Eric Blasko

Dr. Tong Yu

CSE 461

March 2019

Lab 5 – Distributed Computing

\*\*Note – This lab was completed independently. Also, as a reminder, for labs attendance, I am in the room across the hall as I am the TA for cse460. This was approved by Tong Yu. Each section has code or screenshots, followed by a discussion. Extra Credit in second part completed

**Part 1 Remote Procedure Call (random number – single host)**

$**rpcgen -C -a rand.x**

In this command, what are the purposes of the switches **-C** and **-a**?

Answer: -C generates code in ANSI C, this option also generates code that could be compiled with the c++ compiler. -a generates all the files including sample code for the client and server side.

**Rand.x**

program RAND\_PROG {

version RAND\_VERS {

void INITIALIZE\_RANDOM (long) = 1;

double GET\_NEXT\_RANDOM (void) = 2;

} = 1;

} = 005670557;

**Rand\_server.c**

#include "rand.h"

#include <time.h>

void \*

initialize\_random\_1\_svc(long \*argp, struct svc\_req \*rqstp)

{

static char \* result;

return (void \*) &result;

}

double \*

get\_next\_random\_1\_svc(void \*argp, struct svc\_req \*rqstp)

{

static double result;

double range = (1000 - 0);

double div = RAND\_MAX / range;

srand(time(NULL));

result += (rand() / div);

printf("%f \n",result);

return &result;

}

**Rand\_client.c**

#include "rand.h"

double

rand\_prog\_1(char \*host)

{

CLIENT \*clnt;

void \*result\_1;

long initialize\_random\_1\_arg;

double \*result\_2;

char \*get\_next\_random\_1\_arg;

#ifndef DEBUG

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

if (clnt == NULL) {

clnt\_pcreateerror (host);

exit (1);

}

#endif /\* DEBUG \*/

result\_1 = initialize\_random\_1(&initialize\_random\_1\_arg, clnt);

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

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

}

result\_2 = get\_next\_random\_1((void\*)&get\_next\_random\_1\_arg, clnt);

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

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

}

#ifndef DEBUG

clnt\_destroy (clnt);

#endif /\* DEBUG \*/

return \*result\_2;

}

int

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

{

char \*host;

if (argc < 2) {

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

exit (1);

}

host = argv[1];

rand\_prog\_1 (host);

double x;

int i;

printf("\n 20 Random Numbers\n");

for(i=0; i <20; i++)

{

x = rand\_prog\_1(host);

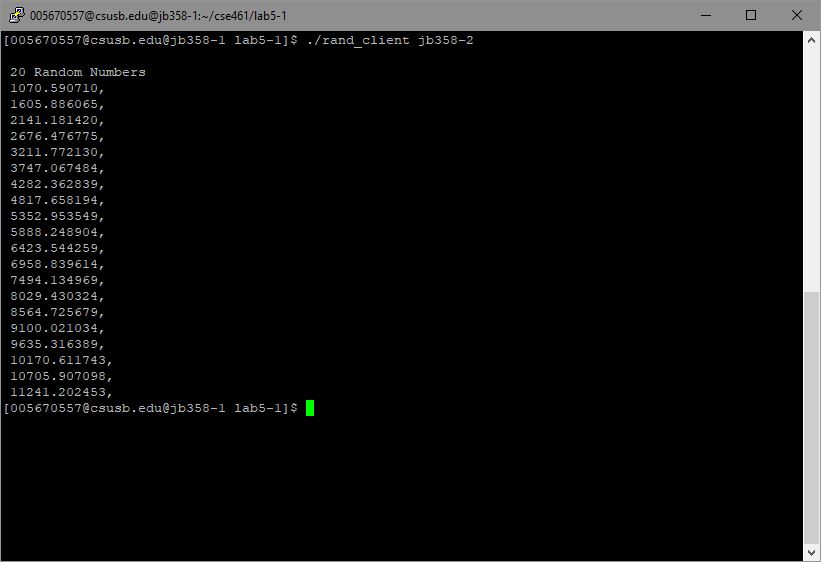
printf(" %f, \n", x);

