Distributed Systems (CS304)

Assignment - 6

**U19CS012**

Simulate RPC (Create any one procedure on remote machine and call it from local machine)

**List of Programs for RPC**

1.) Find out the **Factorial** of given Number.

**[factorial.x]**

program factorial\_PROG{

    version factorial\_VERS{

        int factorial(int)=1;

    }=1;

}=0x4562877;

**Run Command** : rpcgen –a –C factorial.x

* All required files will be created.
* The factorial\_client.c and factorial\_server.c files would be modified as following:

**[factorial\_client.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* "factorial.h"

void

factorial\_prog\_1(char \*host, int x)

{

    CLIENT \*clnt;

    int  \*result\_1;

    int  factorial\_1\_arg;

*#ifndef* DEBUG

    clnt = clnt\_create (host, factorial\_PROG, factorial\_VERS, "udp");

*if* (clnt == NULL) {

        clnt\_pcreateerror (host);

        exit (1);

    }

*#endif*  */\* DEBUG \*/*

    factorial\_1\_arg = x;

    result\_1 = factorial\_1(&factorial\_1\_arg, clnt);

*if* (result\_1 == (int \*) NULL) {

        clnt\_perror (clnt, "call failed");

    }*else*{

        printf("Result : %d! = %d\n",x,\*result\_1);

    }

*#ifndef* DEBUG

    clnt\_destroy (clnt);

*#endif*   */\* DEBUG \*/*

}

int

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

{

    char \*host;

*if* (argc < 3) {

        printf ("usage: %s server\_host\n", argv[0]);

        exit (1);

    }

    host = argv[1];

    factorial\_prog\_1 (host,atoi(argv[2]));

exit (0);

}

**[factorial\_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* "factorial.h"

int \*

factorial\_1\_svc(int \*argp, struct svc\_req \*rqstp)

{

    static int  result;

*/\**

*\* insert server code here*

*\*/*

    int  temp = 1;

*for*(int i = 1; i <= \*argp; i++)

    {

        temp \*= i;

    }

    printf("Factorial of %d is called\n",\*argp);

    result = temp;

*return* &result;

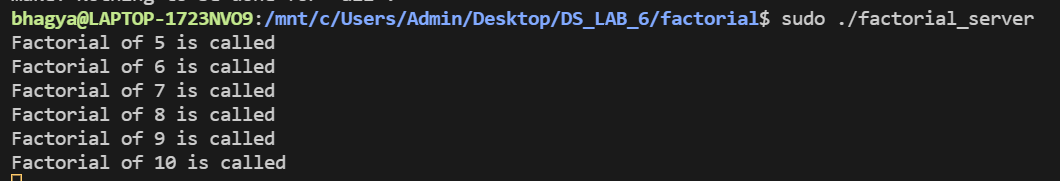
}

**Run Command** : make –f Makefile.factorial

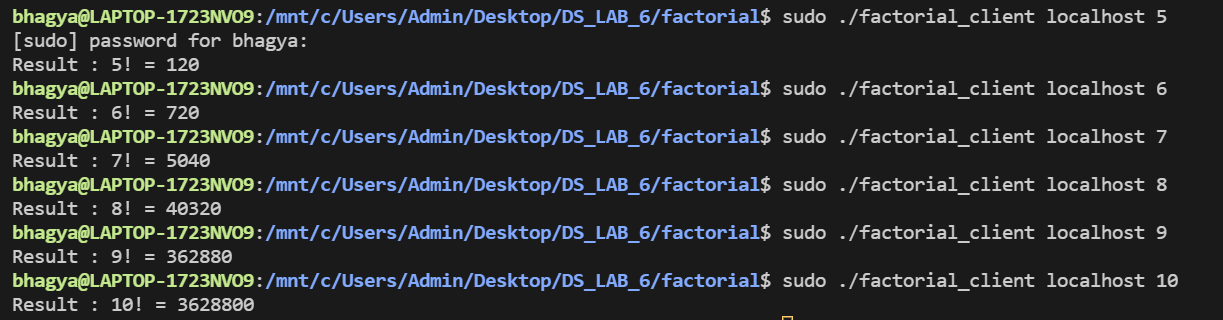
Very IMP Step – **Run Command** : sudo rpcbind

[**Output**]

Server:



Client:



2.) Implement **Calculator** (Basic operation).

