**DISTRIBUTED COMPONENTS LAB**

**EXERCISE 2**

**PROGRAMS WITH RPCGEN**

**Aim:**

To write programs using SUN - RPCGEN package available in Linux operating system.

**1. Implement a factorial procedure**

**.X CODE:**

program fact {

    version FACT\_VERS {

        int factorial(int)=1;

    }=1;

}=0x23451111;

**SERVER CODE:**

#include "fact.h"

int \*

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

{

    static int result = 1;

    int facter = 1;

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

    for(int i=1;i<=\*argp;i++) {

        facter = facter \* i;

    }

    result = facter;

    return &result;

}

**CLIENT CODE:**

/\*

 \* 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 "fact.h"

void

fact\_1(char \*host)

{

    CLIENT \*clnt;

    int  \*result\_1;

    int  factorial\_1\_arg;

#ifndef DEBUG

    clnt = clnt\_create (host, fact, FACT\_VERS, "udp");

    if (clnt == NULL) {

        clnt\_pcreateerror (host);

        exit (1);

    }

#endif  /\* DEBUG \*/

    while(361) {

        int choice;

        printf("Enter a choice:\n1.Compute factorial\n2.Exit\n");

        scanf("%d",&choice);

        if(choice==1) {

            int num;

            printf("Enter a number: ");

            scanf("%d",&num);

            factorial\_1\_arg = num;

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

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

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

            }

            else {

                printf("The factorial of the number is: %d\n",\*result\_1);

            }

        }

        else {

            exit(0);

        }

    }

#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);

    }

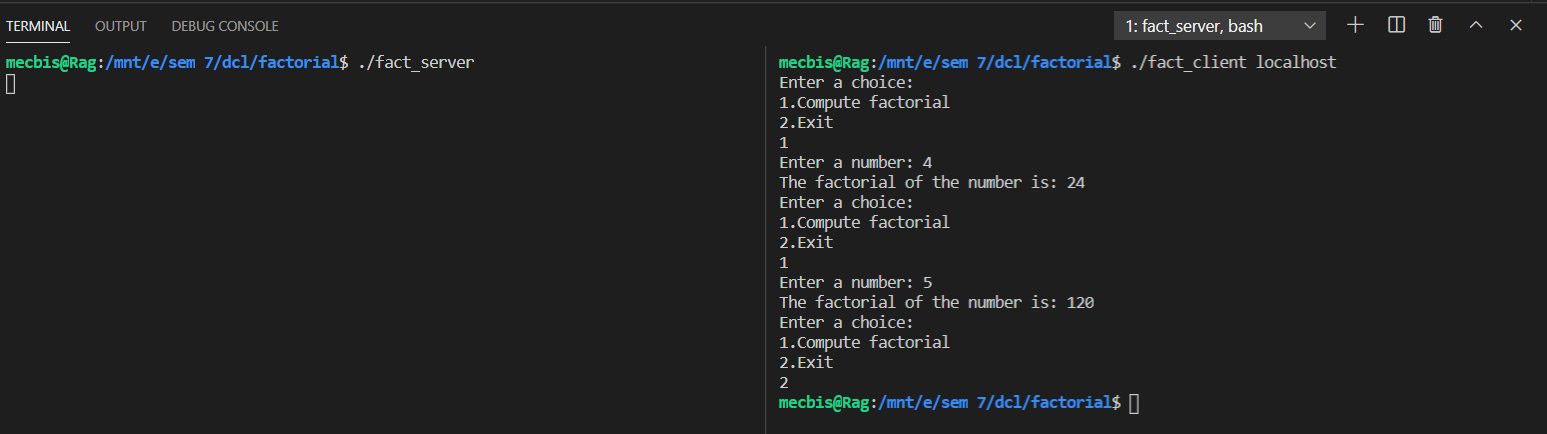
    host = argv[1];

    fact\_1 (host);

exit (0);

