proccessId;

int pclock[100];

int channelDelay[100];

int delivBuf[100][100];

int delivBufIndex=0;
```

```

struct node {
    int rclock[100];
    int isDelivered;
    struct node *next;
};

struct node *head;

void DieWithError(char *errorMessage){
    perror (errorMessage) ;
    exit(1);
}

void addToBuffer(int *rclock){
    struct node *temp = (struct node*) malloc(sizeof(struct node));
    for(int p=0;p<=totalProcesses;p++){
        temp->rclock[p] = *(rclock+p);
    }
    temp->isDelivered = 0;
    temp->next = head;
    head = temp;
}

void displayLocalClock(){
    printf("Local Clock \n");
    for(int p =0;p<totalProcesses;p++){
        printf("%d,",pclock[p]);
    }
    printf("\n\n");
}

```

```

}

void displayBuffer(){
    struct node* temp =head;
    printf("Buffered Messages Clock \n");
    while(temp !=NULL){
        for(int p =-1;p<totalProcesses;p++){
            printf("-");
        }
        printf("\n");
        for(int p =0;p<totalProcesses;p++){
            printf("%d|",temp->rclock[p]);
        }
        printf("\n");
        for(int p =-1;p<totalProcesses;p++){
            printf("-");
        }
        temp = temp->next;
    }
    printf("\n\n");
}

void checkBufferAndDeliver(struct sockaddr_in peerAddr){
    struct node* temp =head;
    while(temp !=NULL && !(temp->isDelivered)){
        if(pclock[temp->rclock[totalProcesses]] == temp->rclock[temp-
>rclock[totalProcesses]] -1){

```



```

int deliver =1;
for(int c=0;c<totalProcesses;c++){
    if(c!= temp->rclock[totalProcesses]){
        if(pclock[c] < temp->rclock[c]){
            deliver=0;
            break;
        }
    }
}
if(deliver){
    printf("\nDelivered BUffered Message Which Was Received From:-
%s(%d), Msg :- %d \n", inet_ntoa(peerAddr.sin_addr), ports[temp-
>rclock[totalProcesses]], temp->rclock[temp->rclock[totalProcesses]]);

    temp->isDelivered =1;
    for(int c=0;c<totalProcesses;c++){
        pclock[c] = pclock[c] > temp->rclock[c] ? pclock[c] : temp-
>rclock[c];
    }
}
temp = temp->next;
}
}

void handleDelivery(int *rclock,struct sockaddr_in peerAddr){
    // Check all messages are recived which sent from sender before this
event

```

```

    if(pclock[* (rclock +totalProcesses)] == *(rclock +(* (rclock
+totalProcesses))) -1){

        int deliver =1;

        for(int c=0;c<totalProcesses;c++){

            if(c!= (* (rclock+totalProcesses))) {

                if(pclock[c] <(* (rclock+c))) {

                    deliver=0;

                    break;

                }

            }

        }

        if(deliver){

            printf("\nDelivered Message From:- %s(%d), Msg :- %d \n",
inet_ntoa(peerAddr.sin_addr), ports[* (rclock+totalProcesses)], *(rclock+
(* (rclock+totalProcesses))));

            for(int c=0;c<totalProcesses;c++){

                pclock[c] = pclock[c] > (* (rclock+c)) ? pclock[c] : (* (rclock+c));

            }

            displayLocalClock();

            checkBufferAndDeliver(peerAddr);

        }else{

            addToBuffer(rclock);

            printf("*** All messages sent from other proceeses before this event is
not yet received. Putting into buffer *****\n");

            displayLocalClock();

            displayBuffer();

```

```

    }

    }else{

        addToBuffer(rclock);

        printf("*** All message sent from %s(%d) before this event is not yet
received. Putting into buffer *****\n",inet_ntoa(peerAddr.sin_addr),
ports[(*(rclock+totalProcesses))]);

        displayLocalClock();

        displayBuffer();

    }

}

void handleTCPClient(int clntSocket,struct sockaddr_in peerAddr){

    int rclock[100];

    int recvMsgSize;

    if ((recvMsgSize = recv(clntSocket, &rclock, sizeof(rclock), 0)) < 0)

        DieWithError("recv() failed");

    printf("\nReceived Message From:- %s(%d), Msg :- %d \n",
inet_ntoa(peerAddr.sin_addr), ports[rclock[totalProcesses]],
rclock[rclock[totalProcesses]]);

    handleDelivery(rclock,peerAddr);

    close(clntSocket);