}

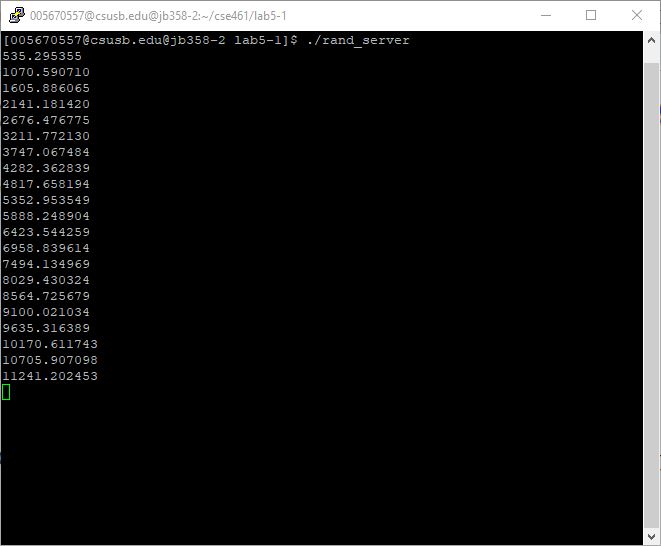
exit (0);

}

Client on jb358-1



Server on jb358-2



**Part 2 (Parallel Random Number Gen – with extra credit)**

**Rand.x**

struct params

{

int xleft;

int xright;

};

program RAND\_PROG {

version RAND\_VERS {

int GET\_NEXT\_RANDOM (params) = 1;

} = 1;

} = 005670557;

**Rand\_server.c**

#include "rand.h"

int \*

get\_next\_random\_1\_svc(params \*argp, struct svc\_req \*rqstp)

{

static int result;

int xl, xr;

xl = argp->xleft;

xr = argp->xright;

result = (11\*xl+13\*result+4\*xr) %31;

printf("%d\n",result);

return &result;

}

**Rand\_client.c**

#include <SDL/SDL.h>

#include <SDL/SDL\_thread.h>

#include "rand.h"

#define N 4

char \*hosts[N];

SDL\_mutex \*mutex;

SDL\_cond \*barrierQueue;

int count = 0;

int era = 0;

int x[N];

int rns[N][10];

int

rand\_prog\_1(char \*host, int xl, int xr)

{

CLIENT \*clnt;

int \*result\_1;

params get\_next\_random\_1\_arg;

get\_next\_random\_1\_arg.xleft = xl;

get\_next\_random\_1\_arg.xright = xr;

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

if (clnt == NULL) {

clnt\_pcreateerror (host);

exit (1);

}

result\_1 = get\_next\_random\_1(&get\_next\_random\_1\_arg, clnt);

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

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

}

clnt\_destroy (clnt);

return \*result\_1;

}

void barrier()

{

int myEra;

SDL\_LockMutex(mutex);

count++;

if(count < N) {

myEra = era;

while(myEra == era)

SDL\_CondWait(barrierQueue, mutex);

} else {

count = 0;

era++;

SDL\_CondBroadcast(barrierQueue);

}

SDL\_UnlockMutex(mutex);

}

int threads (void \*data)

{

int i, i\_minus\_1, i\_plus\_1, id, xleft, xright;

id = \*((int\*) data);

printf("Thread %d",id);

for(i=0; i < 10; i++){

i\_minus\_1 = id -1;

if(i\_minus\_1 < 0)

i\_minus\_1 += N;

xleft = x[i\_minus\_1];

i\_plus\_1 = (id+1) % N;

xright = x[i\_plus\_1];

x[id] = rand\_prog\_1 (hosts[id], xleft, xright);

printf("(%d: %d)",id,x[id]);

rns[id][i] = x[id];

barrier();

}

return 0;

}

int

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

{

int i, j;