}

**OUTPUT:**



**2. Develop a distributed application that implements dictionary lookup operation.**

**Server should maintain a dictionary of words and client interface should keep a cache of recently referenced words and searches the cache before making a remote procedure call. The dictionary may be maintained using string arrays.**

**When the client submits a word, it is checked in the client cache and if it is not found, a request is made to the server to fetch the meaning. Cache may be implemented using string array in the client side.**

**.X CODE:**

struct word {

    char s[100];

};

program dict {

    version DICT\_VERS {

        word search(word)=1;

    }=1;

}=0x23451111;

**SERVER CODE:**

/\*

 \* 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 "dict.h"

word \*

search\_1\_svc(word \*argp, struct svc\_req \*rqstp)

{

    static word result;

    char words[100][100] = {"socket","blockchain","queue","database","UPS"};

    char meaning[100][100] = {"Programming in networks","Decentralized network","A data structure","Storehouse of data","Backup battery"};

    int index = -1;

    printf("Searching for the word...%s\n",argp->s);

    for(int i=0;i<100;i++) {

        if(strcmp(words[i],argp->s)==0) {

            index = i;

            break;

        }

    }

    if(index==-1) {

        printf("Word not found!\n");

        char ret[100] = "NF";

        strcpy(result.s,ret);

    }

    else {

        printf("Word found!\n");

        strcpy(result.s,meaning[index]);

        printf("The meaning returned is: %s\n",meaning[index]);

    }

    return &result;

}

**CLIENT CODE:**

/\*

 \* 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 "dict.h"

void

dict\_1(char \*host)

{

    CLIENT \*clnt;

    word  \*result\_1;

    word  search\_1\_arg;

#ifndef DEBUG

    clnt = clnt\_create (host, dict, DICT\_VERS, "udp");

    if (clnt == NULL) {

        clnt\_pcreateerror (host);

        exit (1);

    }

#endif  /\* DEBUG \*/

    char cache[100][100];

    char meaning[100][100];

    int cache\_size = 0;

    while(361) {

        int choice;

        printf("1. Search for a word\n2. Exit\n");

        scanf("%d",&choice);

        if(choice==1) {

            int flag = 0;

            char finder[100];

            printf("Enter the word to be searched\n");

            scanf("%s",finder);

            printf("The word searched is: %s\n",finder);

            for(int i=0;i<cache\_size;i++) {

                if(strcmp(cache[i],finder)==0) {

                    printf("Word found in cache\n");

                    printf("The meaning is: %s\n",meaning[i]);

                    flag = 1;

                    break;

                }

            }

            if(flag==0) {

                strcpy(search\_1\_arg.s,finder);

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

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

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

                }

                else {

                    printf("Word not found in cache!\n");

                    if(strcmp(result\_1->s,"NF")==0) {

                        printf("Word was not found!\n");

                        continue;

                    }

                    printf("The meaning retrieved from server is: %s\n",result\_1->s);

                    printf("Length is: %ld\n",strlen(result\_1->s));

                    strcpy(cache[cache\_size],finder);

                    strcpy(meaning[cache\_size],result\_1->s);

                    cache\_size++;

                    if(cache\_size==101) {

                        printf("Cache full!!\n");

                        cache\_size -= 5;

                        printf("Deleting last 5 value!\n");

                        printf("Freed up last 5 values successfully!\n");

                    }

                }

            }

        }

        if(choice==2) {

            exit(0);

        }

    }

#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);

