

**Name:** .....

**Department. :** *Computer Engineering*

**Class & Semester:** *BE (Fourth Year), SEM VIII*

**Subject:** *Distributed Computing Lab (DCL)*

---

***Expt. No.05***

**Title:** *Write A Program to Demonstrate Berkeley  
Clock Synchronization Algorithm.*

**Date:**

**Subject In-charge Sign:**

.....

## Experiment No. 05

**Aim:** Write A Program to Demonstrate Berkeley Clock Synchronization Algorithm.

### Theory:

The Berkeley algorithm is a method of clock synchronization in distributed computing which assumes no machine has an accurate time source. It was developed by Gusella and Zatti at the University of California, Berkeley in 1989 and like Cristian's algorithm is intended for use within intranets.

### Algorithm:

Unlike Cristian's algorithm, the server process in the Berkeley algorithm, called the master, periodically polls other slave processes. Generally speaking, the algorithm is:

1. A master is chosen via an election process such as Chang and Roberts's algorithm.
2. The master polls the slaves who reply with their time in a similar way to Cristian's algorithm.
3. The master observes the round-trip time (RTT) of the messages and estimates the time of each slave and its own.
4. The master then averages the clock times, ignoring any values it receives far outside the values of the others.
5. Instead of sending the updated current time back to the other process, the master then sends out the amount (positive or negative) that each slave must adjust its clock. This avoids further uncertainty due to RTT at the slave processes.

With this method the average cancels out individual clock's tendencies to drift. Gusella and Zatti released results involving 15 computers whose clocks were synchronized to within about 20-25 milliseconds using their protocol. Computer systems normally avoid rewinding their clock when they receive a negative clock alteration from the master. Doing so would break the property of monotonic time, which is a fundamental assumption in certain algorithms in the system itself or in programs such as make. A simple solution to this problem is to halt the clock for the duration specified by the master, but this simplistic solution can also cause problems, although they are less severe. For minor corrections, most systems slow the clock (known as "clock slew"), applying the correction over a longer period of time.

### Program:

```
import java.io.*;
import java.util.*;

public class Berkley
{
    float diff(int h,int m, int s, int nh, int nm, int ns)
    {
        int dh = h-nh;
        int dm = m-nm;
        int ds = s-ns;
        int diff = (dh*60*60)+(dm*60)+ds;
        return diff;
    }

    float average(float diff[],int n)
    {
        int sum=0;
        for(int i=0;i<n;i++)
            sum+=diff[i];
        float average = (float)sum/(n+1);
        System.out.println("The Average of all Time Differences is "+average);
        return average;
    }

    void sync(float diff[], int n, int h, int m, int s, int nh[], int nm[], int ns[], float average)
    {
        for(int i=0;i<n;i++)
        {
            diff[i]+=average;
            int dh=(int)diff[i]/(60*60);
            diff[i]%= (60*60);
            int dm=(int)diff[i]/60;
            diff[i]%=60;
            int ds = (int)diff[i];
            nh[i]+=dh;

            if(nh[i]>23)
                nh[i]%=24;
            nm[i]+=dm;
            if(nm[i]>59)
            {
                nh[i]++;
                nm[i]%=60;
            }
            ns[i]+=ds;
        }
    }
}
```

```

if(ns[i]>59)
{
nm[i]++;
ns[i]%=60;
}
if(ns[i]<0)
{
nm[i]--;
ns[i]+=60;
}
}
h+=(int)(average/(60*60));

```

```

if(h>23)
h%=24;

```

```

m+=(int)(average/(60*60*60));
if(m>59)
{
h++;
m%=60;
}

```

```

s+=(int)(average%(60*60*60));
if(s>59)
{
m++;
s%=60;
}
if(s<0)
{
m--;
s+=60;
}

```

```

System.out.println("The Synchronized Clocks are:\nTime Server ---> "+h":"+m+" :

```

```

"+s);
for(int i=0;i<n;i++)
System.out.println("Node "+(i+1)+" ---> "+nh[i]+" : "+ns[i]);
}