**[calc.x]**

struct numbers{

    int a;

    int b;

    char op;

};

program calc\_PROG{

    version calc\_VERS{

        int calc(numbers)=1;

    }=1;

}=0x4562877;

**Run Command** : rpcgen –a –C calc.x

* All required files will be created.
* The calc\_client.c and calc\_server.c files would be modified as following:

**[calc\_client.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* "calc.h"

void calc\_prog\_1(char \*host)

{

    CLIENT \*clnt;

    int \*result\_1;

    numbers calc\_1\_arg;

    char temp;

*#ifndef* DEBUG

    clnt = clnt\_create(host, calc\_PROG, calc\_VERS, "udp");

*if* (clnt == NULL)

    {

        clnt\_pcreateerror(host);

        exit(1);

    }

*#endif* */\* DEBUG \*/*

    printf("Enter 2 numbers: ");

    scanf("%d%d", &(calc\_1\_arg.a), &(calc\_1\_arg.b));

    scanf("%c", &temp);

    printf("Press\n a for addition\n s for subtraction\n m for multiplication\n d for division\n r for modulus\n Choice : ");

    scanf("%c", &(calc\_1\_arg.op));

    result\_1 = calc\_1(&calc\_1\_arg, clnt);

*if* (result\_1 == (int \*)NULL)

    {

        clnt\_perror(clnt, "call failed");

    }

*else*

    {

*if* (\*result\_1 == INT\_MIN && calc\_1\_arg.op == 'd')

        {

            printf("Division by zero is not allowed\n");

        }

*else*

        {

            printf("Result = %d\n", \*result\_1);

        }

    }

*#ifndef* DEBUG

    clnt\_destroy(clnt);

*#endif* */\* DEBUG \*/*

}

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\_prog\_1(host);

    exit(0);

}

**[calc\_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* "calc.h"

int \*calc\_1\_svc(numbers \*argp, struct svc\_req \*rqstp)

{

    static int result;

*if* (argp->op == 'a')

    {

        printf("add(%d,%d)is called\n", argp->a, argp->b);

        result = (argp->a) + (argp->b);

    }

*else* *if* (argp->op == 's')

    {

        printf("sub(%d,%d)is called\n", argp->a, argp->b);

        result = (argp->a) - (argp->b);

    }

*else* *if* (argp->op == 'm')

    {

        printf("mul(%d,%d)is called\n", argp->a, argp->b);

        result = (argp->a) \* (argp->b);

    }

*else* *if* (argp->op == 'r')

    {

        printf("mod(%d,%d)is called\n", argp->a, argp->b);

        result = (argp->a) % (argp->b);

    }

*else*

    {

        printf("div(%d,%d)is called\n", argp->a, argp->b);

*if* (argp->b != 0)

        {

            result = (argp->a) / (argp->b);

        }

*else*

        {

            result = INT\_MIN;

        }

    }

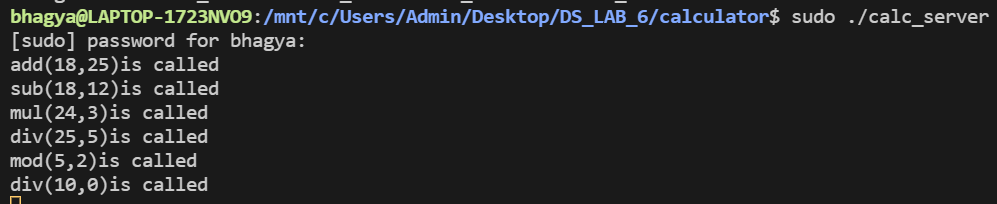
*return* &result;