}

void *recvBCMMsg(void *attribute){

    int recvSock;

    int peerSock;

    struct sockaddr_in recvAddr;

    struct sockaddr_in peerAddr;

```

```

unsigned int recvPort = ports[processId];
unsigned int peerLen;
if ((recvSock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0)
    DieWithError( "socket () failed\n" ) ;
memset(&recvAddr, 0, sizeof(recvAddr));
recvAddr.sin_family = AF_INET;
recvAddr.sin_addr.s_addr = htonl(INADDR_ANY);
recvAddr.sin_port = htons(recvPort);
if (bind(recvSock, (struct sockaddr *)&recvAddr, sizeof(recvAddr)) < 0)
    DieWithError ("bind() failed\n");
if (listen(recvSock, MAXPENDING) < 0)
    DieWithError("listen() failed\n");
printf("\nSocket Binded To Recv Messages\n");
for (;;) {
    peerLen = sizeof(peerAddr);
    if ((peerSock = accept(recvSock, (struct sockaddr *)
&peerAddr,&peerLen)) < 0)
        DieWithError("accept() failed\n");
    handleTCPClient(peerSock,peerAddr);
}
}

void sendBCMMsg(char * msgsend){
    int sendSock;
    struct sockaddr_in peerAddr;
    char *peerIP = "127.0.0.1";

```

```

pclock[processId] = pclock[processId] + 1;
int sent=0;
for(int p=processId+1;sent<totalProcesses-1;p++){
    if ((sendSock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0)
        DieWithError("socket () failed\n") ;
    memset(&peerAddr, 0, sizeof(peerAddr));
    peerAddr.sin_family = AF_INET;
    peerAddr.sin_addr.s_addr = inet_addr(peerIP);
    peerAddr.sin_port = htons(ports[ p % totalProcesses]);
    sleep(channelDelay[p % totalProcesses]);
    if (connect(sendSock, (struct sockaddr *) &peerAddr, sizeof(peerAddr))
    < 0)
        DieWithError("connect () failed\n");
    if (send(sendSock, &pclock, sizeof(pclock), 0) != sizeof(pclock))
        DieWithError("send() sent a different number of bytes than expected\n");
    sent++;
}
}

void initChannelDelay(){
    int delay=2;
    channelDelay[processId] = 0;
    int chupdated=1;
    for(int ch=processId+1;chupdated<totalProcesses;ch++){
        channelDelay[ ch % totalProcesses ] = delay;
        delay+=2;
    }
}

```

```

        chupdated++;
    }
    printf("Channel Delays to Process %d to %d ----> ",0,totalProcesses-1);
    for(int ch =0 ;ch<totalProcesses;ch++){
        printf("%d,",channelDelay[ch]);
    }
    printf("\n");
}

int main(int argc, const char * argv[])
{
    printf("Enter Total Number Of Processes (< 100):\n");
    scanf("%d",&totalProcesses);

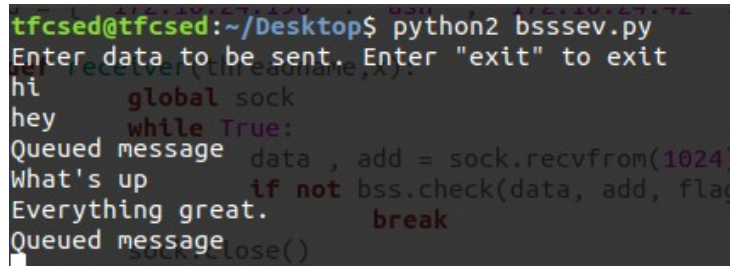
    printf("Enter Process Id of Current Process (should be 0 to %d)\n",totalProcesses-1);
    scanf("%d",&processId);

    for(int p=0;p<totalProcesses;p++){
        printf("Enter (Procees-%d) Port To Recevice Brodacast Message\n",p);
        scanf("%d",&ports[p]);    }
    pclock[totalProcesses] = processId;
    printf("This Procees Port No Is : %d\n",ports[processId]);
    initChannelDelay();
    pthread_t t;
    if(pthread_create(&t, NULL, recvBCMsg, NULL)){
        DieWithError("Failed To Create Thread To Receive Broadcast Messages");
    }

