Ziqi Zhao, Varsa Ijeri, Swapnil Acharya

StarId: ea4963aw Due: 03/02/2020

Purpose

The objective of this project is to use RPC as the means of communication.

CSCI 610  
 Advanced concepts in operating systems

Project 3- Group

**Man Page Documentation**

This project demonstrates the use of Remote Procedure Call (RPC) as the means of communication. In this project a token is passed back and forth repeatedly between the client and server. A “primeToken.x” file was designed containing a struct which is used to keep track of the number of hops it has passed over, time at which this token was generate, highest prime number found, number of primes found and next integer to start prime number search from. After designing “primeToken.x”, it was time to generate rpc using command “rpcgen -C primeToken.x”, this command generates files:

1. primeToken\_clnt.c(client stub),
2. primeToken\_svc.c(server stub),
3. primeToken\_xdr.c
4. primeToken.h

The client and server stub files which are responsible for translating the transmitted data to and from the XDR (external data representation) format. After that “primeToken\_server.c” “primeToken\_client.c” were designed.

**To compile and execute the project, do the following steps:**

1. Open coursework folder for CSCI 610 class and then open student work folder
2. Inside Student work folder, open ea4963aw folder
3. Inside ea4963aw, open Project 3 folder
4. Compile the code in a Linux terminal using command: make
5. Start RPC sever by entering following command in terminal: ./server &
6. Start RPC client by entering following command in terminal: ./client localhost
7. Once the client code exists, to view running server executable type: ps
8. Then kill the running server process by using command: kill processId#
9. Clean files by entering following command in terminal: make clean

**Makefile**

#@since 3/1/2020

#@file Makefile

all**:** generateRpc client server

generateRpc**:** primeToken.x

rpcgen -C primeToken.x

client**:** primeToken.h primeToken\_client.c primeToken\_clnt.c primeToken\_xdr.c

gcc primeToken.h primeToken\_client.c primeToken\_clnt.c primeToken\_xdr.c -o client

server**:** primeToken.h primeToken\_server.c primeToken\_svc.c primeToken\_xdr.c

gcc primeToken.h primeToken\_server.c primeToken\_svc.c primeToken\_xdr.c -o server

clean**:**

rm -f client

rm -f server

rm -f primeToken\_clnt.c

rm -f primeToken\_svc.c

rm -f primeToken\_xdr.c

rm -f primeToken.h

**primeToken.x**

const LIMIT = 100000;

struct Tkn{

int \_numberOfHops;

hyper int \_timeOfCreation;

int \_nextInteger;

int \_numberOfPrimesFound;

int \_highestPrimeFound;

};

typedef struct Tkn Token;

program PRIME\_COMPUTATION{

version PCOMP\_1{

Token START\_PRIME\_COMPUTATION(Token) = 1;

} = 1;

} = 0x20000001;

**primeToken\_client.c**

/\*\*

\***@since** 3/1/2020

\***@file** primeToken\_client.c

\* This file demonstrates the client part of RPC based client-server

\* communication

\*\*/

#include "primeToken.h"

#include <stdio.h>

#include <time.h>

#include <stdbool.h>

/\*\*

\* This function prints the details of token after being processed by client/

\* before being passed to server.

\***@param** \_tkn The token structure whose detail is to be printed

\***@return** nothing

\*/

void printTokenDetailsBefore**(**Token \_tkn**){**

printf**(**"\n\n Token Before Server\n"**);**

printf**(**"Client:-- Hops: %i\n"**,**\_tkn**.**\_numberOfHops**);**

printf**(**"Client:-- Time: %i\n"**,**\_tkn**.**\_timeOfCreation**);**

printf**(**"Client:-- Next Int: %i\n"**,**\_tkn**.**\_nextInteger**);**

printf**(**"Client:-- Num Prime: %i\n"**,**\_tkn**.**\_numberOfPrimesFound**);**

printf**(**"Client:-- Highest: %i\n"**,**\_tkn**.**\_highestPrimeFound**);**

printf**(**"\n\n"**);**

**}**

/\*\*

\* This function prints the details of token after being processed by server/

\* before being passed to client.