}

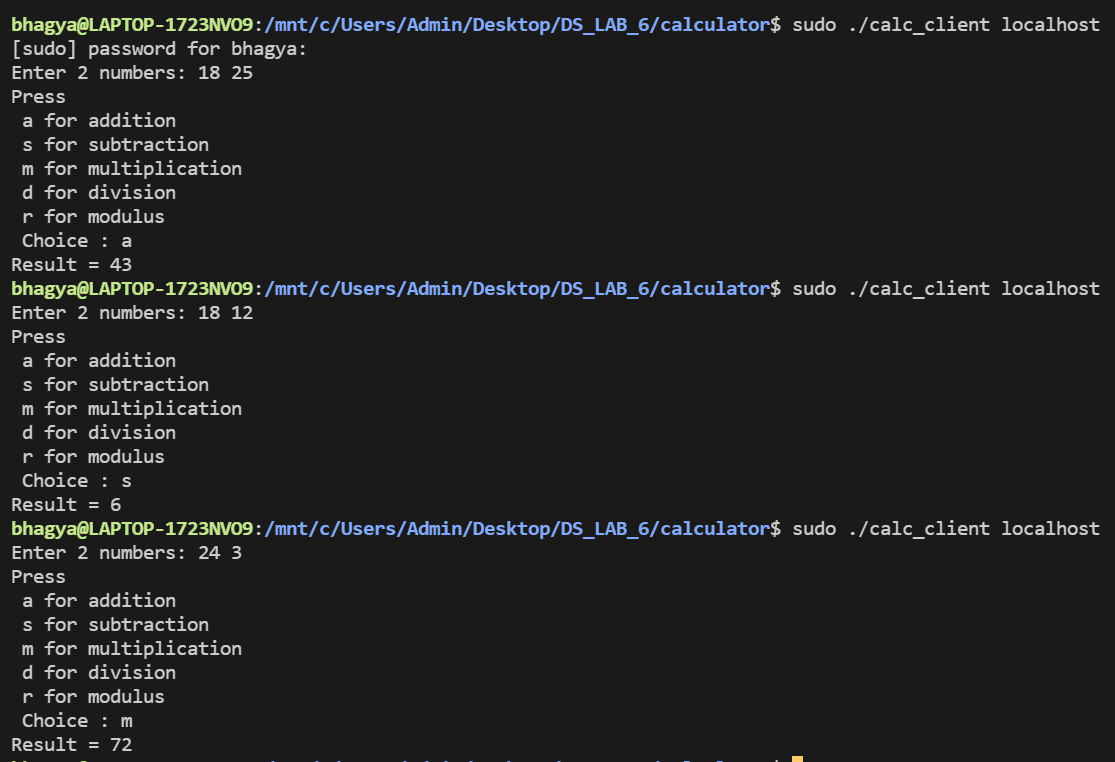
**Run Command** : make –f Makefile.calc

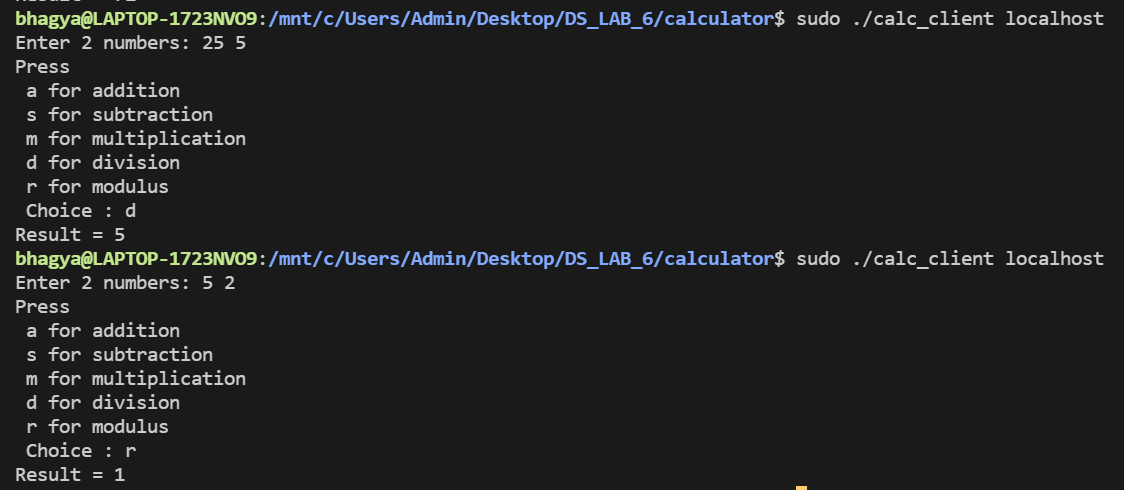
[**Output**]

