DISTRIBUTED COMPUTING

ASSIGNMENT

Abstract

RPC BASED CALCULATOR & LAMPORT CLOCK

Name : Ranjan Mishra

BITS ID: 2018HT12065

ASSIGNMENT TASK 1

- **To implement a calculator service using remote procedure call (SUN RPC package) as discussed in the live sessions, which takes 2 integer arguments as inputs and performs the following operations:**

1. **Addition of two numbers.**

2. **Multiplication of two numbers.**

3. **Subtraction of two numbers.**

4. **Division of two numbers.**

5. **Remainder of two numbers (First Number % Second Number).**

6. **Check if the first number (argument) is a prime number (For example, if the input is 23 and 10, then output should be 1 (or yes)).**

Code for the above functionalities is given below (.x IDL file, client.c, and server.c files). Your task is to add the following additional functionalities into the given program:

7. **Check if both the input arguments are even or both are odd. Display “Yes” if both are either even or both are odd, else display “No”.**

8. **Pl. ensure proper error checks like, if the input arguments are not integers then the server should respond back with an error message as “arguments are not of valid type**

Solution:

1. **Cal.x :**

```
struct input {
      int x;
      int y;
      int ans;
      int choice;
};

struct output {
      int result;
      bool isSuccess;
      int oddOrEven;
};
program calc_prg {
      version calc_version {
                  output calc(input)=1;
      }=1;
}=0x32345676;
```

2. **Cal_client.c :**

```
#include "cal.h"

void calc_prg_1(char *host) {
      CLIENT *clnt;
      output *result_1;
      input calc_1_arg;
  calc_1_arg.choice = 0;
      clnt = clnt_create (host, calc_prg, calc_version, "udp");
      if (clnt == NULL) {
                  clnt_pcreateerror(host);
                  exit (1);
      }
  do  {
    printf("Please select a choice from 1 to 7 as shown below");
      printf("\n1 Addition \n2.Multiplication \n3.Subtraction \n4.Division \n5.Remainder
\n6.Is first number prime\n7.Check Odd or Even\nEnter choice (1 to 7): ");
      if (!scanf("%d",&calc_1_arg.choice)) {
     calc_1_arg.choice = 0;
    }
  } while((calc_1_arg.choice <= 0 || calc_1_arg.choice >= 8));
      printf("Enter values: ");
      if (!scanf("%d %d",&calc_1_arg.x,&calc_1_arg.y)) {
    printf("--------------------------------------------------\n");
    printf("Error: Invalid arguments for the selected operation\n");
    printf("--------------------------------------------------\n");
    exit(1);
  }
      result_1 = calc_1(&calc_1_arg, clnt);
      if (result_1 == (output *) NULL) {
                  clnt_perror (clnt, "Call failed");
      }
      if(calc_1_arg.choice != 6 && calc_1_arg.choice != 7) {
    if (result_1->isSuccess) {
      printf("---------------\n");
      printf("Result %d \n", result_1->result);
      printf("---------------\n");
    } else {
      printf("--------------------------------------------------\n");
      printf("Error: Invalid arguments for the selected operation\n");
      printf("--------------------------------------------------\n");
    }
  } else {
    if(calc_1_arg.choice==6){
      if (result_1->result==1) {
          printf("-----------------\n");
                  printf("Prime Number: YES\n");
          printf("-----------------\n");
      } else {
        printf("----------------\n");
        printf("Prime Number: No\n");
        printf("----------------\n");
      }
    } else if (calc_1_arg.choice==7){
      if (result_1->oddOrEven == 0) {
        printf("---------------------------\n");
        printf("Both inputs are odd numbers\n");
        printf("---------------------------\n");
      } else if (result_1->oddOrEven == 1) {
        printf("----------------------------\n");
        printf("Both inputs are even numbers \n");
        printf("----------------------------\n");
      } else {
        printf("-----------------------------------\n");
        printf("Inputs has both odd and even number\n");
        printf("-----------------------------------\n");
      }
    }
  }
      clnt_destroy (clnt);
}
int main (int argc, char *argv[]) {
      char *host;
      if (argc < 2) {
```

```
                printf ("usage: %s server_host\n", argv[0]);
                exit (1);
        }
        host = argv[1];
        calc_prg_1 (host);
        exit (0);
}
```

3. Cal_clnt.c:

```
/*
 * Please do not edit this file.
 * It was generated using rpcgen.
 */

#include <memory.h> /* for memset */
#include "cal.h"

/* Default timeout can be changed using clnt_control() */
static struct timeval TIMEOUT = { 25, 0 };

output *
calc_1(input *argp, CLIENT *clnt)
{
        static output clnt_res;

        memset((char *)&clnt_res, 0, sizeof(clnt_res));
        if (clnt_call (clnt, calc,
                (xdrproc_t) xdr_input, (caddr_t) argp,
                (xdrproc_t) xdr_output, (caddr_t) &clnt_res,
                TIMEOUT) != RPC_SUCCESS) {
                return (NULL);
        }
        return (&clnt_res);
}
```