\***@param** \_tkn The token structure whose detail is to be printed

\***@return** nothing

\*/

void printTokenDetailsAfter**(**Token \_tkn**){**

printf**(**"\n\n Token After Server\n"**);**

printf**(**"Client:-- Hops: %i\n"**,**\_tkn**.**\_numberOfHops**);**

printf**(**"Client:-- Time: %i\n"**,**\_tkn**.**\_timeOfCreation**);**

printf**(**"Client:-- Next Int: %i\n"**,**\_tkn**.**\_nextInteger**);**

printf**(**"Client:-- Num Prime: %i\n"**,**\_tkn**.**\_numberOfPrimesFound**);**

printf**(**"Client:-- Highest: %i\n"**,**\_tkn**.**\_highestPrimeFound**);**

printf**(**"\n\n"**);**

**}**

/\*\*

\* This function checks where a given number si prime or not

\***@param** \_num This parameter is checked if its prime or not

\***@return** true if the param a is prime, false otherwise

\*/

bool isPrimeNumber**(**int \_num**){**

bool \_flag **=** true**;**

int i **=** 2**;**

//only need to scan half of the number

//because if it is no divible by any number

//lest than or equal its half, its prime

**for(**i **=** 2**;** i **<=** \_num **/** 2**;** i**++)** **{**

**if(**\_num **%** i **==** 0**)** **{**

\_flag **=** false**;**

**break;**

**}**

**}**

**return** \_flag**;**

**}**

int main**(**int \_argc**,** char **\*** \_argv**[],** char **\*** \_envp**[]){**

//if server is not specifed then exit

**if(**\_argc **<** 2**){**

perror**(**"Client: Server Not Specified"**);**

exit**(**1**);**

**}**

CLIENT **\***\_client**;** //client struct provided by rpc

Token **\***\_returnValue **=** **(**Token**\*)**malloc**(sizeof(**Token**));** //token to hold return

char **\***\_server**;** //vaiable to hold server sent as argument to client executable

//get specified server

\_server **=** \_argv**[**1**];**

//generate client handle to call the server and set communication protocol to tcp

**if** **(** **(**\_client **=** clnt\_create**(**\_server**,** PRIME\_COMPUTATION**,**PCOMP\_1**,**"tcp"**))** **==** **(**CLIENT**\*)NULL** **){**

clnt\_pcreateerror**(**\_server**);** //if there is an errnor creating client then display the error then exit

exit**(**2**);**

**}**

//instantiate token

Token \_tkn**;**

//initialize token

\_tkn**.**\_numberOfHops **=** 0**;**

\_tkn**.**\_timeOfCreation **=** 0**;**

\_tkn**.**\_nextInteger **=** 2**;**

\_tkn**.**\_numberOfPrimesFound **=** 0**;**

\_tkn**.**\_highestPrimeFound **=** 0**;**

//create final elapsed time variable

time\_t \_startTime**,** \_endTime**;**

//get time of token creation (before first hop)

time**(&(**\_tkn**.**\_timeOfCreation**));**

//start pime computation

int \_startIndex **=** 0**;**

bool \_initialHop **=** true**;**

time**(&(**\_startTime**));** //start timer

printf**(**"\n\nClient: Started Token Hopping Client-Server System with Limit = %i\n\n"**,**LIMIT**);**

**while(**\_startIndex **<** LIMIT**/**2**){**

//get the next integer to start finding next prime

int \_currentInteger **=** \_tkn**.**\_nextInteger**;**

//find the next prime and update token

bool \_isPrime **=** false**;**

**while(**\_isPrime **==** false**){**

//get wether the given number is prime or not by calling "isPrimeNumber()" function

**if(**isPrimeNumber**(**\_currentInteger**)** **==** true**){**

//set next integer to be one more than the value of the last prime

\_tkn**.**\_nextInteger **=** \_currentInteger **+** 1**;**

//increment the number of prime numbers found by 1

\_tkn**.**\_numberOfPrimesFound **=** \_tkn**.**\_numberOfPrimesFound **+** 1**;**