SDL\_Thread \*ids[N];

if (argc < 5) {

printf ("usage: %s server\_host1 host2 host3 host4..\n", argv[0]);

exit (1);

}

mutex = SDL\_CreateMutex();

barrierQueue = SDL\_CreateCond();

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

x[i] = rand() % 31;

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

hosts[i] = argv[i+1];

ids[i] = SDL\_CreateThread (threads, &i);

}

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

SDL\_WaitThread (ids[i], NULL);

printf("\nRandom Numbers:");

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

{

printf("\nFrom Server %d:\n",i);

for(j=0;j <10;++j)

printf("%d, ", rns[i][j]);

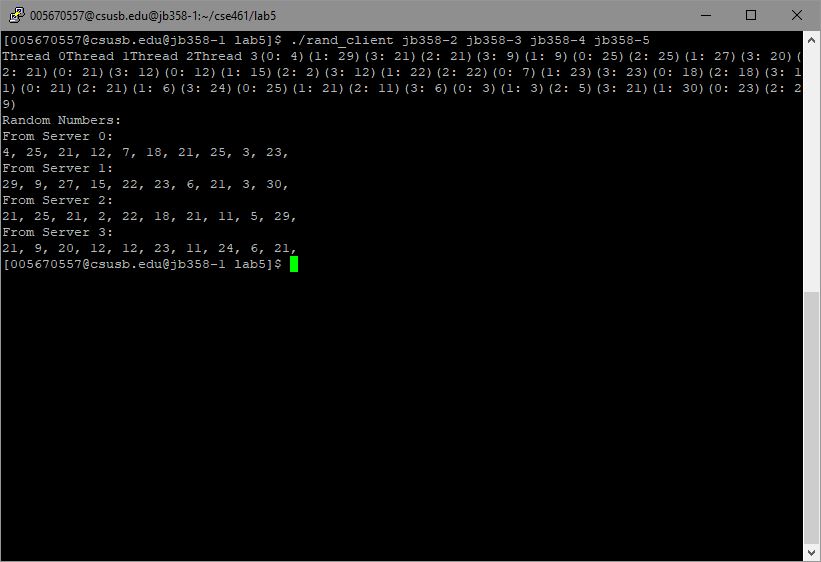
}

printf("\n");

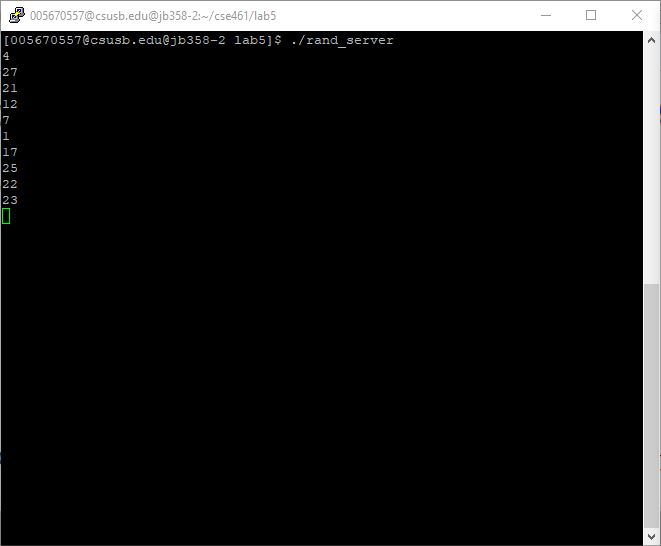
exit (0);