4. Cal_server.c :

```
/*
 * This is sample code generated by rpcgen.
 * These are only templates and you can use them
 * as a guideline for developing your own functions.
 */
```

```
#include "cal.h"
#include <math.h>

int isprime(int a) {
    int i;
    for(i=2;i<=a/2;i++) {
            if(a%i==0){
                    break;
            }
    }
    if(i==a/2+1) {
            printf("\nYES\n");
            return 1;
    } else {
            printf("\nNO\n");
            return 0;
    }
}

int oddOrEven(int a, int b) {
    int aOdd = isOdd(a);
    int bOdd = isOdd(b);
    if (aOdd && bOdd) {
            return 0;
    } else if (!aOdd && !bOdd) {
            return 1;
    } else {
            return -1;
    }
}

int Odd(int a) {
    return a & 1;
}

int isDigit(int a) {
    return floor(a) == ceil(a);
}

output * cal_svc(input *argp, struct svc_req *rqstp) {

    static output  result;
```

```
        result.result = NULL;
        result.oddOrEven = -1;
        if (isDigit(argp->x) && isDigit(argp->y)) {
                result.isSuccess = 1;
                switch(argp->choice) {
                        case 1:
                                result.result=(argp->x)+(argp->y);
                                break;
                        case 2:
                                result.result=(argp->x)*(argp->y);
                                break;
                        case 3:
                                result.result=(argp->x)-(argp->y);
                                break;
                        case 4:
                                if (argp->y != 0) {
                                        result.result=(argp->x)/(argp->y);
                                } else {
                                        result.isSuccess = 0;
                                }
                                break;
                        case 5:
                                result.result=(argp->x)%(argp->y);
                                break;
                        case 6:
                                result.result=isprime(argp->x);
                                break;
                        case 7:
                                result.oddOrEven = oddOrEven(argp->x, argp->y);
                                break;
                        default:
                                result.isSuccess = 0;
                                break;
                }
        } else {
                result.isSuccess = 0;
        }
        return(&result);
    }
```

5. **Cal_svc.c :**

```
/*
 * Please do not edit this file.
 * It was generated using rpcgen.
 */

#include "cal.h"
#include <stdio.h>
#include <stdlib.h>
#include <rpc/pmap_clnt.h>
#include <string.h>
#include <memory.h>
#include <sys/socket.h>
#include <netinet/in.h>

#ifndef SIG_PF
#define SIG_PF void(*)(int)
#endif

static void
calc_prg_1(struct svc_req *rqstp, register SVCXPRT *transp)
{
	union {
		input calc_1_arg;
	} argument;
	char *result;
```

```
xdrproc_t _xdr_argument, _xdr_result;

char *(*local)(char *, struct svc_req *);


switch (rqstp->rq_proc) {

case NULLPROC:

        (void) svc_sendreply (transp, (xdrproc_t) xdr_void, (char *)NULL);

        return;


case calc:

        _xdr_argument = (xdrproc_t) xdr_input;

        _xdr_result = (xdrproc_t) xdr_output;

        local = (char *(*)(char *, struct svc_req *)) calc_1_svc;

        break;


default:

        svcerr_noproc (transp);

        return;

}

memset ((char *)&argument, 0, sizeof (argument));

if (!svc_getargs (transp, (xdrproc_t) _xdr_argument, (caddr_t) &argument)) {

        svcerr_decode (transp);

        return;

}

result = (*local)((char *)&argument, rqstp);

if (result != NULL && !svc_sendreply(transp, (xdrproc_t) _xdr_result, result)) {
		svcerr_systemerr (transp);
	}
	if (!svc_freeargs (transp, (xdrproc_t) _xdr_argument, (caddr_t) &argument)) {
		fprintf (stderr, "%s", "unable to free arguments");
		exit (1);
	}
	return;
}

int
main (int argc, char **argv)
{
	register SVCXPRT *transp;

	pmap_unset (calc_prg, calc_version);

	transp = svcudp_create(RPC_ANYSOCK);
	if (transp == NULL) {
		fprintf (stderr, "%s", "cannot create udp service.");
		exit(1);
	}
	if (!svc_register(transp, calc_prg, calc_version, calc_prg_1, IPPROTO_UDP)) {
		fprintf (stderr, "%s", "unable to register (calc_prg, calc_version, udp).");
		exit(1);
	}
	transp = svctcp_create(RPC_ANYSOCK, 0, 0);

	if (transp == NULL) {

		fprintf (stderr, "%s", "cannot create tcp service.");

		exit(1);

	}

	if (!svc_register(transp, calc_prg, calc_version, calc_prg_1, IPPROTO_TCP)) {

		fprintf (stderr, "%s", "unable to register (calc_prg, calc_version, tcp).");

		exit(1);

	}


	svc_run ();

	fprintf (stderr, "%s", "svc_run returned");

	exit (1);

	/* NOTREACHED */

}
```

6. cal_xdr.c :

```
/*
 * Please do not edit this file.
 * It was generated using rpcgen.
 */

#include "cal.h"

bool_t
xdr_input (XDR *xdrs, input *objp)
{
	register int32_t *buf;


	if (xdrs->x_op == XDR_ENCODE) {
        buf = XDR_INLINE (xdrs, 4 * BYTES_PER_XDR_UNIT);
        if (buf == NULL) {
                 if (!xdr_int (xdrs, &objp->x))
                         return FALSE;
                 if (!xdr_int (xdrs, &objp->y))
                         return FALSE;
                 if (!xdr_int (xdrs, &objp->ans))
                         return FALSE;
                 if (!xdr_int (xdrs, &objp->choice))
                         return FALSE;
        } else {
                IXDR_PUT_LONG(buf, objp->x);
                IXDR_PUT_LONG(buf, objp->y);
                IXDR_PUT_LONG(buf, objp->ans);
                IXDR_PUT_LONG(buf, objp->choice);
        }
        return TRUE;
} else if (xdrs->x_op == XDR_DECODE) {
        buf = XDR_INLINE (xdrs, 4 * BYTES_PER_XDR_UNIT);
        if (buf == NULL) {
                 if (!xdr_int (xdrs, &objp->x))
                         return FALSE;
                 if (!xdr_int (xdrs, &objp->y))
                         return FALSE;
                 if (!xdr_int (xdrs, &objp->ans))
                         return FALSE;
                 if (!xdr_int (xdrs, &objp->choice))
                         return FALSE;
        } else {
                objp->x = IXDR_GET_LONG(buf);
                objp->y = IXDR_GET_LONG(buf);
                objp->ans = IXDR_GET_LONG(buf);
                objp->choice = IXDR_GET_LONG(buf);
        }
 return TRUE;
}

 if (!xdr_int (xdrs, &objp->x))
         return FALSE;
 if (!xdr_int (xdrs, &objp->y))
         return FALSE;
 if (!xdr_int (xdrs, &objp->ans))
```

```
                return FALSE;
         if (!xdr_int (xdrs, &objp->choice))
                return FALSE;
        return TRUE;
    }

    bool_t
    xdr_output (XDR *xdrs, output *objp)
    {
        register int32_t *buf;

         if (!xdr_int (xdrs, &objp->result))
                return FALSE;
         if (!xdr_bool (xdrs, &objp->isSuccess))
                return FALSE;
         if (!xdr_int (xdrs, &objp->oddOrEven))
                return FALSE;
        return TRUE;
    }
```