//update the highest prime

**if(**\_currentInteger **>** \_tkn**.**\_highestPrimeFound**){**

\_tkn**.**\_highestPrimeFound **=** \_currentInteger**;**

**}**

**if(**\_initialHop **==** true**){**

\_initialHop **=** false**;**

**}**

**else{**

//increment hop count

\_tkn**.**\_numberOfHops **=** \_tkn**.**\_numberOfHops **+** 1**;**

**}**

//exit loop

\_isPrime **=** true**;**

**}**

**else{**

\_isPrime **=** false**;**

//increment index for next number to be checked for prime

\_currentInteger **=** \_currentInteger **+** 1**;**

**}**

**}**

//reset loop

\_isPrime **=** false**;**

//printTokenDetailsBefore(\_tkn);

//pass token to server to find next prime

\_returnValue **=** start\_prime\_computation\_1**(&**\_tkn**,**\_client**);**

//copy returned token value from server to local token

\_tkn**.**\_numberOfHops **=**\_returnValue**->**\_numberOfHops**;**

\_tkn**.**\_timeOfCreation **=**\_returnValue**->**\_timeOfCreation**;**

\_tkn**.**\_nextInteger **=**\_returnValue**->**\_nextInteger**;**

\_tkn**.**\_numberOfPrimesFound **=**\_returnValue**->**\_numberOfPrimesFound**;**

\_tkn**.**\_highestPrimeFound **=**\_returnValue**->**\_highestPrimeFound**;**

//printTokenDetailsAfter(\_tkn);

//increase loop index

\_startIndex **=** \_startIndex **+** 1**;**

**}**

//calculate elapsed time

time**(&**\_endTime**);** // stop timer

double \_elapsedTimeSeconds **=** difftime**(**\_endTime**,** \_startTime**);** //get elapsed time

double \_elapsedTimeMilliseconds **=** \_elapsedTimeSeconds **\*** 1000**;** //elpaed time from seconds to milliseconds

//calculate hops per second

double \_hopsPerSecond **=** **(**double**)**\_tkn**.**\_numberOfHops **/** \_elapsedTimeSeconds**;**

//convert token's time of create into a display able format

char \_buff**[**20**];**

time\_t \_tknCreation **=** **(**time\_t**)**\_tkn**.**\_timeOfCreation**;**

strftime**(**\_buff**,** 20**,** "%Y-%m-%d %H:%M:%S"**,** localtime**(&**\_tknCreation**));**

printf**(**"Client: Hops Limit Reached!!!...Now Rest from Hopping\n\n"**);**

printf**(**"\n\n"**);**

printf**(**"----------------------------------------------------\n"**);**

printf**(**"Client: Other Statistics\n"**);**

printf**(**"----------------------------------------------------\n"**);**

printf**(**"LIMIT: %i\n"**,**LIMIT**);** //display the limit used to repeatedly pass token back and forth

printf**(**"Token Time of Creation: %i\n"**,** \_tkn**.**\_timeOfCreation**);** //display token's time of creation in seconds

printf**(**"Token Time of Creation in format: %s\n"**,** \_buff**);**//display token's time of creation in YY/MM/DD- HH/mm/ss format

printf**(**"Hop Start Time : %i seconds\n"**,** \_startTime**);** //display hop start time

printf**(**"Hop End Time : %i seconds\n"**,** \_endTime**);** //display hop end time

printf**(**"The Elapsed Time: %f seconds\n"**,**\_elapsedTimeSeconds**);**//display hop elapsed time

printf**(**"\n\n"**);**

//print summary statistics

printf**(**"----------------------------------------------------\n"**);**

printf**(**"Client: Summary Statistics\n"**);**

printf**(**"----------------------------------------------------\n"**);**

printf**(**"Count of the number of hops: %i\n"**,**\_tkn**.**\_numberOfHops**+**1**);** //display number of hops, since loop starts at index 0, +1 is done to accomodate that

printf**(**"The number of prime numbers that were found starting from 2: %i\n"**,**\_tkn**.**\_numberOfPrimesFound**);** //display number of primes found

printf**(**"Highest Prime Number Found: %i\n"**,**\_tkn**.**\_highestPrimeFound**);** //display highest prime found

printf**(**"The Elapsed Time: %f Milliseconds\n"**,**\_elapsedTimeMilliseconds**);** //display elapsed time in millisconds

printf**(**"The number of hops(messages) per second: %f \n"**,**\_hopsPerSecond**);** //displays hops per seconds

printf**(**"\n\nClient Done, Please Remeber to Kill Server using PS and KILL\n\n"**);**