}

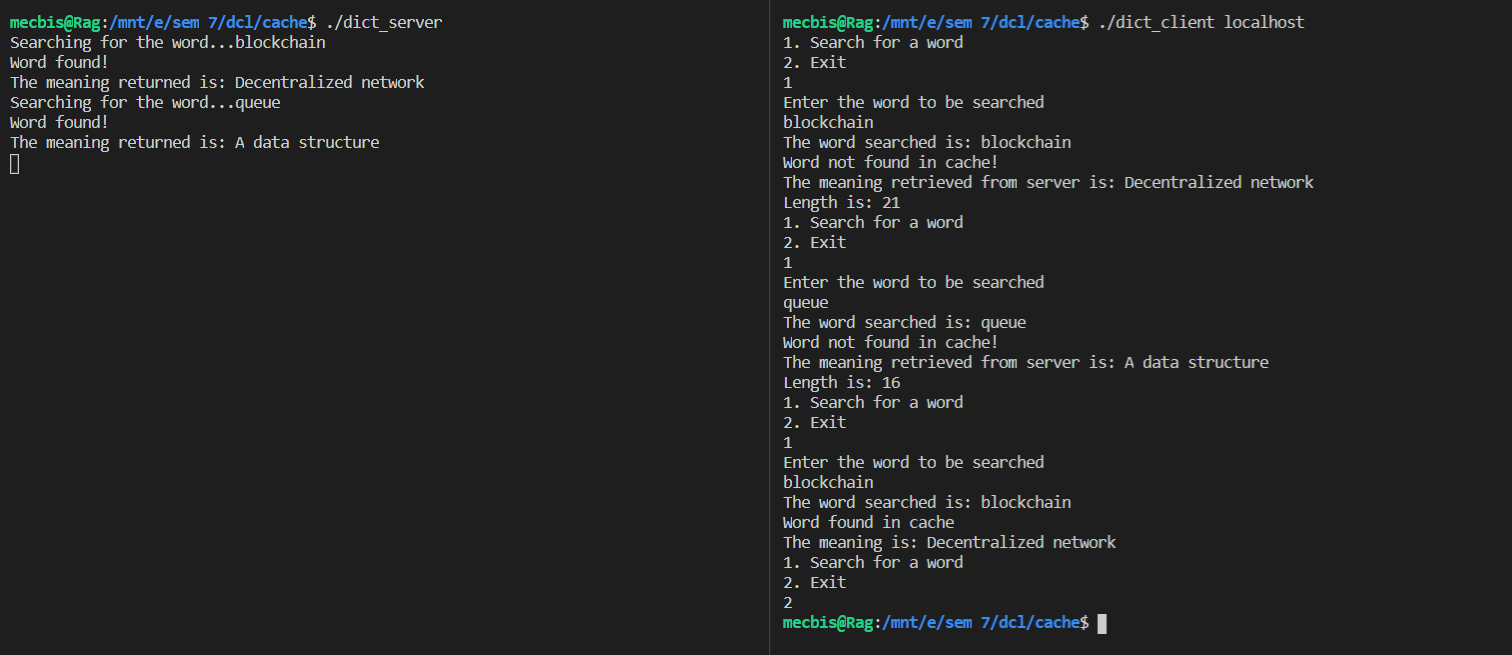
host = argv[1];

    dict\_1 (host);

exit (0);

}

**OUTPUT:**



**3. Online Test**

**.X CODE:**

struct question {

    char q[1000];

};

struct leader {

    char names[100];

    int mark;

};

struct qanda {

    int q;

    int a;

};

struct name {

    char n[100];

};

program test {

    version TEST\_VERS {

        void intialize(name)=1;

        question request(int)=2;

        void judge(qanda)=3;

        int score(name)=4;

        leader leaderboard(int)=5;

    }=1;

}=0x23451111;

**SERVER CODE:**

/\*

 \* 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 "test.h"

#define N 3

char names[100][100];

int scores[100] = {0};

int countPpl = 0;

char questions[100][1000] = {"Where is PSG Tech?\n1.CBE\n2.MAS\n3.PUD\n4.KAN","Where is SVMHSS?\n1.UDT\n2.CBE\n3.MAS\n4.POC","Who is who?\n1.W\n2.L\n3.R\n4.W"};

int answers[100] = {0};

void \*

intialize\_1\_svc(name \*argp, struct svc\_req \*rqstp)

{

    static char \* result;

    answers[0] = 0;

    answers[1] = 0;

    answers[2] = 1;

    printf("Questions initialized and ready to be served!\n");

    strcpy(names[countPpl],argp->n);

    return (void \*) &result;