7. client.c :

```
#include "cal.h"



void calc_prg_1(char *host) {

        CLIENT *clnt;

        output *result_1;

        input calc_1_arg;

  calc_1_arg.choice = 0;

        clnt = clnt_create (host, calc_prg, calc_version, "udp");

        if (clnt == NULL) {

                clnt_pcreateerror(host);

                exit (1);

        }

  do  {
```

```
  printf("Please select a choice from 1 to 7 as shown below");
          printf("\n1 Addition \n2.Multiplication \n3.Subtraction \n4.Division \n5.Remainder \n6.Is
first number prime\n7.Check Odd or Even\nEnter choice (1 to 7): ");
          if (!scanf("%d",&calc_1_arg.choice)) {
   calc_1_arg.choice = 0;
  }
 } while((calc_1_arg.choice <= 0 || calc_1_arg.choice >= 8));
          printf("Enter values: ");
          if (!scanf("%d %d",&calc_1_arg.x,&calc_1_arg.y)) {
  printf("--------------------------------------------------\n");
  printf("Error: Invalid arguments for the selected operation\n");
  printf("--------------------------------------------------\n");
  exit(1);
 }
          result_1 = calc_1(&calc_1_arg, clnt);
          if (result_1 == (output *) NULL) {
                    clnt_perror (clnt, "Call failed");
          }
          if(calc_1_arg.choice != 6 && calc_1_arg.choice != 7) {
  if (result_1->isSuccess) {
   printf("---------------\n");
   printf("Result %d \n", result_1->result);
   printf("---------------\n");
  } else {
   printf("--------------------------------------------------\n");
   printf("Error: Invalid arguments for the selected operation\n");
```

```
    printf("------------------------------------------------------\n");
   }
 } else {
  if(calc_1_arg.choice==6){
    if (result_1->result==1) {
        printf("-----------------\n");
            printf("Prime Number: YES\n");
        printf("-----------------\n");
    } else {
      printf("----------------\n");
      printf("Prime Number: No\n");
      printf("----------------\n");
    }
  } else if (calc_1_arg.choice==7){
    if (result_1->oddOrEven == 0) {
      printf("---------------------------\n");
      printf("Both inputs are odd numbers\n");
      printf("---------------------------\n");
    } else if (result_1->oddOrEven == 1) {
      printf("----------------------------\n");
      printf("Both inputs are even numbers \n");
      printf("----------------------------\n");
    } else {
      printf("-----------------------------------\n");
      printf("Inputs has both odd and even number\n");
```

```
       printf("----------------------------------\n");
      }
    }
  }
            clnt_destroy (clnt);
}
int main (int argc, char *argv[]) {
            char *host;
            if (argc < 2) {
                        printf ("usage: %s server_host\n", argv[0]);
                        exit (1);
            }
            host = argv[1];
            calc_prg_1 (host);
            exit (0);
}
```

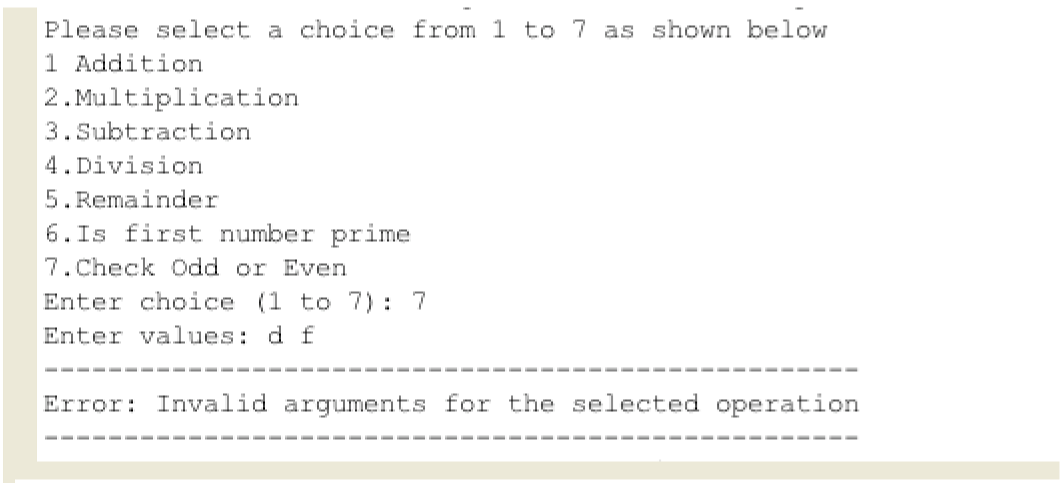
8. server.c :

```
/*
 * This is sample code generated by rpcgen.
 * These are only templates and you can use them
 * as a guideline for developing your own functions.
 */

#include "cal.h"
#include <math.h>

int isprime(int a) {
     int i;
     for(i=2;i<=a/2;i++) {
               if(a%i==0){
                break;
            }
        }
        if(i==a/2+1) {
            printf("\nYES\n");
            return 1;
        } else {
            printf("\nNO\n");
            return 0;
        }
}

int oddOrEven(int a, int b) {
        int aOdd = isOdd(a);
        int bOdd = isOdd(b);
        if (aOdd && bOdd) {
            return 0;
        } else if (!aOdd && !bOdd) {
            return 1;
        } else {
            return -1;
        }
}

int isOdd(int a) {
        return a & 1;
}

int isDigit(int a) {
        return floor(a) == ceil(a);
}

output * calc_1_svc(input *argp, struct svc_req *rqstp) {

        static output  result;
        result.result = NULL;
        result.oddOrEven = -1;
        if (isDigit(argp->x) && isDigit(argp->y)) {
            result.isSuccess = 1;
            switch(argp->choice) {
                case 1:
                    result.result=(argp->x)+(argp->y);
```

