

# ASSIGNMENT 1

## INTRODUCTION TO DISTRIBUTED SYSTEMS

UJJWAL SHARMA , BT18CSE021 , [usharma@students.vnit.ac.in](mailto:usharma@students.vnit.ac.in)

### Question 1 - Server Client Chat via R.P.C

Chat.x

Created a struct that will be used both as argument and return parameter so to convey messages

```
≡ chat.x ×  
chatRPC > code > ≡ chat.x  
1 struct Message{  
2     char message[100];  
3 };  
4  
5 program CHAT_PROG{  
6     version CHAT_VERS{  
7         Message chat(Message)=1;  
8     }=1;  
9 }=0x1231231;
```

Then run `rpcgen -aC chat.x` to generate corresponding C files.  
Then edit server and client files

chat\_server.c

C chat\_server.c X

chatRPC &gt; code &gt; C chat\_server.c

```
1  /*
2  * This is sample code generated by rpcgen.
3  * These are only templates and you can use them
4  * as a guideline for developing your own functions.
5  */
6
7  #include "chat.h"
8
9  Message *
10 chat_1_svc(Message *argp, struct svc_req *rqstp)
11 {
12     static Message  result;
13
14     /*
15      * insert server code here
16      */
17     printf("client ->> %s\n" , argp->message);
18     printf("server ->> ");
19     fgets(result.message, 100, stdin);
20     // printf("%s\n", result.message);
21
22     return &result;
23 }
24
```

chat\_client.c

C chat\_client.c ×

chatRPC &gt; code &gt; C chat\_client.c

```
39
40  int
41  main (int argc, char *argv[])
42  {
43      char *host;
44
45      if (argc < 2) {
46          printf ("usage: %s server_host\n", argv[0]);
47          exit (1);
48      }
49      host = argv[1];
50      printf("press ctrl + c to exit\n");
51      while(1){
52          chat_prog_1 (host);
53      }
54
55      exit (0);
56  }
57
```

Function : - chat\_prog\_1

```
#endif /* DEBUG */
printf("client ->> ");
fgets(chat_1_arg.message, 100, stdin);

// printf("%s\n" , chat_1_arg.message);
result_1 = chat_1(&chat_1_arg, clnt);
if (result_1 == (Message *) NULL) {
    clnt_perror (clnt, "call failed");
}else{
    printf("server >> %s\n", result_1->message);
}

#ifdef DEBUG
    clnt_destroy (clnt);
#endif /* DEBUG */
}
```

# Output Of Question 1

```
>_ Terminal x +
$ ./chat_client localhost
press ctrl + c to exit
client ->> this is hi from client

>_ Terminal x +
$ ./chat_server
[]

>_ Terminal x +
$ ./chat_client localhost
press ctrl + c to exit
client ->> this is hi from client

>_ Terminal x +
$ ./chat_server
client ->> this is hi from client

server ->> []

>_ Terminal x +
$ ./chat_client localhost
press ctrl + c to exit
client ->> this is hi from client
server >> This is response from server
client ->> []

>_ Terminal x +
$ ./chat_server
client ->> this is hi from client

server ->> This is response from server
client ->> this is hi from client

server ->>
```

## Question 2 - Factorial using R.P.C

Factorial.x

≡ factorial.x ✕

factorialRPC > ≡ factorial.x

```
1 struct number{
2     int fact;
3 };
4
5 program FACT_PROG{
6     version FACT_VERS{
7         int factorial(number)=1;
8     }=1;
9 }=0x1231232;
```

Factorial\_server.c

C factorial\_server.c ✕

factorialRPC > code > C factorial\_server.c

```
8
9 int calculateFactorial(int fact){
10     if(fact < 0){
11         return 0;
12     }
13     int result = 1;
14     for(int i=1; i<=fact; i++){
15         result*=i;
16     }
17     return result;
18 }
19
20 int *
21 factorial_1_svc(number *argp, struct svc_req *rqstp)
22 {
23     static int result;
24
25     /*
26      * insert server code here
27      */
28     printf("Calculating Factorial of %d ...\n", argp->fact);
29     result = calculateFactorial(argp->fact);
30     printf("Done\n");
31
32     return &result;
33 }
34
```