    char str[100];

```

```
for(;;){  
    printf("Enter something to send broadcast message\n");  
    scanf("%s" ,str);  
    sendBCMMsg(str);  
}  
}
```



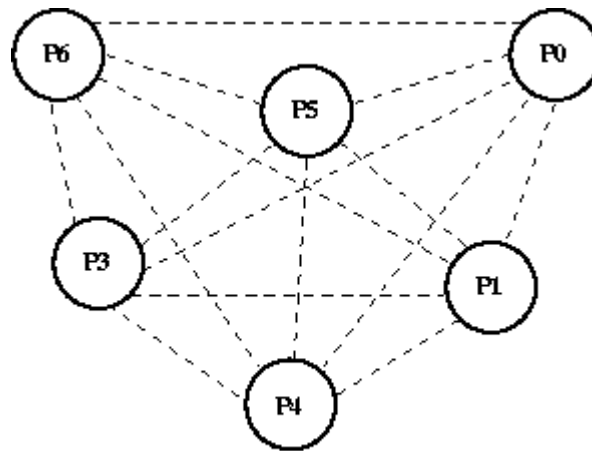
```
tfcsed@tfcsed:~/Desktop$ python2 bsssev.py  
Enter data to be sent. Enter "exit" to exit  
hi  
hey  
Queued message  
What's up  
Everything great.  
Queued message
```

PRACTICAL - 8

Ques. Implement Bully's Election algorithm.

Objective: This algorithm applies to system where every process can send a message to every other process in the system.

Diagram:



Bully Algorithm: Step 0

```
#include<stdio.h>
#include<string.h>
#include<iostream>
#include<stdlib.h>
using namespace std;
struct rr
{
char name[10];
int prior;
char state[10];
}proc[10];
```



```

int i,j,k,l,m,n;

int main()
{
    cout<<"\n enter the number of proceess \t";
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        cout<<"\nenter the name of process\t";
        cin>>proc[i].name;
        cout<<"\nenter the priority of process\t";
        cin>>proc[i].prior;
        strcpy(proc[i].state,"active");
    }
    for(i=0;i<n-1;i++)
    {
        for(j=0;j<n-1;j++)
        {
            if(proc[j].prior<proc[j+1].prior)
            {
                char ch[5];
                int t=proc[j].prior;
                proc[j].prior= proc[j+1].prior;
                proc[j+1].prior=t;
                strcpy(ch,proc[j].name);
                strcpy(proc[j].name,proc[j+1].name);
            }
        }
    }
}

```

```

strcpy(proc[j+1].name,ch);
}}}
int min=0;
for(i=0;i<n;i++)
cout<<"\n"<<proc[i].name<<"\t"<<proc[i].prior;
for(i=0;i<n;i++)
{
for(i=0;i<n;i++)
{
if(min<proc[i].prior)
min=proc[i].prior;
}}
for(i=0;i<n;i++)
{
if(proc[i].prior==min)
{
cout<<"\nprocess "<<proc[i].name<<" select as coordinator";
strcpy(proc[i].state,"iactive");
break;
}}
int pr;
while(1)
{
int ch;
cout<<"\n1)election\t";

```

```

cout<<"\n 2) exit  \t";
cin>>ch;
int max=0;
int ar[20];
k=0;
int fl=0;
switch(ch)
{
case 1: char str[5];
    cout<<"\n 1)intialise election\t";
    cin>>str;
    fl=0;
l1: for(i=0;i<n;i++)
    {
        if(strcmp(str,proc[i].name)==0)
        {
            pr=proc[i].prior;
        } }
    //cout<<"\n"<<pr;
    for(i=0;i<n;i++)
    {
        if(pr<proc[i].prior)
        {
            cout<<"\nprocess "<<str<<" send message to "<<proc[i].name;
        } }

```

```

for(i=0;i<n;i++)
{
    if(pr<proc[i].prior && strcmp(proc[i].state,"active")==0 )
    {
        if(fl==0)
        {
            ar[k]= proc[i].prior;
            k++;
        }
        cout<<"\nprocess "<<proc[i].name<<" send OK message to "<<str;
        if(proc[i].prior>max)
            max=proc[i].prior;
    } }
    fl=1;
    if(k!=0)
    {
        k=k-1;
        for(i=0;i<n;i++)
        {
            if(ar[k]==proc[i].prior)
                strcpy(str,proc[i].name);
        }
        goto l1;
    }
    m=0;

```