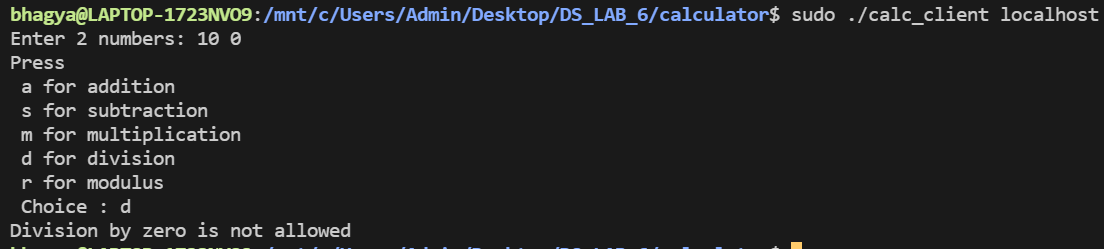
Server:



Client:







3.) Find out whether given number is **Prime** Number or not.

**[prime.x]**

program prime\_PROG{

    version prime\_VERS{

        int prime(int)=1;

    }=1;

}=0x4562877;

**Run Command** : rpcgen –a –C prime.x

* All required files will be created.
* The prime\_client.c and prime\_server.c files would be modified as following:

**[prime\_client.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* "prime.h"

void prime\_prog\_1(char \*host)

{

    CLIENT \*clnt;

    int \*result\_1;

    int prime\_1\_arg;

*#ifndef* DEBUG

    clnt = clnt\_create(host, prime\_PROG, prime\_VERS, "udp");

*if* (clnt == NULL)

    {

        clnt\_pcreateerror(host);

        exit(1);

    }

*#endif* */\* DEBUG \*/*

*do*

    {

        printf("Enter a non-zero postive number: ");

        scanf("%d", &prime\_1\_arg);

    } *while* (prime\_1\_arg < 1);

    result\_1 = prime\_1(&prime\_1\_arg, clnt);

*if* (result\_1 == (int \*)NULL)

    {

        clnt\_perror(clnt, "call failed");

    }

*else*

    {

*if* (\*result\_1 == 0)

            printf("%d is not prime\n", prime\_1\_arg);

*else* *if* (\*result\_1 == 1)

            printf("%d is prime\n", prime\_1\_arg);

*else*

            printf("%d is neither prime nor composite\n", prime\_1\_arg);

    }

*#ifndef* DEBUG

    clnt\_destroy(clnt);

*#endif* */\* DEBUG \*/*

}

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

{

    char \*host;

*if* (argc < 2)

    {

        printf("usage: %s server\_host\n", argv[0]);

        exit(1);

    }

    host = argv[1];

    prime\_prog\_1(host);

    exit(0);

}

**[prime\_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* "prime.h"

int \*prime\_1\_svc(int \*argp, struct svc\_req \*rqstp)

{

    static int result;

    int n = \*argp;

    printf("prime(%d) is called\n", n);

*if* (n == 1)

    {

        result = 2;

*return* &result;

    }

    result = 1;

*for* (int i = 2; i <= n / 2; i++)

    {

*if* (n % i == 0)

        {

            result = 0;

*break*;

        }

    }

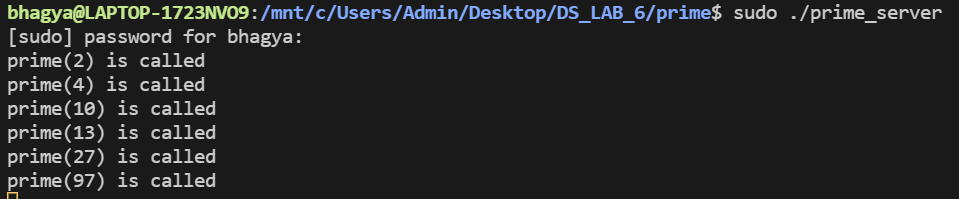
*return* &result;

}

**Run Command** : make –f Makefile.prime

[**Output**]

Server:



Client:



4.) Print out the **Fibonacci** series till the given number.