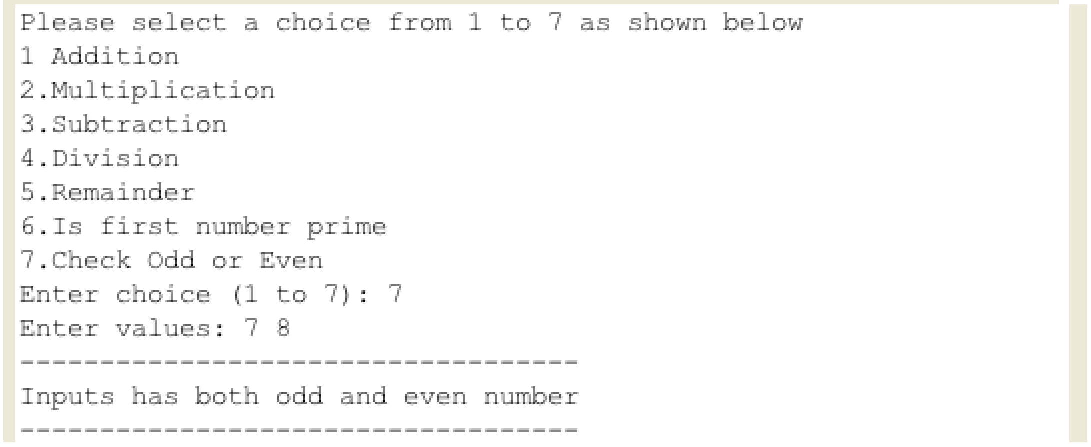
```
                                break;
                        case 2:
                                result.result=(argp->x)*(argp->y);
                                break;
                        case 3:
                                result.result=(argp->x)-(argp->y);
                                break;
                        case 4:
                                if (argp->y != 0) {
                                        result.result=(argp->x)/(argp->y);
                                } else {
                                        result.isSuccess = 0;
                                }
                                break;
                        case 5:
                                result.result=(argp->x)%(argp->y);
                                break;
                        case 6:
                                result.result=isprime(argp->x);
                                break;
                        case 7:
                                result.oddOrEven = oddOrEven(argp->x, argp->y);
                                break;
                        default:
                                result.isSuccess = 0;
                                break;
                }
        } else {
                result.isSuccess = 0;
        }
        return(&result);
}
```

Snapshots :

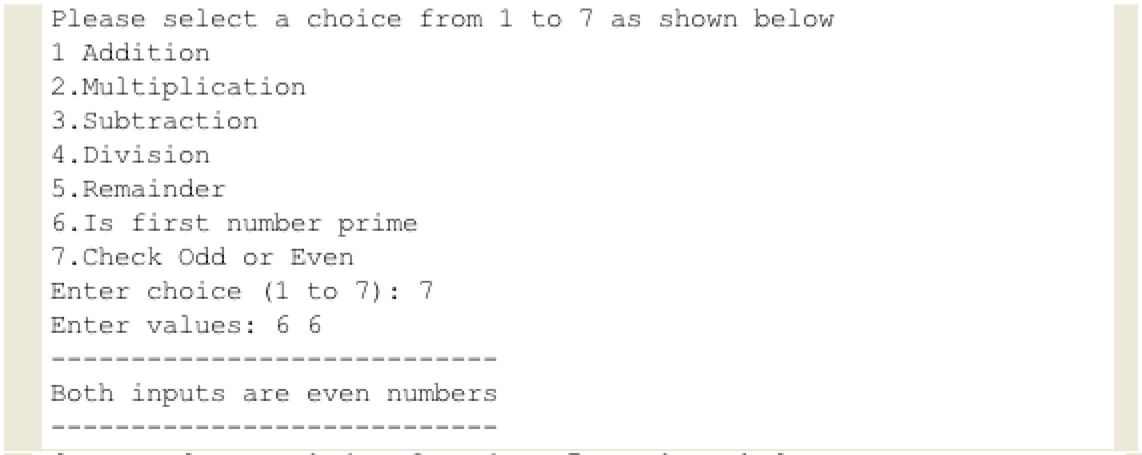
```
Please select a choice from 1 to 7 as shown below
1 Addition
2.Multiplication
3.Subtraction
4.Division
5.Remainder
6.Is first number prime
7.Check Odd or Even
Enter choice (1 to 7): 1
Enter values: 2 3
---------------
Result 5
Please select a choice from 1 to 7 as shown below
1 Addition
2.Multiplication
3.Subtraction
4.Division
5.Remainder
6.Is first number prime
7.Check Odd or Even
Enter choice (1 to 7): 4
Enter values: 4 0
---------------------------------------------------
Error: Invalid arguments for the selected operation
---------------------------------------------------
Please select a choice from 1 to 7 as shown below
1 Addition
2.Multiplication
3.Subtraction
4.Division
5.Remainder
6.Is first number prime
7.Check Odd or Even
Enter choice (1 to 7): 4
Enter values: 4 4
---------------
Result 1
---------------
```