factorial\_client.c

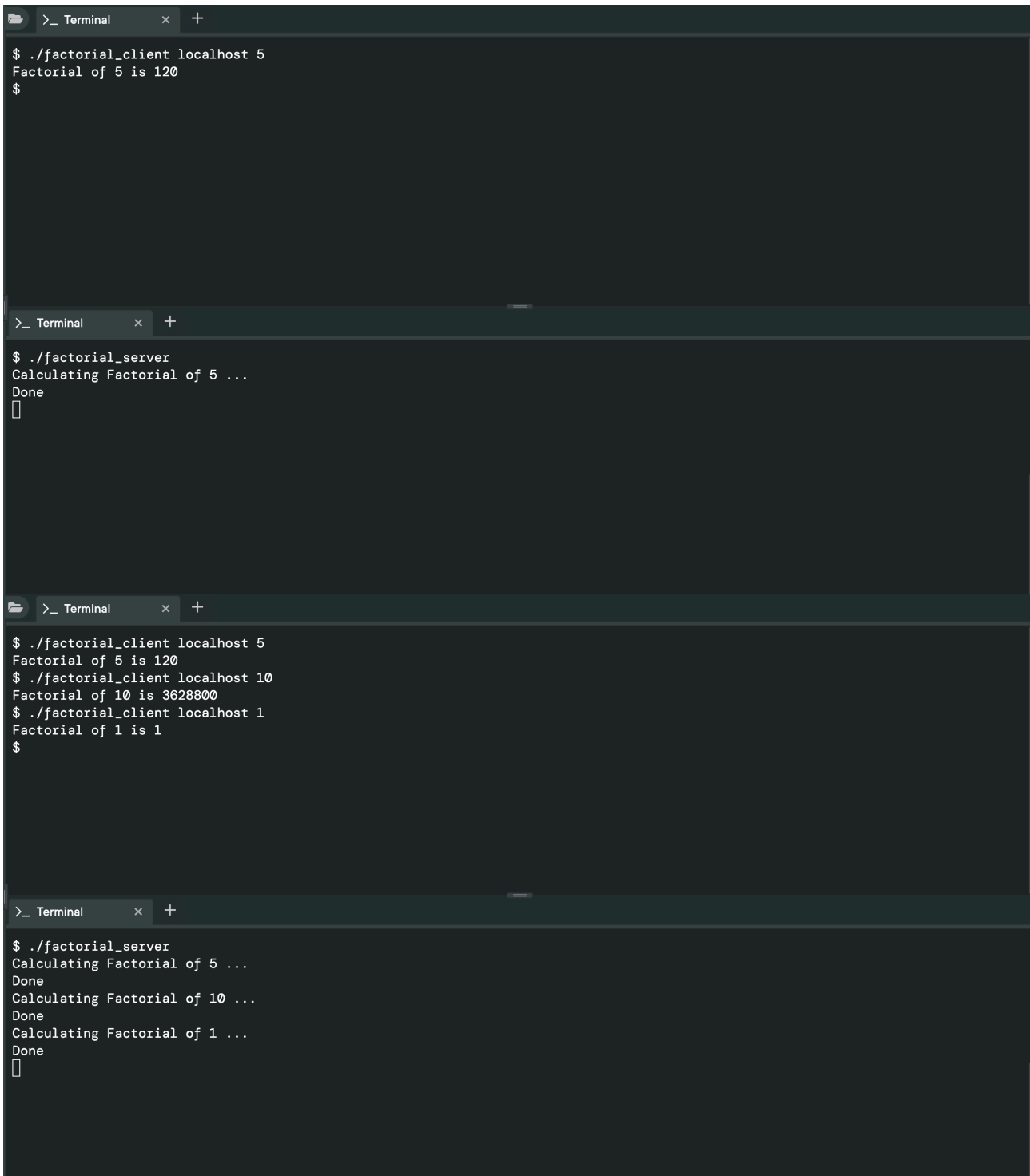
Function - fact\_prgm\_1

C factorial\_client.c ×

factorialRPC &gt; code &gt; C factorial\_client.c

```
22     }
23     #endif /* DEBUG */
24     factorial_1_arg.fact = fact;
25     result_1 = factorial_1(&factorial_1_arg, clnt);
26     if (result_1 == (int *) NULL) {
27         clnt_perror (clnt, "call failed");
28     }else{
29         printf("Factorial of %d is %d\n",fact , *result_1);
30     }
31     #ifndef DEBUG
32         clnt_destroy (clnt);
33     #endif /* DEBUG */
34 }
35
36
37 int
38 main (int argc, char *argv[])
39 {
40     char *host;
41
42     if (argc != 3) {
43         printf ("usage: %s server_host NUMBER\n", argv[0]);
44         exit (1);
45     }
46     host = argv[1];
47     fact_prog_1 (host , atoi(argv[2]));
48     exit (0);
49 }
50
```

## Output of answer 2



The image displays three sequential terminal window screenshots, each with a title bar that reads ">\_ Terminal" and window control buttons. The first screenshot shows a client command: `$. ./factorial_client localhost 5`, resulting in the output: `Factorial of 5 is 120`. The second screenshot shows a server command: `$. ./factorial_server`, resulting in the output: `Calculating Factorial of 5 ...`, `Done`, and a small square icon. The third screenshot shows a series of client commands: `$. ./factorial_client localhost 5`, `$. ./factorial_client localhost 10`, and `$. ./factorial_client localhost 1`, with corresponding outputs: `Factorial of 5 is 120`, `Factorial of 10 is 3628800`, and `Factorial of 1 is 1`. The server window in the third screenshot shows the output: `Calculating Factorial of 5 ...`, `Done`, `Calculating Factorial of 10 ...`, `Done`, `Calculating Factorial of 1 ...`, `Done`, and the small square icon.

```
>_ Terminal x +