```

    for(j=0;j<n;j++)
    {
    if(proc[j].prior>m && strcmp(proc[j].state,"active")==0 )
    {
    cout<<"\nprocess "<<proc[j].name <<" is select as new coordinator";
    strcpy(proc[j].state,"inactive");

    break;
    } }

    for(i=0;i<n;i++)
    {
    if(strcmp(proc[i].state,"active")==0 && proc[j].prior>proc[i].prior)
    {

    cout<<"\nprocess "<<proc[j].name<<" send alert message to
"<<proc[i].name;

    } }

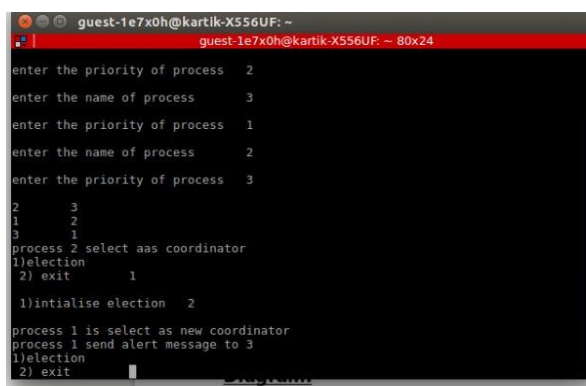
    break;

case 2:exit(1);

}}

return 0;}

```



```

guest-1e7x0h@kartik-X556UF: ~
guest-1e7x0h@kartik-X556UF: ~ 80x24
enter the priority of process  2
enter the name of process     3
enter the priority of process  1
enter the name of process     2
enter the priority of process  3
2      3
1      2
3      1
process 2 select aas coordinator
1)election
2) exit      1

1)initialise election  2

process 1 is select as new coordinator
process 1 send alert message to 3
1)election
2) exit

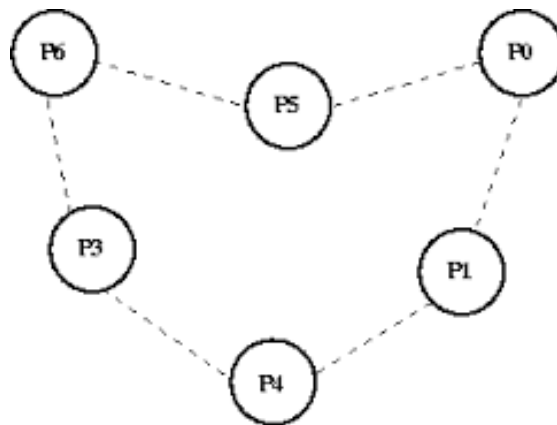
```

PRACTICAL - 9

Ques. Implementation of Ring based Election algorithm.

Objective: This algorithm applies to systems organized as a ring(logically or physically). In this algorithm we assume that the link between the process are unidirectional and every process can message to the process on its right only

Diagram:



Token Ring Election Algorithm: Step 0

```
#include<string.h>
#include<iostream>
#include<stdio.h>
#include<stdlib.h>
using namespace std;
struct rr
{
int index;
int id;
int f;
```

```

char state[10];
}proc[10];
int i,j,k,m,n;
int main()
{
int temp;
char str[10];
cout<<"\n enter the number of process\t";
cin>>n;
for(i=0;i<n;i++)
{
proc[i].index;
cout<<"\n enter id of process\t";
cin>>proc[i].id;
strcpy(proc[i].state,"active");
proc[i].f=0;
}
// sorting
for(i=0;i<n-1;i++)
{
for(j=0;j<n-1;j++)
{
if(proc[j].id>proc[j+1].id)
{
temp=proc[j].id;

```

```

proc[j].id=proc[j+1].id;
proc[j+1].id=temp;
}}}
for(i=0;i<n;i++)
printf("[%d] %d\t",i,proc[i].id);

int init;

int ch;

int temp1;

int temp2;

int arr[10];

strcpy(proc[n-1].state,"inactive");

cout<<"\nprocess "<<proc[n-1].id<<" select as coordinator";

while(1)
{
cout<<"\n1)election 2)quit\n";

scanf("%d",&ch);

for(i=0;i<n;i++)
{
proc[i].f=0;
}

switch(ch)
{

case 1:

cout<<"\nenter the process Number who intialised election";

scanf("%d",&init);

```



```

temp2=init;
temp1=init+1;
i=0;
while(temp2!=temp1)
{
if(strcmp(proc[temp1].state,"active")==0 && proc[temp1].f==0 )
{
cout<<"process "<<proc[init].id<<"send message to
"<<proc[temp1].id<<"\n";
proc[temp1].f=1;
init=temp1;
arr[i]=proc[temp1].id;
i++;
}
if(temp1==n)
temp1=0;
else
temp1++;
}

cout<<"process "<<proc[init].id<<"send message to
"<<proc[temp1].id<<"\n";
arr[i]=proc[temp1].id;
i++;
int max=-1;
for(j=0;j<i;j++)
{

```