**return** 0**;**

**}**

**primeToken\_server.c**

/\*\*

\***@since** 3/1/2020

\***@file** primeToken\_server.c

\*This file demonstrates the server part of RPC based client-server communication

\*\*/

#include "primeToken.h"

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

/\*\*

\* This function prints the details of token

\***@param** \_tkn The token structure whose detail is to be printed

\***@return** nothing

\*/

void printTokenDetails**(**Token \_tkn**){**

printf**(**"\n\n Token In Server\n"**);**

printf**(**"Server:-- Hops: %i\n"**,**\_tkn**.**\_numberOfHops**);**

printf**(**"Server:-- Time: %i\n"**,**\_tkn**.**\_timeOfCreation**);**

printf**(**"Server:-- Next Int: %i\n"**,**\_tkn**.**\_nextInteger**);**

printf**(**"Server:-- Num Prime: %i\n"**,**\_tkn**.**\_numberOfPrimesFound**);**

printf**(**"Server:-- Highest: %i\n"**,**\_tkn**.**\_highestPrimeFound**);**

printf**(**"\n\n"**);**

**}**

/\*\*

\* This function checks where a given number si prime or not

\***@param** \_num This parameter is checked if its prime or not

\***@return** true if the param a is prime, false otherwise

\*/

bool isPrimeNumber**(**int \_num**){**

bool \_flag **=** true**;**

int i **=** 2**;**

//only need to scan half of the number

//because if it is no divible by any number

//lest than or equal its half, its prime

**for(**i **=** 2**;** i **<=** \_num **/** 2**;** i**++)** **{**

**if(**\_num **%** i **==** 0**)** **{** //if number is divisible break

\_flag **=** false**;**

**break;**

**}**

**}**

**return** \_flag**;**

**}**

Token **\*** start\_prime\_computation\_1\_svc**(**Token **\*** \_token**,** struct svc\_req **\*** req**){**

//It is impertative that the return identifier referenced

//be of type type static, as opposed to local. Local identifiers

//are allocated on stack, and a reference to their contents would be

//invalid once the function returns -- John shapely gray ch 9 rpc

//create token

static Token \_tkn**;**

//initialize token by oopying the value sent by client to server's local token

\_tkn**.**\_numberOfHops **=** \_token**->**\_numberOfHops**;** //copy

\_tkn**.**\_timeOfCreation **=** \_token**->**\_timeOfCreation**;**

\_tkn**.**\_nextInteger **=** \_token**->**\_nextInteger**;**

\_tkn**.**\_numberOfPrimesFound **=** \_token**->**\_numberOfPrimesFound**;**

\_tkn**.**\_highestPrimeFound **=** \_token**->**\_highestPrimeFound**;**

//get the next integer to start finding next prime

int \_currentInteger **=** \_tkn**.**\_nextInteger**;**

//find the next prime and update token

bool \_isPrime **=** false**;**

**while(**\_isPrime **==** false**){**

//get wether the given number is prime or not by calling "isPrimeNumber()" function

**if(**isPrimeNumber**(**\_currentInteger**)** **==** true**){**

//set next integer to be one more than the value of the last prime

\_tkn**.**\_nextInteger **=** \_currentInteger **+** 1**;**

//increment the number of prime numbers found by 1

\_tkn**.**\_numberOfPrimesFound **=** \_tkn**.**\_numberOfPrimesFound **+** 1**;**

//update the highest prime

**if(**\_currentInteger **>** \_tkn**.**\_highestPrimeFound**){**

\_tkn**.**\_highestPrimeFound **=** \_currentInteger**;**

**}**

//increment hop count

\_tkn**.**\_numberOfHops **=** \_tkn**.**\_numberOfHops **+** 1**;**

//exit loop

\_isPrime **=** true**;**

**}**

**else{**

\_isPrime **=** false**;**

//increment index for next number to be checked for prime

\_currentInteger **=** \_currentInteger **+** 1**;**

**}**

**}**

//printTokenDetails(\_tkn);

**return** **(&**\_tkn**);** //return the local token to client

**}**

**primeToken\_clnt.c**

/\*

\* Please do not edit this file.