$. ./factorial_client localhost 5
Factorial of 5 is 120
$

>_ Terminal x +
$. ./factorial_server
Calculating Factorial of 5 ...
Done
□

>_ Terminal x +
$. ./factorial_client localhost 5
Factorial of 5 is 120
$. ./factorial_client localhost 10
Factorial of 10 is 3628800
$. ./factorial_client localhost 1
Factorial of 1 is 1
$

>_ Terminal x +
$. ./factorial_server
Calculating Factorial of 5 ...
Done
Calculating Factorial of 10 ...
Done
Calculating Factorial of 1 ...
Done
□
```

```
>_ Terminal x +  
$ ./factorial_client localhost 5  
Factorial of 5 is 120  
$ ./factorial_client localhost 10  
Factorial of 10 is 3628800  
$
```

```
>_ Terminal x +  
$ ./factorial_server  
Calculating Factorial of 5 ...  
Done  
Calculating Factorial of 10 ...  
Done  
$
```

```
>_ Terminal x +  
$ ./factorial_client localhost 5  
Factorial of 5 is 120  
$ ./factorial_client localhost 10  
Factorial of 10 is 3628800  
$ ./factorial_client localhost 1  
Factorial of 1 is 1  
$ ./factorial_client localhost 0  
Factorial of 0 is 1  
$
```

```
>_ Terminal x +  
$ ./factorial_server  
Calculating Factorial of 5 ...  
Done  
Calculating Factorial of 10 ...  
Done  
Calculating Factorial of 1 ...  
Done  
Calculating Factorial of 0 ...  
Done  
$
```

```
>_ Terminal x +  
$ ./factorial_client localhost 5  
Factorial of 5 is 120  
$ ./factorial_client localhost 10  
Factorial of 10 is 3628800  
$ ./factorial_client localhost 1  
Factorial of 1 is 1  
$ ./factorial_client localhost 0  
Factorial of 0 is 1  
$ ./factorial_client localhost -10  
Factorial of -10 is 0  
$
```

```
>_ Terminal x +  
$ ./factorial_server  
Calculating Factorial of 5 ...  
Done  
Calculating Factorial of 10 ...  
Done  
Calculating Factorial of 1 ...  
Done  
Calculating Factorial of 0 ...  
Done  
Calculating Factorial of -10 ...  
Done  
$
```