```

if(max<arr[j])
max=arr[j];
}

cout<<"\nprocess "<<max<<" select as coordinator";
for(i=0;i<n;i++)
{
if(proc[i].id==max)
{
strcpy(proc[i].state,"inactive");
// cout<<"\n"<<i<<" "<<proc[i].id<<"deactivate\n";
} }

break;

break;

}}

return 0;

}

```

```

~$ g++ prac9.cpp
~$ ./a.out

enter the number of process    5

enter id of process    1
enter id of process    2
enter id of process    3
enter id of process    4
enter id of process    5
[0] 1  [1] 2  [2] 3  [3] 4  [4] 5
process 5 select as coordinator
1)election 2)quit
3
1)election 2)quit
1
enter the process Number who intialised election 2
process 3send message to 4
process 4send message to 1
process 1send message to 2
process 2send message to 3
process 4 select as coordinator
1)election 2)quit

```

PRACTICAL - 10

Ques. Implementation of Huang's Algorithm

Objective: Huang's algorithm is an algorithm for detecting termination in a distributed system. The algorithm was proposed by **Shing-Tsaan Huang** in 1989 in the Journal of Computers. In a distributed system, a process is either in an active state or in an idle state at any given point of time. Termination occurs when all of the processes becomes idle and there are no any in transit (on its way to be delivered) computational message.

```
#include<iostream>

using namespace std;

int main()
{
    int n,CA,p,p1,p2,i;
    float wt1;

    cout<<"Enter the number of processes\n";
    cin>>n;

    float wt[n]={0};

    cout<<"Enter the controlling agent\n";
    cin>>CA;
    wt[CA-1]= 1;

    while(1)
    {
        cout<<"Enter 1 to send, 2 to reply and 3 to exit\n";
```

```

        cin>>i;
switch(i)
{
    case 1:
        {
            cout<<"Enter the sending process\n";
            cin>>p2;
            cout<<"Enter the receiving process\n";
            cin>>p;
            cout<<"Enter the weight to be send along with the
message";

            cin>>wt1;
            wt[p-1]+=wt1;
            cout<<"Process "<<p<<" becomes active\n";
            wt[p2-1]-=wt1;
            break;
        }
    case 2:
        {
            cout<<"Enter the Replying process\n";
            cin>>p1;
            cout<<"Enter the process to whom you are
replying\n";

            cin>>p2;
            wt[p2-1]+=wt[p1-1];
            wt[p1-1]=0;

```

```

        cout<<"Process "<<p1<<" becomes inactive\n";

        break;

    }

    case 3:

        {    if(wt[CA-1]==1)

                cout<<"Termination Successful\n";

            else

                cout<<"Error in termination\n";

            exit(1);

            break;

        }

    }

return 0;

}

```

```

1
~$ g++ prac10.cpp
prac10.cpp: In function 'int main()':
prac10.cpp:55:11: error: 'exit' was not declared in this scope
    exit(1);
    ^
~$ g++ prac10.cpp
prac10.cpp: In function 'int main()':
prac10.cpp:55:11: error: 'exit' was not declared in this scope
    exit(0);
    ^
~$ g++ prac10.cpp
~$ ./a.out
Enter the number of processes
5
Enter the controlling agent
3
Enter 1 to send, 2 to reply and 3 to exit
1
Enter the sending process
2
Enter the receiving process
3
Enter the weight to be send along with the message12
2
Enter 1 to send, 2 to reply and 3 to exit
2
Enter the Replying process
3
Enter the process to whom you are replying
2
Process 3 becomes inactive
Enter 1 to send, 2 to reply and 3 to exit
1
Enter the sending process
3
Enter the receiving process
2
Enter the weight to be send along with the message23
2
Process 2 becomes active
Enter 1 to send, 2 to reply and 3 to exit

```