}

question \*

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

{

    printf("Sending a question!\n");

    static question  result;

    strcpy(result.q,questions[\*argp]);

    return &result;

}

void \*

judge\_1\_svc(qanda \*argp, struct svc\_req \*rqstp)

{

    static char \* result;

    if(answers[argp->q]==argp->a) {

        scores[countPpl]++;

    }

    if(argp->q==N-1) {

        countPpl++;

    }

    return (void \*) &result;

}

int \*

score\_1\_svc(name \*argp, struct svc\_req \*rqstp)

{

    printf("Score released!\n");

    static int  result;

    int index = -1;

    for(int i=0;i<countPpl;i++) {

        if(strcmp(names[i],argp->n)==0) {

            index = i;

            break;

        }

    }

    if(index==-1) {

        result = -1;

    }

    else {

        result = scores[index];

    }

    return &result;

}

leader \*

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

{

    static leader  result;

    if(\*argp==0) {

        printf("Leaderboard released!\n");

    }

    if(\*argp >= countPpl) {

        result.mark = -1;

    }

    else {

        strcpy(result.names,names[\*argp]);

        result.mark = scores[\*argp];

    }

    return &result;

}

**CLIENT CODE;**

/\*

 \* 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 "test.h"

#define N 3

void

test\_1(char \*host)

{

    CLIENT \*clnt;

    void  \*result\_1;

    name  intialize\_1\_arg;

    question  \*result\_2;

    int  request\_1\_arg;

    void  \*result\_3;

    qanda  judge\_1\_arg;

    int  \*result\_4;

    name  score\_1\_arg;

    leader  \*result\_5;

    int  leaderboard\_1\_arg;

#ifndef DEBUG

    clnt = clnt\_create (host, test, TEST\_VERS, "udp");

    if (clnt == NULL) {

        clnt\_pcreateerror (host);

        exit (1);

    }

#endif

    while(361) {

        printf("1.Start your test\n2.Show Leaderboard\n3.Exit\n");

        int choice;

        scanf("%d",&choice);

        if(choice==1) {

            printf("Enter your sweet name\n");

            scanf("%s",intialize\_1\_arg.n);

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

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

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

            }

            else {

                printf("Starting your test\n");

                for(int i=0;i<N;i++) {

                    request\_1\_arg = i;

                    result\_2 = request\_1(&request\_1\_arg, clnt);

                    if (result\_2 == (question \*) NULL) {

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

                    }

                    else {

                        int option;

                        printf("\nYour question %d\n",i+1);

                        printf("%s",result\_2->q);

                        printf("\nEnter your choice: ");

                        scanf("%d",&option);

                        judge\_1\_arg.q = i;

                        judge\_1\_arg.a = option%4;

                        result\_3 = judge\_1(&judge\_1\_arg, clnt);

                        if (result\_3 == (void \*) NULL) {

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

                        }

                    }

                }

                printf("Your test is completed now\n");

                strcpy(score\_1\_arg.n,intialize\_1\_arg.n);

                result\_4 = score\_1(&score\_1\_arg, clnt);

                if (result\_4 == (int \*) NULL) {

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

                }

                printf("Your score is: %d\n",\*result\_4);

                printf("You answered %d out of %d questions\n",\*result\_4,N);

                int sc = \*result\_4;

                printf("Your accuracy is: %f\n",(float)sc/(float)N);

            }

        }

        else if(choice==2) {

            int cur = 0;

            leaderboard\_1\_arg = cur;

            printf("\nName\tScore\n");

            while(361) {

                result\_5 = leaderboard\_1(&leaderboard\_1\_arg, clnt);

                if (result\_5 == (leader \*) NULL) {

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

                }

                if(result\_5->mark==-1) {

                    break;

                }

                else {

                    printf("%s\t%d\n",result\_5->names,result\_5->mark);

                    leaderboard\_1\_arg = ++cur;

                }

            }

            printf("\n");

        }

        else {

            printf("Thank you!\n");

            exit(0);

        }

    }

#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);