### Question 3 - Print date , time of the server using R.P.C

dateTime.x

```
≡ dateTime.x ×  
dateTimeRPC > ≡ dateTime.x  
1  program TIME_PROG{  
2      version TIME_VERS{  
3          long dateTime(void)=1;  
4      }=1;  
5  
6  }= 0x1231233;
```

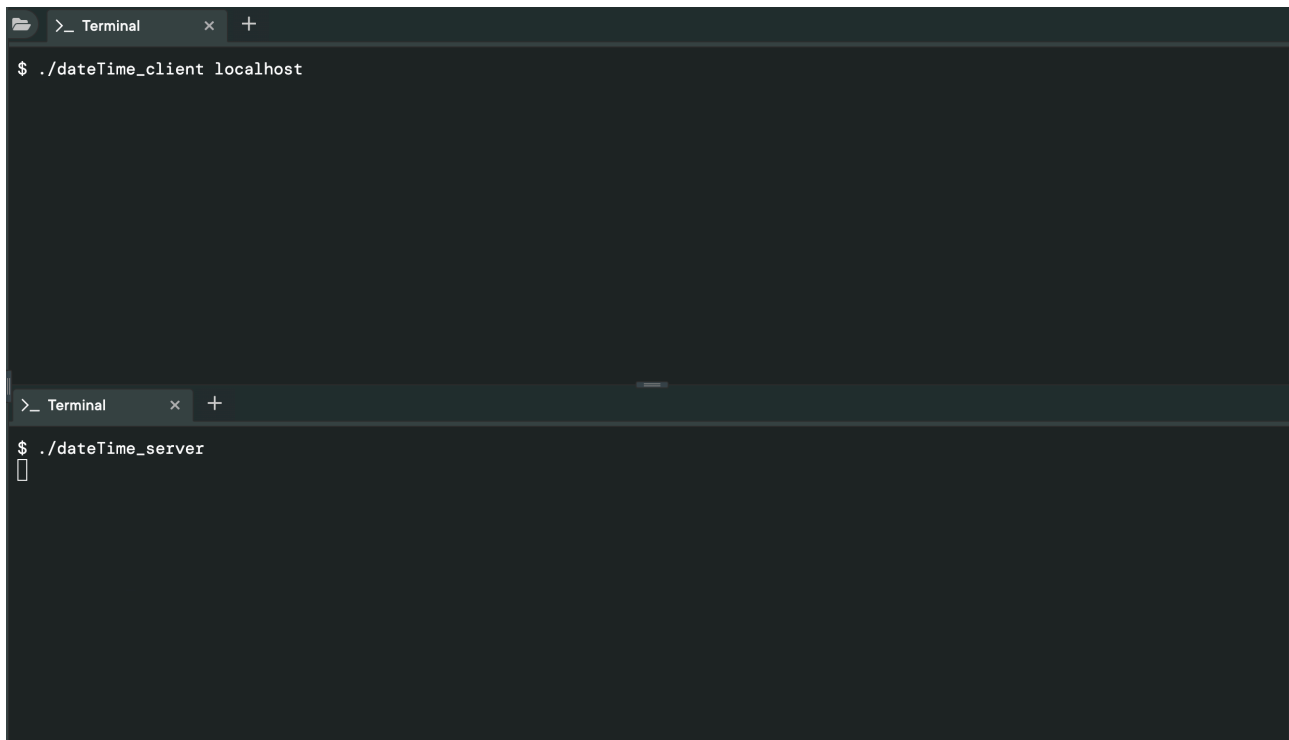
dateTime\_server.c

```
C dateTime_server.c ×  
dateTimeRPC > code > C dateTime_server.c  
1  /*  
2      * This is sample code generated by rpcgen.  
3      * These are only templates and you can use them  
4      * as a guideline for developing your own functions.  
5      */  
6  
7  #include "dateTime.h"  
8  #include<time.h>  
9  
10 long *  
11 datetime_1_svc(void *argp, struct svc_req *rqstp)  
12 {  
13     static long result;  
14  
15     /*  
16      * insert server code here  
17      */  
18     printf("Sending current date and time . \n");  
19     result = time((long*)0);  
20     return &result;  
21 }  
22
```