\* It was generated using rpcgen.

\*/

#include <memory.h> /\* for memset \*/

#include "primeToken.h"

/\* Default timeout can be changed using clnt\_control() \*/

static struct timeval TIMEOUT **=** **{** 25**,** 0 **};**

Token **\***

start\_prime\_computation\_1**(**Token **\***argp**,** CLIENT **\***clnt**)**

**{**

static Token clnt\_res**;**

memset**((**char **\*)&**clnt\_res**,** 0**,** **sizeof(**clnt\_res**));**

**if** **(**clnt\_call **(**clnt**,** START\_PRIME\_COMPUTATION**,**

**(**xdrproc\_t**)** xdr\_Token**,** **(**caddr\_t**)** argp**,**

**(**xdrproc\_t**)** xdr\_Token**,** **(**caddr\_t**)** **&**clnt\_res**,**

TIMEOUT**)** **!=** RPC\_SUCCESS**)** **{**

**return** **(NULL);**

**}**

**return** **(&**clnt\_res**);**

**}**

**primeToken\_svc.c**

/\*

\* Please do not edit this file.

\* It was generated using rpcgen.

\*/

#include "primeToken.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

prime\_computation\_1**(**struct svc\_req **\***rqstp**,** register SVCXPRT **\***transp**)**

**{**

union **{**

Token start\_prime\_computation\_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** START\_PRIME\_COMPUTATION**:**

\_xdr\_argument **=** **(**xdrproc\_t**)** xdr\_Token**;**

\_xdr\_result **=** **(**xdrproc\_t**)** xdr\_Token**;**

local **=** **(**char **\*(\*)(**char **\*,** struct svc\_req **\*))** start\_prime\_computation\_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 **(**PRIME\_COMPUTATION**,** PCOMP\_1**);**

transp **=** svcudp\_create**(**RPC\_ANYSOCK**);**

**if** **(**transp **==** **NULL)** **{**

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

exit**(**1**);**

**}**

**if** **(!**svc\_register**(**transp**,** PRIME\_COMPUTATION**,** PCOMP\_1**,** prime\_computation\_1**,** IPPROTO\_UDP**))** **{**

fprintf **(**stderr**,** "%s"**,** "unable to register (PRIME\_COMPUTATION, PCOMP\_1, 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**,** PRIME\_COMPUTATION**,** PCOMP\_1**,** prime\_computation\_1**,** IPPROTO\_TCP**))** **{**

fprintf **(**stderr**,** "%s"**,** "unable to register (PRIME\_COMPUTATION, PCOMP\_1, tcp)."**);**

exit**(**1**);**

**}**

svc\_run **();**

fprintf **(**stderr**,** "%s"**,** "svc\_run returned"**);**

exit **(**1**);**

/\* NOTREACHED \*/

**}**

**primeToken\_xdr.c**

/\*

\* Please do not edit this file.

\* It was generated using rpcgen.

\*/

#include "primeToken.h"