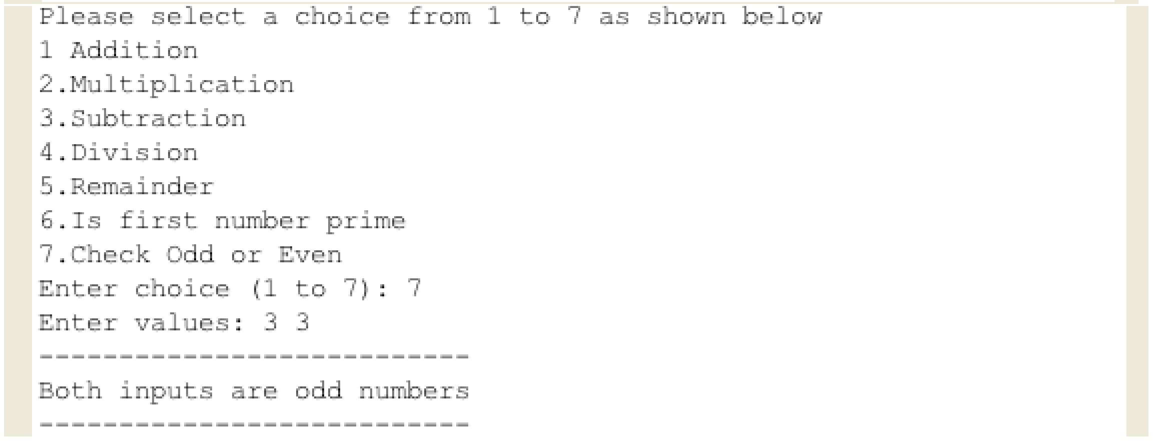
```
Please select a choice from 1 to 7 as shown below
1 Addition
2.Multiplication
3.Subtraction
4.Division
5.Remainder
6.Is first number prime
7.Check Odd or Even
Enter choice (1 to 7): 7
Enter values: d f
---------------------------------------------------
Error: Invalid arguments for the selected operation
---------------------------------------------------
```



```
Please select a choice from 1 to 7 as shown below
1 Addition
2.Multiplication
3.Subtraction
4.Division
5.Remainder
6.Is first number prime
7.Check Odd or Even
Enter choice (1 to 7): 7
Enter values: 7 8
-----------------------------------
Inputs has both odd and even number
-----------------------------------
```



```
Please select a choice from 1 to 7 as shown below
1 Addition
2.Multiplication
3.Subtraction
4.Division
5.Remainder
6.Is first number prime
7.Check Odd or Even
Enter choice (1 to 7): 7
Enter values: 6 6
----------------------------
Both inputs are even numbers
----------------------------
```



```
Please select a choice from 1 to 7 as shown below
1 Addition
2.Multiplication
3.Subtraction
4.Division
5.Remainder
6.Is first number prime
7.Check Odd or Even
Enter choice (1 to 7): 7
Enter values: 3 3
---------------------------
Both inputs are odd numbers
---------------------------
```

Procedure :

- **Using command : rpcgen -C cal.x**
  It will create automatically cal.h, cal_clnt.c, cal_svc.c, cal_xdr.c
- **gcc -lnsl -o client cal_client.c cal_clnt.c cal_xdr.c**
- **gcc -lrpcsvc -lnsl -o server cal_server.c cal_svc.c cal_xdr.c**

ASSIGNMENT TASK 2

- **Lamport proposed a scheme to order the events in a distributed system by using logical clocks. Due to the absence of synchronized clocks and hence global time in a distributed system, the order in which events occur at two different machines is impossible to be determined based on the local time at which they occurred.**
  The implementation rules are as below:
  a) Clock Ci is implemented between any two events of the same process as Ci = Ci + d (d >0)
  b) If event a is a send message by process Pi and the same is received by process Pj, then tm = Ci (a), and Cj = max (Cj, tm+d), d>0.

  Below code is an implementation of the above two rules (simulation). Output is also shown for clarity. Your task is to modify the below program such that value of 'd' becomes 3 for all the processes, and the starting value of the clock at P1, P2, and P3 is 4. Submit the code with a snapshot of the run.

Solution :

Java Program :

```
package com.dc;


import java.util.ArrayList;

import java.util.HashMap;

import java.util.Scanner;


public class LamportLogicalClock {


        private static Scanner scanner = new Scanner(System.in);
```