dateTime\_client.c

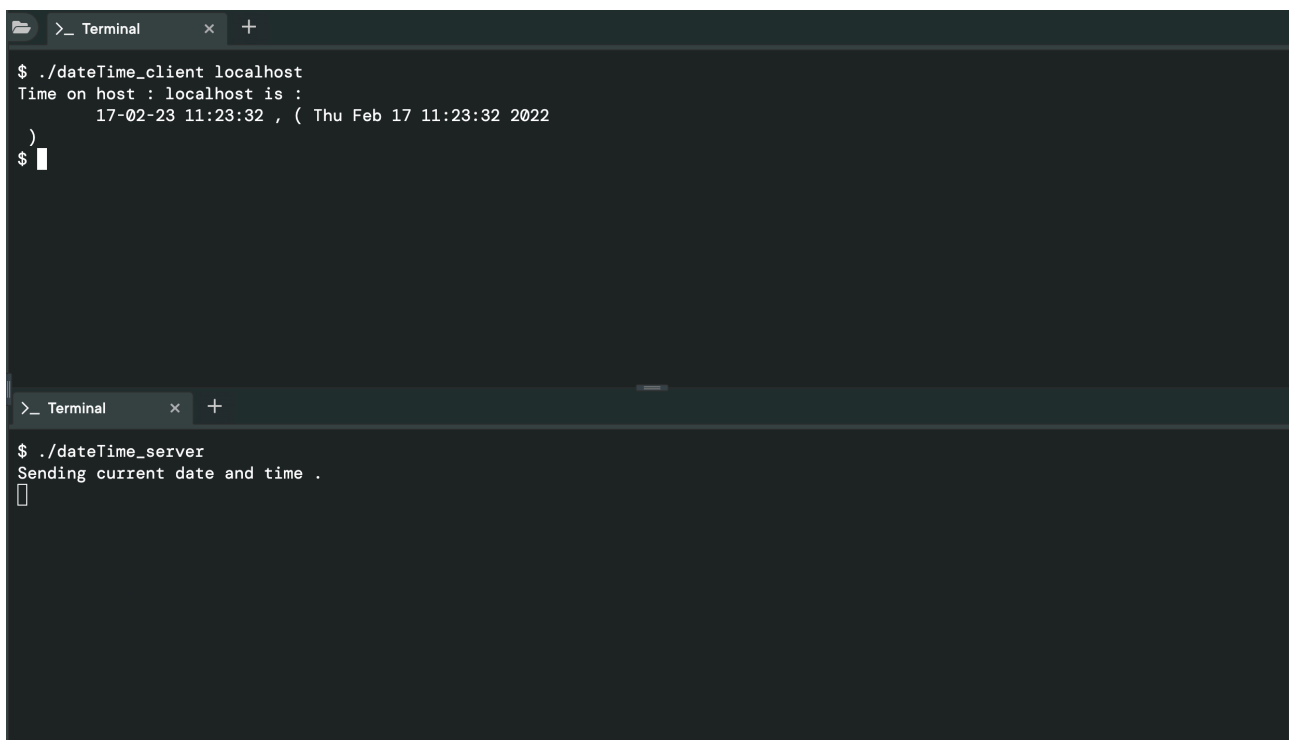
```
C dateTime_client.c ×
dateTimeRPC > code > C dateTime_client.c
28     }else{
29
30
31     char dateTimeBuffer[100];
32     strftime (dateTimeBuffer, 100, "%d-%m-%M %H:%M:%S", localtime (result_1));
33
34     printf("Time on host : %s is :\n\t%s , ( %s )\n", host , dateTimeBuffer , ctime(result_1));
35     }
36 #ifndef DEBUG
37     clnt_destroy (clnt);
38 #endif /* DEBUG */
39 }
40
41
42 int
43 main (int argc, char *argv[])
44 {
45     char *host;
46
47     if (argc < 2) {
48         printf ("usage: %s server_host\n", argv[0]);
49         exit (1);
50     }
51     host = argv[1];
52     time_prog_1 (host);
53     exit (0);
54 }
55
```