    }

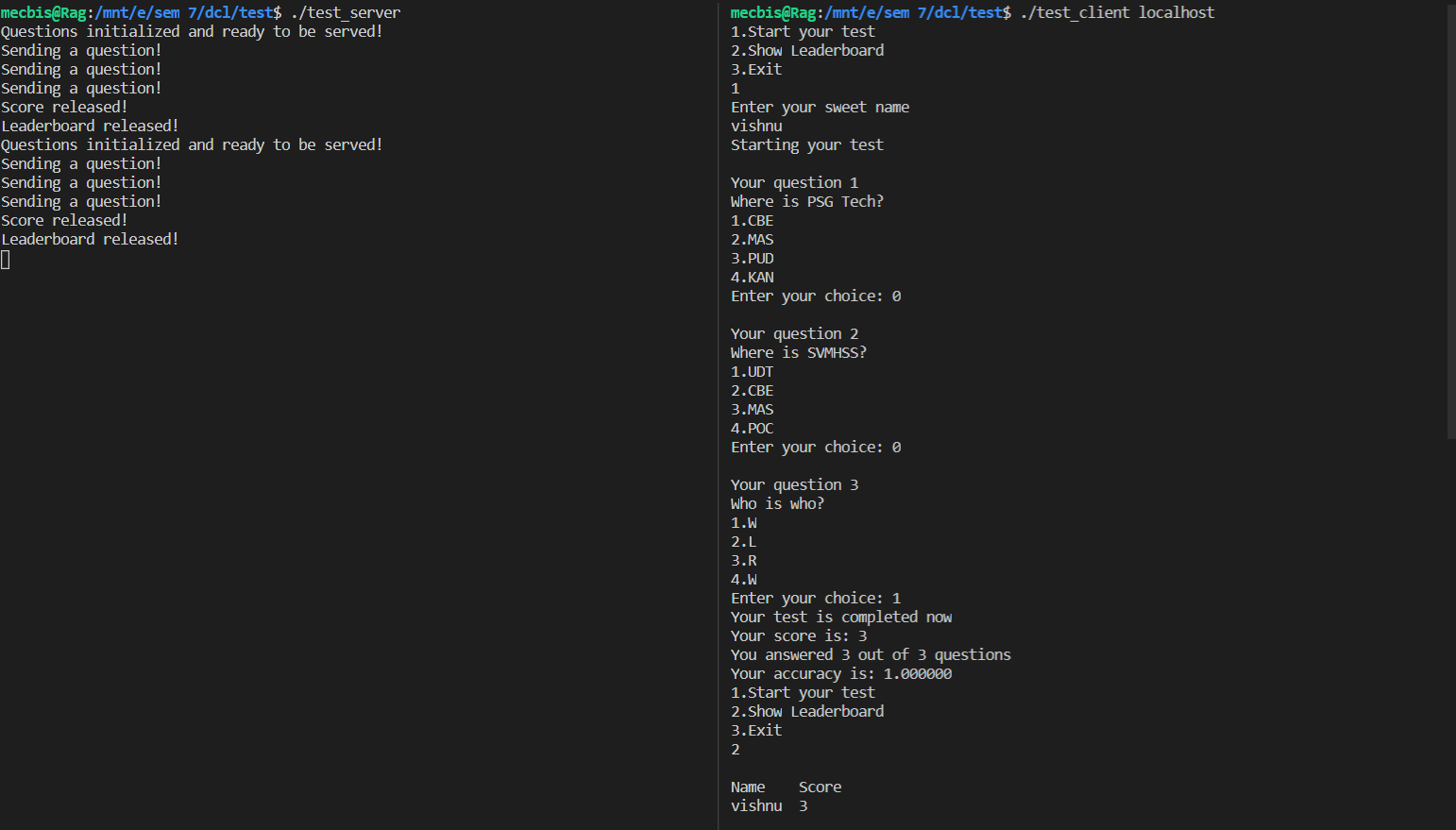
    host = argv[1];