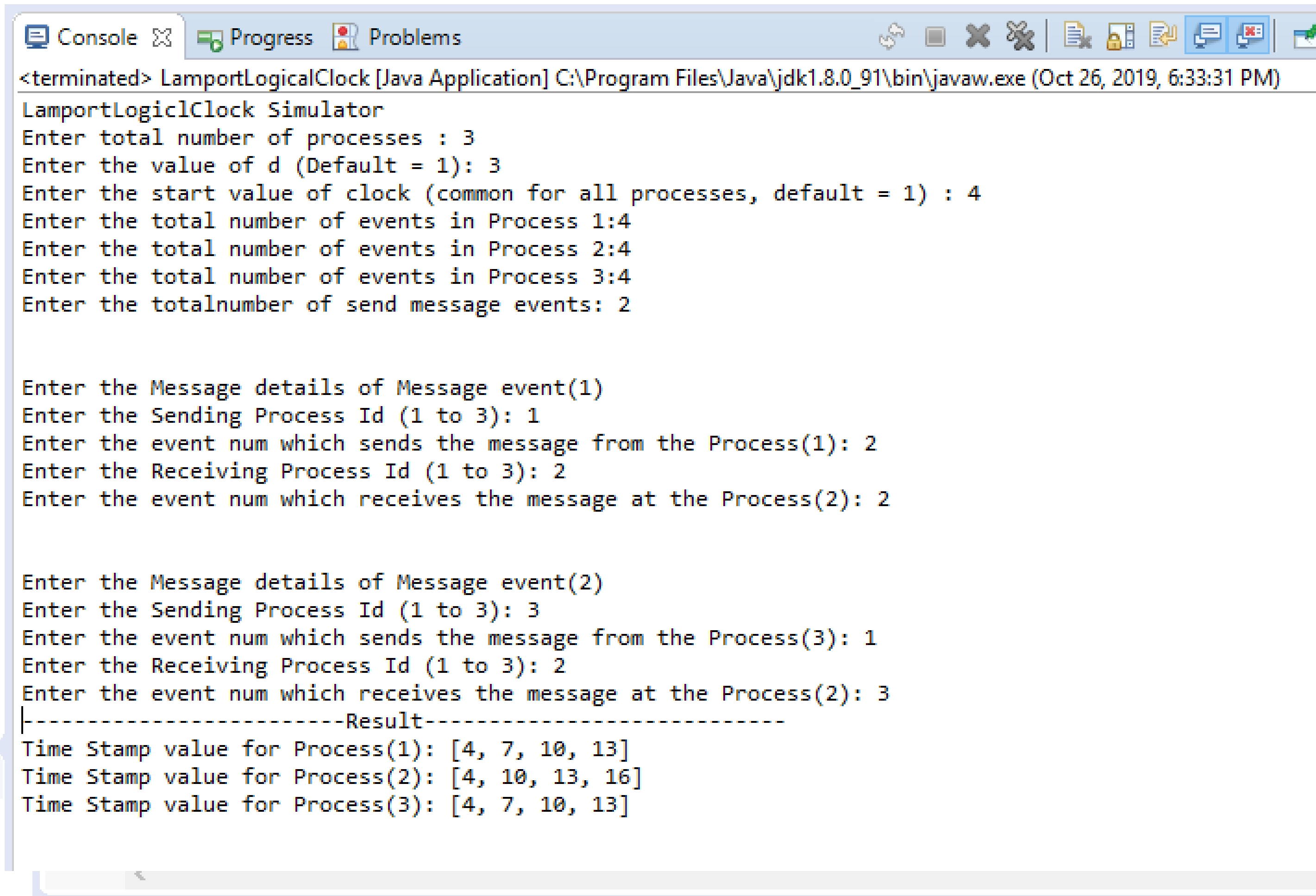
```
	private int nProcesses;

	private int d = 1;

	private int startClock = 1;

	private HashMap<Integer, ArrayList<Integer>> processes = new HashMap<Integer,
ArrayList<Integer>>();

	private ArrayList<MessageEvent> messageEvents = new ArrayList<MessageEvent>();



	public void start() {

		System.out.println("LamportLogiclClock Simulator");
		System.out.print("Enter total number of processes : ");
		nProcesses = scanner.nextInt();
		System.out.print("Enter the value of d (Default = 1): ");
		d = scanner.nextInt();
		System.out.print("Enter the start value of clock (common for all processes, default = 1)
: ");
		startClock = scanner.nextInt();
		initialize();
		readSendAndReceiveEvents();
		processLamportLogiclClock();
		printResult();
	}



	private void printResult() {
		System.out.println("-------------------------Result---------------------------");
		for (int eachPid = 1; eachPid <= nProcesses; eachPid++) {
			System.out.println("Time Stamp value for Process("+ eachPid + "): " +
processes.get(eachPid));
		}
	}

	private MessageEvent isRecievingEvent(int recvPid, int recvEventId) {
		for (int i = 0; i< messageEvents.size(); i++) {
			MessageEvent event = messageEvents.get(i);
			if (event.getRecvProcessNum() == recvPid && event.getRecvEventNum() ==
recvEventId) {
				return event;
			}
		}
		return null;
	}



	private void processLamportLogiclClock() {
		for (int eachPid = 1; eachPid <= nProcesses; eachPid++) {
```

```
                    ArrayList<Integer> processEvents = processes.get(eachPid);
                    for (int eachEvnId = 0; eachEvnId< processEvents.size(); eachEvnId++) {
                        MessageEvent messageEvent = isRecievingEvent(eachPid, eachEvnId +
1);
                        if ( messageEvent != null) {
                            int sendProcId = messageEvent.getFromProcessNum();
                            int sendEventId = messageEvent.getFromEventNum();
                            processEvents.set(eachEvnId,

    (Math.max(processes.get(sendProcId).get(sendEventId -1),
                                                    processEvents.get(eachEvnId))
+ d));


                        } else {
                            if (eachEvnId != 0) {
                                processEvents.set(eachEvnId,
                                        processEvents.get(eachEvnId - 1) + d);
                            }
                        }
                    }
                }
    }


    private void readSendAndReceiveEvents() {
        System.out.print("Enter the totalnumber of send message events: ");
        int sendEvents = scanner.nextInt();
        for (int i=1; i <= sendEvents; i++) {
```

```
                              System.out.print("\n\nEnter the Message details of Message event(" + i +
")\n");

                              System.out.print("Enter the Sending Process Id (1 to " + nProcesses + "): ");

                              int fromProcess = scanner.nextInt();

                              System.out.print("Enter the event num which sends the message from the
Process(" + fromProcess + "): ");

                              int fromEvent = scanner.nextInt();

                              System.out.print("Enter the Receiving Process Id (1 to " + nProcesses + "): ");

                              int recvProcess = scanner.nextInt();

                              System.out.print("Enter the event num which receives the message at the
Process(" + recvProcess + "): ");

                              int recvEvent = scanner.nextInt();

                              MessageEvent messageEvent = new MessageEvent(fromProcess, fromEvent,
recvProcess, recvEvent);

                              messageEvents.add(messageEvent);

                    }

          }




          private void initialize() {

                    for (int i = 1; i <= nProcesses; i++) {

                              System.out.print("Enter the total number of events in Process " + i + ":");

                              int nEvents = scanner.nextInt();

                              ArrayList<Integer> events = new ArrayList<Integer>();

                              for (int j=0; j< nEvents; j++) {

                                        if (j ==0) {

                                                  events.add(startClock);
                    } else {

                        events.add(events.get(j-1) + d);

                    }

                }

                processes.put(i, events);

            }

        }



        public static void main(String...a) {


            LamportLogicalClock lamportLogiclClock = new LamportLogicalClock();

            lamportLogiclClock.start();

        }


    }


    class MessageEvent {


        int fromProcessNum;

        int fromEventNum;

        int recvProcessNum;

        int recvEventNum;


        public MessageEvent() {
}

public MessageEvent(int frmProc, int fromEvnt, int rcvProc, int rcvEvnt) {
        this.fromProcessNum = frmProc;
        this.fromEventNum = fromEvnt;
        this.recvProcessNum = rcvProc;
        this.recvEventNum = rcvEvnt;
}

public int getFromProcessNum() {
        return fromProcessNum;
}
public void setFromProcessNum(int fromProcessNum) {
        this.fromProcessNum = fromProcessNum;
}
public int getFromEventNum() {
        return fromEventNum;
}
public void setFromEventNum(int fromEventNum) {
        this.fromEventNum = fromEventNum;
}
public int getRecvProcessNum() {
        return recvProcessNum;
}
```

```
	public void setRecvProcessNum(int recvProcessNum) {

		this.recvProcessNum = recvProcessNum;

	}

	public int getRecvEventNum() {

		return recvEventNum;

	}

	public void setRecvEventNum(int recvEventNum) {

		this.recvEventNum = recvEventNum;

	}


	public String toString() {

		return "[fromProcessNum: "+ this.fromProcessNum+", fromEventNum:"

				+ this.fromEventNum + ", recvProcessNum: "

				+ this.recvProcessNum + ", recvEventNum: "+ this.recvEventNum + "]";

	}

}
```