## Output of answer 3



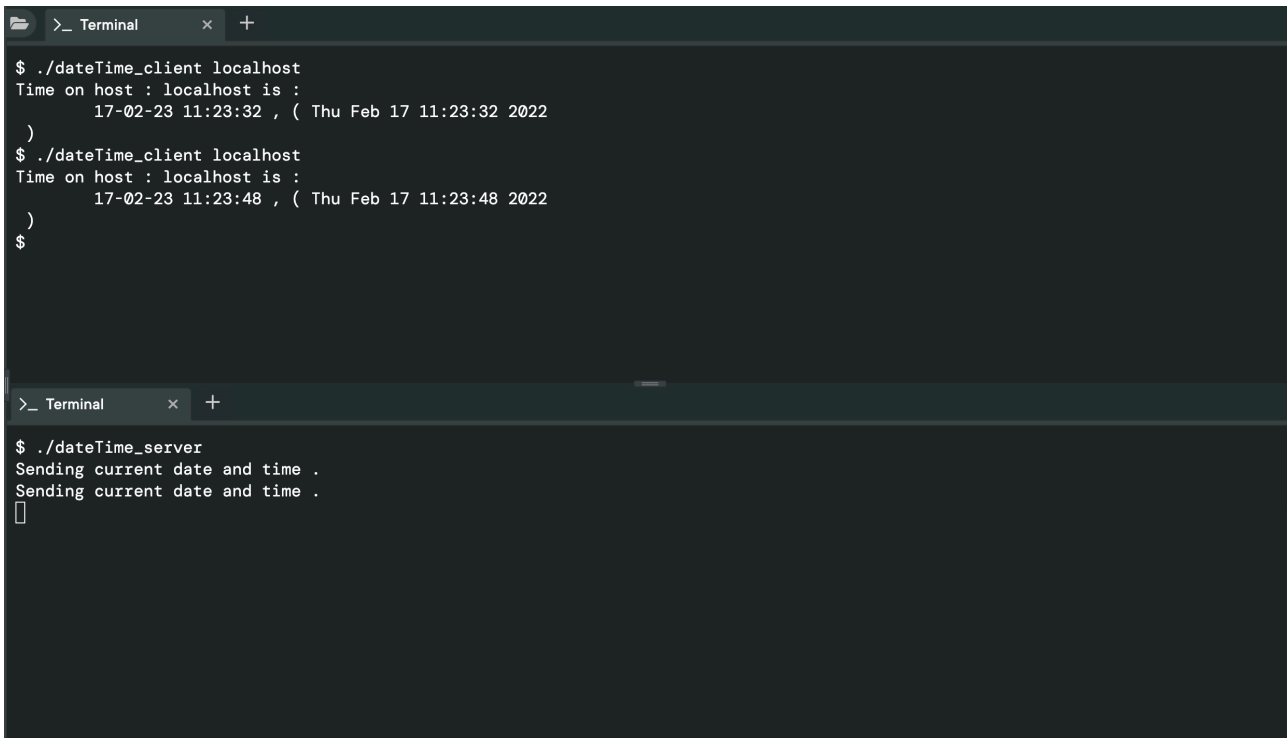
```
>_ Terminal x +
$ ./dateTime_client localhost

>_ Terminal x +
$ ./dateTime_server
█
```



```
>_ Terminal x +
$ ./dateTime_client localhost
Time on host : localhost is :
    17-02-23 11:23:32 , ( Thu Feb 17 11:23:32 2022
)
$ █

>_ Terminal x +
$ ./dateTime_server
Sending current date and time .
█
```



```
>_ Terminal x +
$ ./dateTime_client localhost
Time on host : localhost is :
    17-02-23 11:23:32 , ( Thu Feb 17 11:23:32 2022
)
$ ./dateTime_client localhost
Time on host : localhost is :
    17-02-23 11:23:48 , ( Thu Feb 17 11:23:48 2022
)
$

>_ Terminal x +
$ ./dateTime_server
Sending current date and time .
Sending current date and time .
█
```

**UJJWAL SHARMA**

**BT18CSE021**

**[usharma@students.vnit.ac.in](mailto:usharma@students.vnit.ac.in)**

\*\*\*\*\* END \*\*\*\*\*