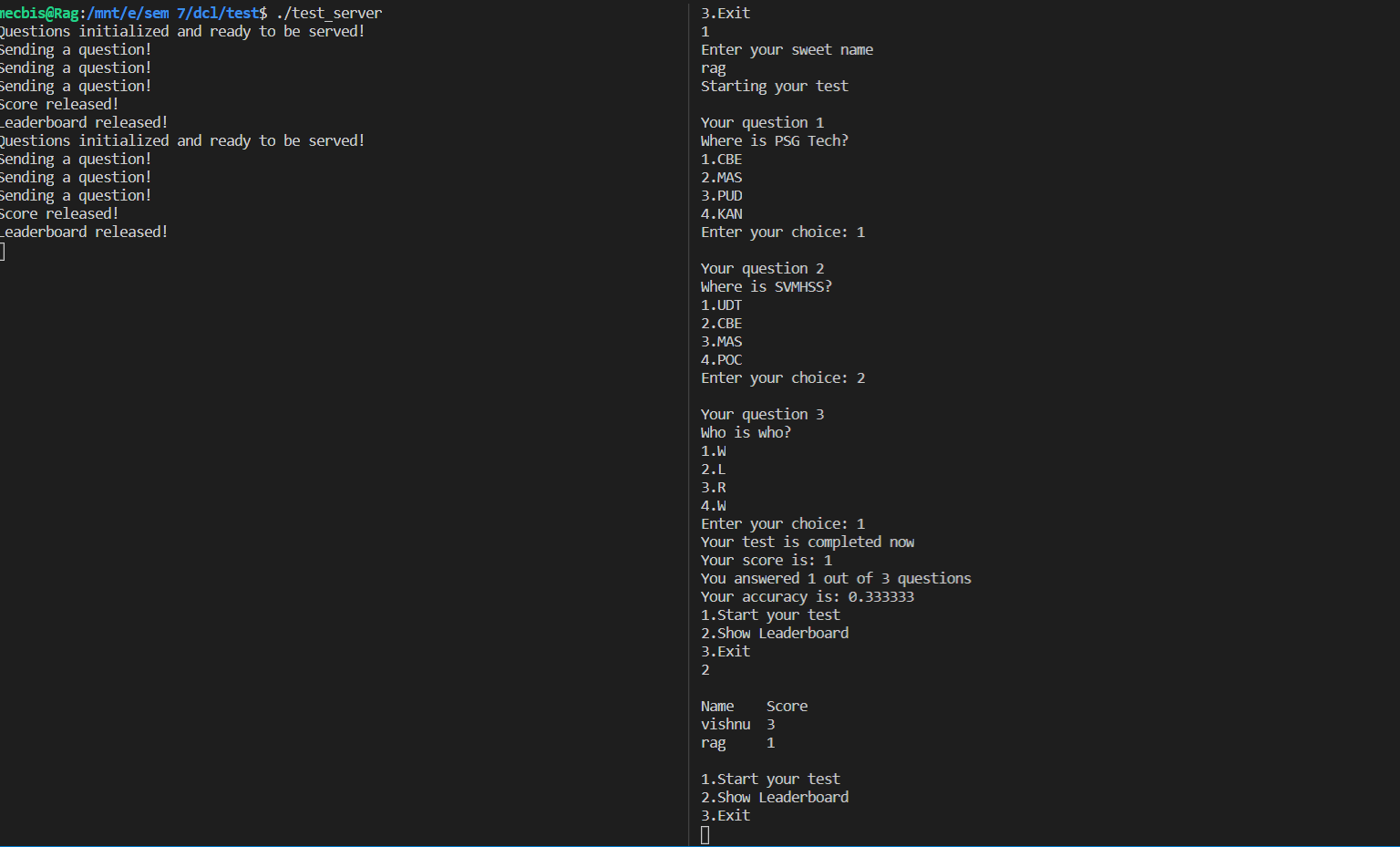
    test\_1 (host);

exit (0);

}

**OUTPUT:**

****

****

**Result:**

The programs are done successfully using the Sun RPCGEN.