}

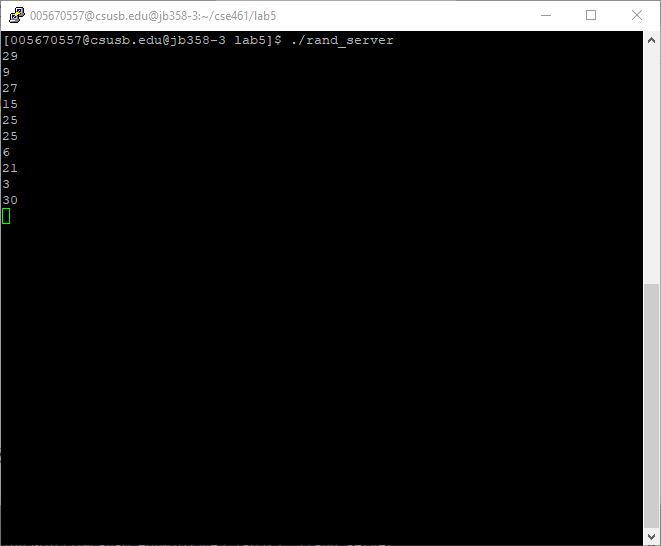
Client on jb358-1



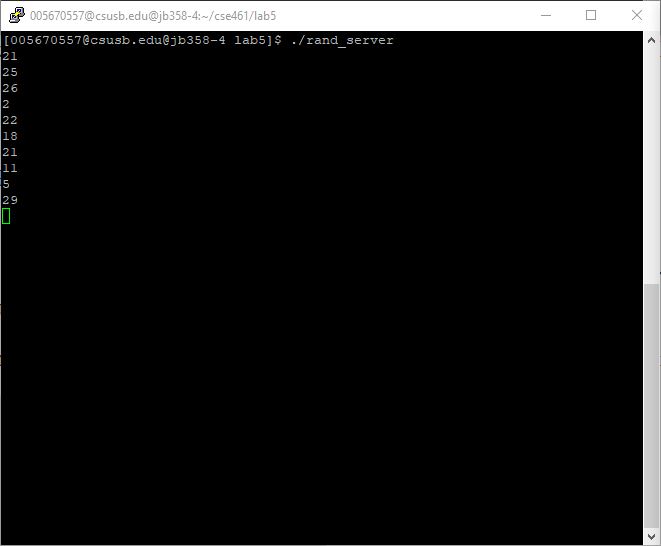
Server 1 on jb358-2



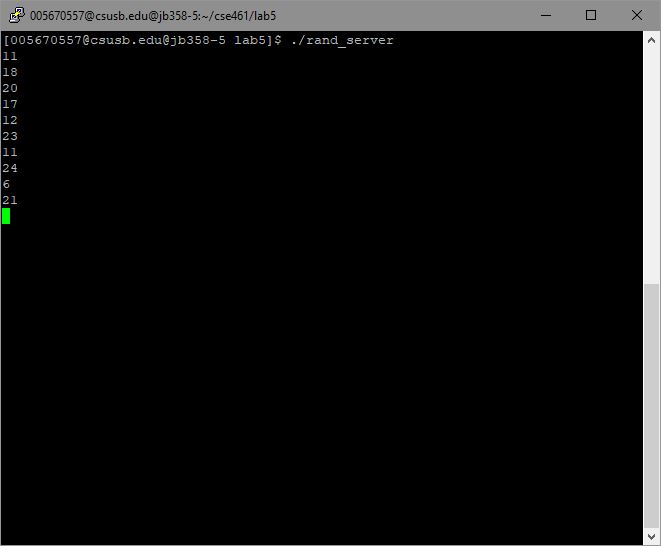
Server 2 on jb358-3



Server 3 on jb358-4



Server 4 on jb358-5



**Discussion/self-evaluation (30/20) extra credit completed**

I was able to complete all required sections of this lab. For the first part I was able to alter the client/server code so that the server generated random numbers every time. The range, however, was keeps intestinally between 0 and 1000. This can easily be altered to generate random numbers between any range. I also edited the server side to print out the results of each number that was generated.

For the second part, I was also able to complete successfully. It did take a while to get to work as I was using jb359 computers which would not let the RPC connect. Once I switched over to jb358 computers, it worked perfectly. As for the extra credit section of this lab, I was able to successfully alter the code so that there could be 4 servers who would return values to the client. Overall, this was a great lab for learning how to use RPC messaging and how it works between the client and server side.