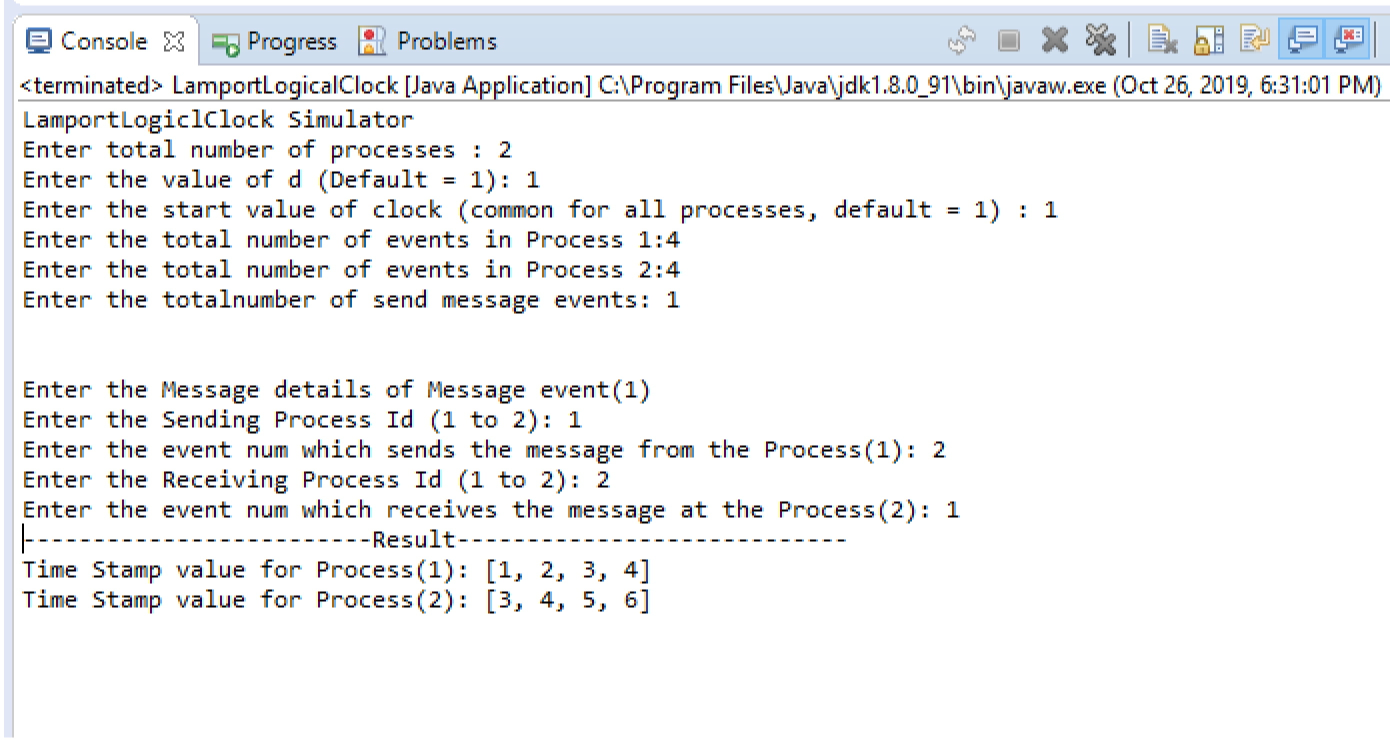
Snapshots :



Console ☒ Progress Problems

<terminated> LamportLogicalClock [Java Application] C:\Program Files\Java\jdk1.8.0_91\bin\javaw.exe (Oct 26, 2019, 6:33:31 PM)