```

```

public static void main(String args) throws IOException
{
Berkely b = new Berkely();
Date date = new Date();

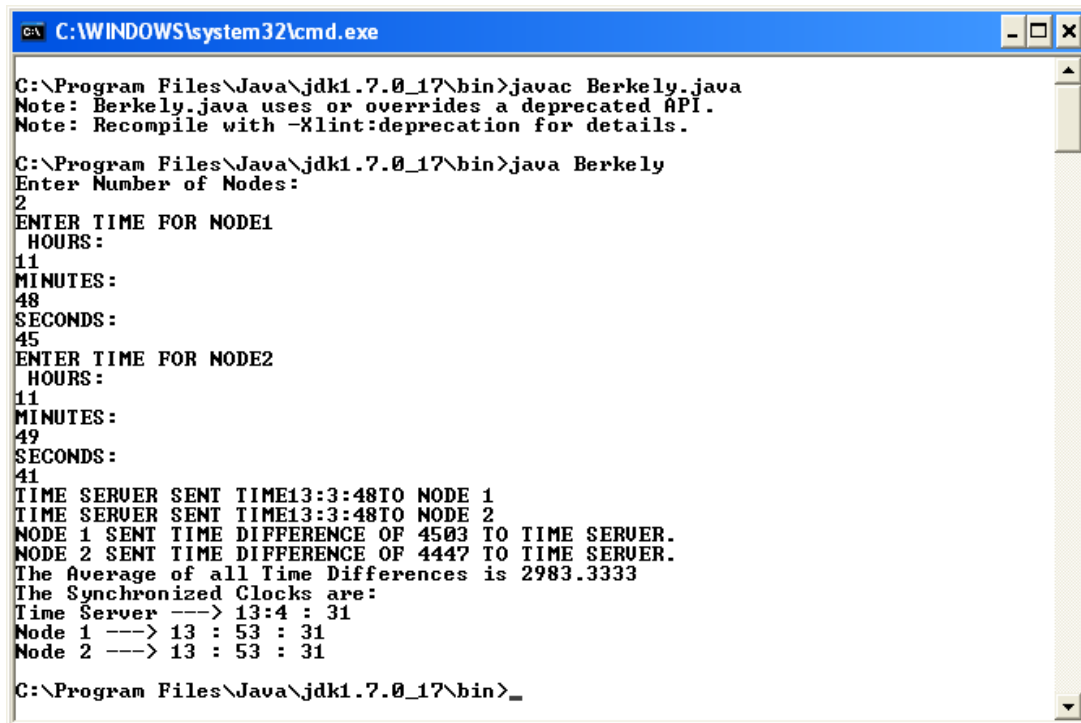
```

```

BufferedReader obj = new BufferedReader(new InputStreamReader(System.in));
System.out.println("Enter Number of Nodes:");
int n = Integer.parseInt(obj.readLine());
int h = date.getHours();
int m = date.getMinutes();
int s = date.getSeconds();
int nh[]=new int[n];
int nm[]=new int[n];
int ns[]=new int[n];
for(int i=0;i<n;i++)
{
System.out.println("ENTER TIME FOR NODE" +(i+1) + "\n HOURS:");
nh[i]=Integer.parseInt(obj.readLine());
System.out.println("MINUTES:")
nm[i]=Integer.parseInt(obj.readLine());
System.out.println("SECONDS:")
ns[i]=Integer.parseInt(obj.readLine());
}
for(int i=0;i<n;i++)
{
System.out.println("TIME SERVER SENT TIME" +h+ ":" +m+ ":" +s+ "TO NODE " +(i+1));
}
float diff[]=new float[n];
for(int i=0;i<n;i++)
{
diff[i]=b.diff(h,m,s,nh[i],ns[i]);
System.out.println("NODE " +(i+1) " SENT TIME DIFFERENCE OF " +(int) diff[i] + " TO TIME
SERVER.");
}
float average=b.average(diff,n);
b.sync(diff,n,h,m,s,nh,nm,ns,average);
}
}

```

## Output:



```
C:\WINDOWS\system32\cmd.exe

C:\Program Files\Java\jdk1.7.0_17\bin>javac Berkely.java
Note: Berkely.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

C:\Program Files\Java\jdk1.7.0_17\bin>java Berkely
Enter Number of Nodes:
2
ENTER TIME FOR NODE1
HOURS:
11
MINUTES:
48
SECONDS:
45
ENTER TIME FOR NODE2
HOURS:
11
MINUTES:
49
SECONDS:
41
TIME SERVER SENT TIME13:3:48TO NODE 1
TIME SERVER SENT TIME13:3:48TO NODE 2
NODE 1 SENT TIME DIFFERENCE OF 4503 TO TIME SERVER.
NODE 2 SENT TIME DIFFERENCE OF 4447 TO TIME SERVER.
The Average of all Time Differences is 2983.3333
The Synchronized Clocks are:
Time Server ---> 13:4 : 31
Node 1 ---> 13 : 53 : 31
Node 2 ---> 13 : 53 : 31

C:\Program Files\Java\jdk1.7.0_17\bin>_
```

## Conclusion:-

Thus we have studied a program to demonstrate Berkeley clock synchronization algorithm.