bool\_t

xdr\_Tkn **(**XDR **\***xdrs**,** Tkn **\***objp**)**

**{**

register int32\_t **\***buf**;**

**if** **(!**xdr\_int **(**xdrs**,** **&**objp**->**\_numberOfHops**))**

**return** FALSE**;**

**if** **(!**xdr\_quad\_t **(**xdrs**,** **&**objp**->**\_timeOfCreation**))**

**return** FALSE**;**

**if** **(!**xdr\_int **(**xdrs**,** **&**objp**->**\_nextInteger**))**

**return** FALSE**;**

**if** **(!**xdr\_int **(**xdrs**,** **&**objp**->**\_numberOfPrimesFound**))**

**return** FALSE**;**

**if** **(!**xdr\_int **(**xdrs**,** **&**objp**->**\_highestPrimeFound**))**

**return** FALSE**;**

**return** TRUE**;**

**}**

bool\_t

xdr\_Token **(**XDR **\***xdrs**,** Token **\***objp**)**

**{**

register int32\_t **\***buf**;**

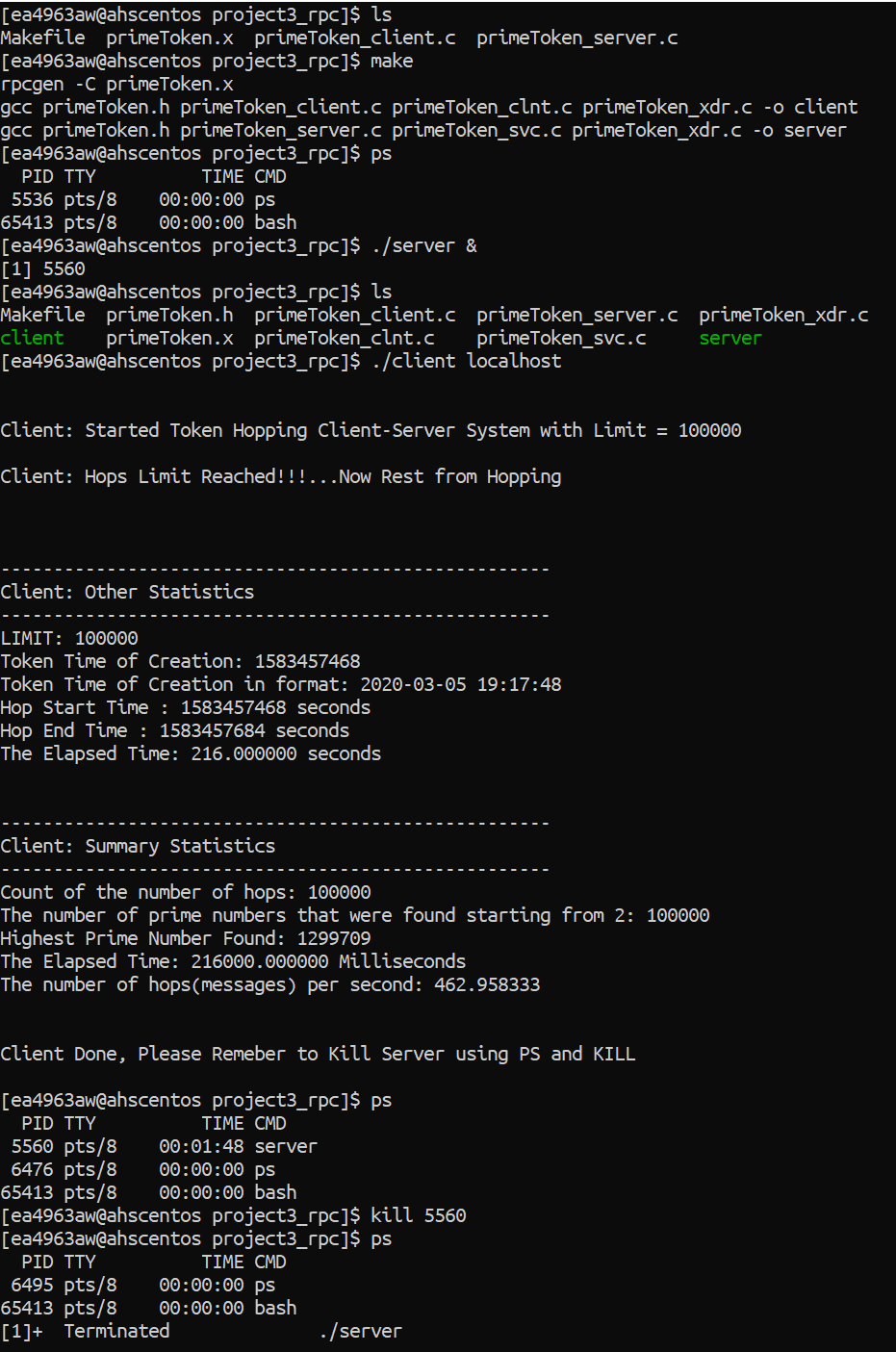
**if** **(!**xdr\_Tkn **(**xdrs**,** objp**))**

**return** FALSE**;**

**return** TRUE**;**

**}**

**OUTPUT LIMIT = 100000**

****