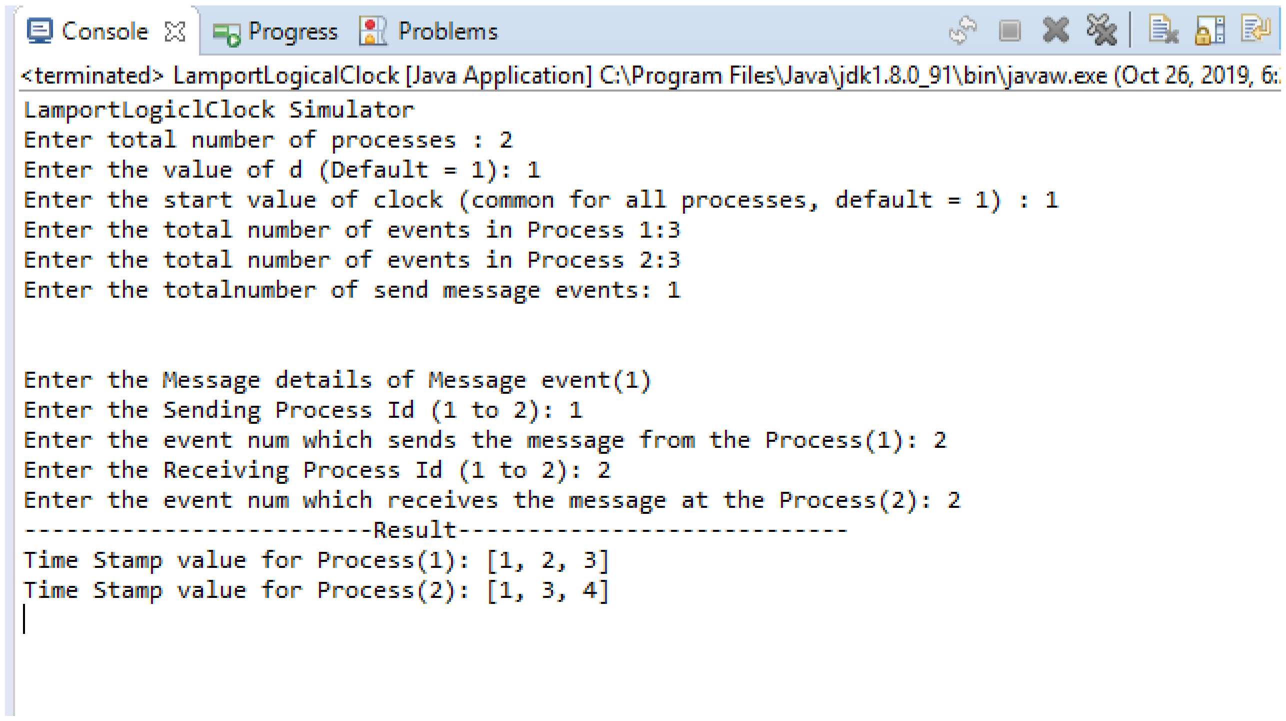
```
LamportLogiclClock Simulator
Enter total number of processes : 3
Enter the value of d (Default = 1): 3
Enter the start value of clock (common for all processes, default = 1) : 4
Enter the total number of events in Process 1:4
Enter the total number of events in Process 2:4
Enter the total number of events in Process 3:4
Enter the totalnumber of send message events: 2


Enter the Message details of Message event(1)
Enter the Sending Process Id (1 to 3): 1
Enter the event num which sends the message from the Process(1): 2
Enter the Receiving Process Id (1 to 3): 2
Enter the event num which receives the message at the Process(2): 2


Enter the Message details of Message event(2)
Enter the Sending Process Id (1 to 3): 3
Enter the event num which sends the message from the Process(3): 1
Enter the Receiving Process Id (1 to 3): 2
Enter the event num which receives the message at the Process(2): 3
-------------------------Result---------------------------
Time Stamp value for Process(1): [4, 7, 10, 13]
Time Stamp value for Process(2): [4, 10, 13, 16]
Time Stamp value for Process(3): [4, 7, 10, 13]
```



Console ☒ Progress Problems

<terminated> LamportLogicalClock [Java Application] C:\Program Files\Java\jdk1.8.0_91\bin\javaw.exe (Oct 26, 2019, 6:31:01 PM)

```
LamportLogiclClock Simulator
Enter total number of processes : 2
Enter the value of d (Default = 1): 1
Enter the start value of clock (common for all processes, default = 1) : 1
Enter the total number of events in Process 1:4
Enter the total number of events in Process 2:4
Enter the totalnumber of send message events: 1


Enter the Message details of Message event(1)
Enter the Sending Process Id (1 to 2): 1
Enter the event num which sends the message from the Process(1): 2
Enter the Receiving Process Id (1 to 2): 2
Enter the event num which receives the message at the Process(2): 1
-------------------------Result---------------------------
Time Stamp value for Process(1): [1, 2, 3, 4]
Time Stamp value for Process(2): [3, 4, 5, 6]
```



```
<terminated> LamportLogicalClock [Java Application] C:\Program Files\Java\jdk1.8.0_91\bin\javaw.exe (Oct 26, 2019, 6:
LamportLogiclClock Simulator
Enter total number of processes : 2
Enter the value of d (Default = 1): 1
Enter the start value of clock (common for all processes, default = 1) : 1
Enter the total number of events in Process 1:3
Enter the total number of events in Process 2:3
Enter the totalnumber of send message events: 1


Enter the Message details of Message event(1)
Enter the Sending Process Id (1 to 2): 1
Enter the event num which sends the message from the Process(1): 2
Enter the Receiving Process Id (1 to 2): 2
Enter the event num which receives the message at the Process(2): 2
-------------------------Result----------------------------
Time Stamp value for Process(1): [1, 2, 3]
Time Stamp value for Process(2): [1, 3, 4]
```