**[fib.x]**

struct sequence{

    int a[50];

};

program FIB\_PROG{

    version FIB\_VERS{

        sequence fib(int)=1;

    }=1;

}=0x4562877;

**Run Command** : rpcgen –a –C fib.x

* All required files will be created.
* The prime\_client.c and prime\_server.c files would be modified as following:

**[fib\_client.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* "fib.h"

void fib\_prog\_1(char \*host, int x)

{

    CLIENT \*clnt;

    sequence \*result\_1;

    int fib\_1\_arg;

*#ifndef* DEBUG

    clnt = clnt\_create(host, FIB\_PROG, FIB\_VERS, "udp");

*if* (clnt == NULL)

    {

        clnt\_pcreateerror(host);

        exit(1);

    }

*#endif* */\* DEBUG \*/*

    fib\_1\_arg = x;

    result\_1 = fib\_1(&fib\_1\_arg, clnt);

*if* (result\_1 == (sequence \*)NULL)

    {

        clnt\_perror(clnt, "call failed");

    }

*else*

    {

        printf("Fib series upto %d: ", fib\_1\_arg);

        int i = 0;

*while* (1)

        {

*if* (result\_1->a[i] < fib\_1\_arg)

                printf("%d ", result\_1->a[i++]);

*else*

*break*;

        }

        printf("\n");

    }

*#ifndef* DEBUG

    clnt\_destroy(clnt);

*#endif* */\* DEBUG \*/*

}

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

{

    char \*host;

*if* (argc < 3)

    {

        printf("usage: %s server\_host\n", argv[0]);

        exit(1);

    }

    host = argv[1];

    fib\_prog\_1(host, atoi(argv[2]));

    exit(0);

}

**[fib\_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* "fib.h"

sequence \*

fib\_1\_svc(int \*argp, struct svc\_req \*rqstp)

{

    static sequence result;

    result.a[0] = 0;

    result.a[1] = 1;

    int i = 2;

    printf("Fib upto %d is called\n", \*argp);

*while* (result.a[i - 1] < (\*argp))

    {

        result.a[i] = result.a[i - 1] + result.a[i - 2];

        i++;

    }

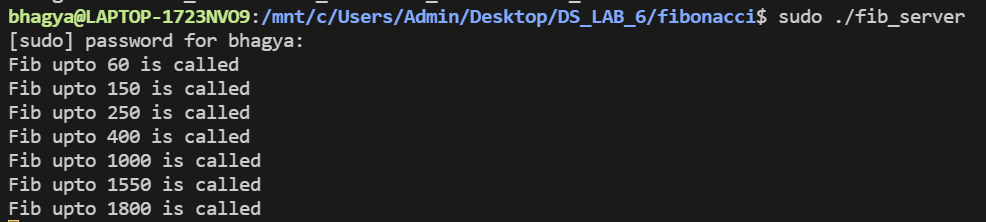
*return* &result;

}

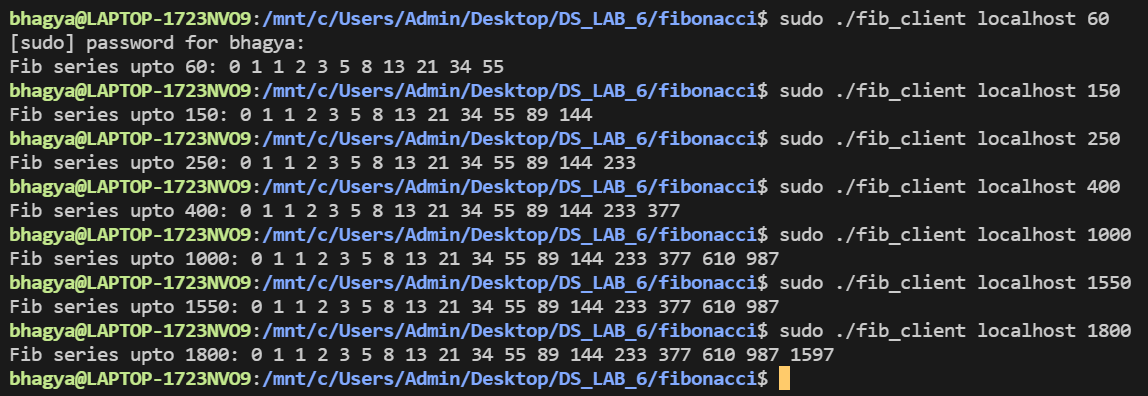
**Run Command** : make –f Makefile.fib

[**Output**]

Server:



Client:



5.) Find the **Maximum** value of an array of integers using RPC.

**[maxArray.x]**

struct numbers{

    int array[100];

    int size;

};

program maxArray\_PROG{

    version maxArray\_VERS{

        int maxArray(numbers)=1;

    }=1;

}=0x4562877;

**Run Command** : rpcgen –a –C **maxArray**.x

* All required files will be created.
* The maxArray\_client.c and maxArray \_server.c files would be modified as following:

**[maxArray\_client.c]**

*#include* "maxArray.h"

void maxarray\_prog\_1(char \*host)

{

    CLIENT \*clnt;

    int \*result\_1;

    numbers maxarray\_1\_arg;

*#ifndef* DEBUG

    clnt = clnt\_create(host, maxArray\_PROG, maxArray\_VERS, "udp");

*if* (clnt == NULL)

    {

        clnt\_pcreateerror(host);

        exit(1);

    }

*#endif* */\* DEBUG \*/*

    printf("Enter the number of elements in the array : ");

    scanf("%d", &maxarray\_1\_arg.size);

    printf("Enter the elements of the array : \n");

*for* (int i = 0; i < maxarray\_1\_arg.size; i++)

    {

        scanf("%d", &maxarray\_1\_arg.array[i]);

    }

    result\_1 = maxarray\_1(&maxarray\_1\_arg, clnt);

*if* (result\_1 == (int \*)NULL)

    {

        clnt\_perror(clnt, "call failed");

    }

*else*

    {

        printf("The maximum element in the array is : %d\n", \*result\_1);

    }

*#ifndef* DEBUG

    clnt\_destroy(clnt);

*#endif* */\* DEBUG \*/*

}

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

{

    char \*host;

*if* (argc < 2)

    {

        printf("usage: %s server\_host\n", argv[0]);

        exit(1);

    }

    host = argv[1];

    maxarray\_prog\_1(host);

    exit(0);

}

**[maxArray\_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* "maxArray.h"

int \*maxarray\_1\_svc(numbers \*argp, struct svc\_req \*rqstp)

{

    static int result;

    printf("The maximum of the array : (");

*for* (int i = 0; i < argp->size; i++)

    {

        printf("%d ", argp->array[i]);

    }

    printf(") is required\n");

    int n = argp->size;

    int \*arr = argp->array;

    int max = arr[0];

*for* (int i = 1; i < n; i++)

    {

*if* (arr[i] > max)

        {

            max = arr[i];

        }

    }

    result = max;

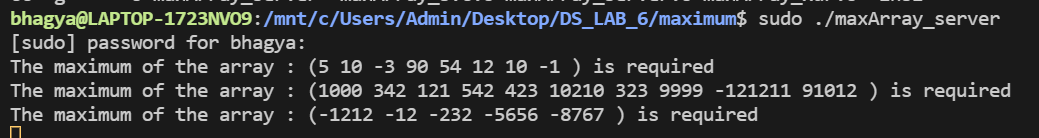
*return* &result;

}

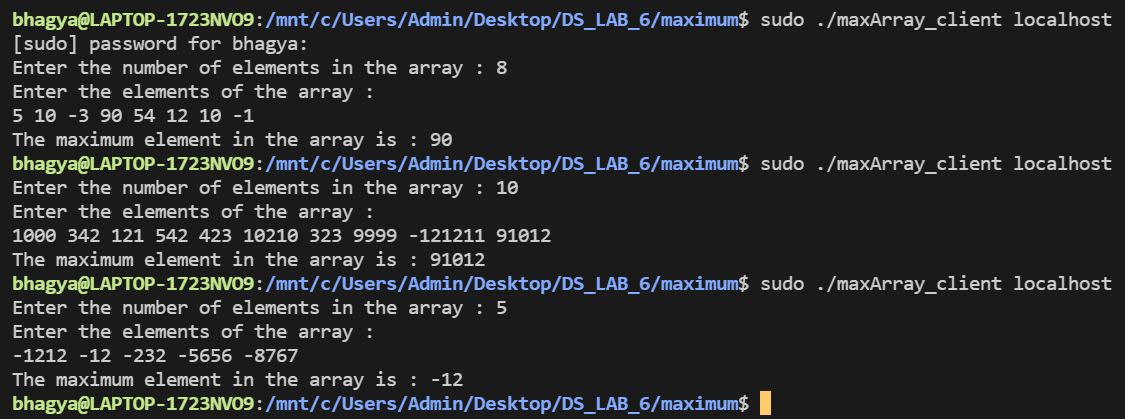
**Run Command** : make –f Makefile.maxArray

[**Output**]

Server:



Client:



**SUBMITTED BY**: U19CS012

BHAGYA VINOD RANA