George Suarez

David Cruz

CSE 461

Lab 5 – Distributed Computing

1. What is XDR and what is it for?

- XDR is a standard data serialization format for computer network protocols. It allows data to be transferred between different kinds of computer systems.

1. How do you compile an input file into XDR routines?

- By adding the -c flag when compiling.

1. What are the purposes of the switches -C and -a?

- The -C flag generates code in ANSI C and also generates code that could be compiled with the C++ compiler.

- The -a flag generates all the files including sample code for client and server side.

Part 1)

**rand\_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 "rand.h"

void \*

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

{

static char \* result;

/\*

\* insert server code here

\*/

return (void \*) &result;

}

double \*

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

{

static double result;

result += 0.31;

if ( result >= 1.0 )

result -= 0.713;

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

return &result;

}

**rand\_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 "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;

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

if (clnt == NULL) {

clnt\_pcreateerror (host);

exit (1);

}

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");

}

clnt\_destroy (clnt);

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];

double x;

int i;

printf("\n Twenty random numbers ");

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

{

x = rand\_prog\_1 (host);

printf(" %f, ", x );

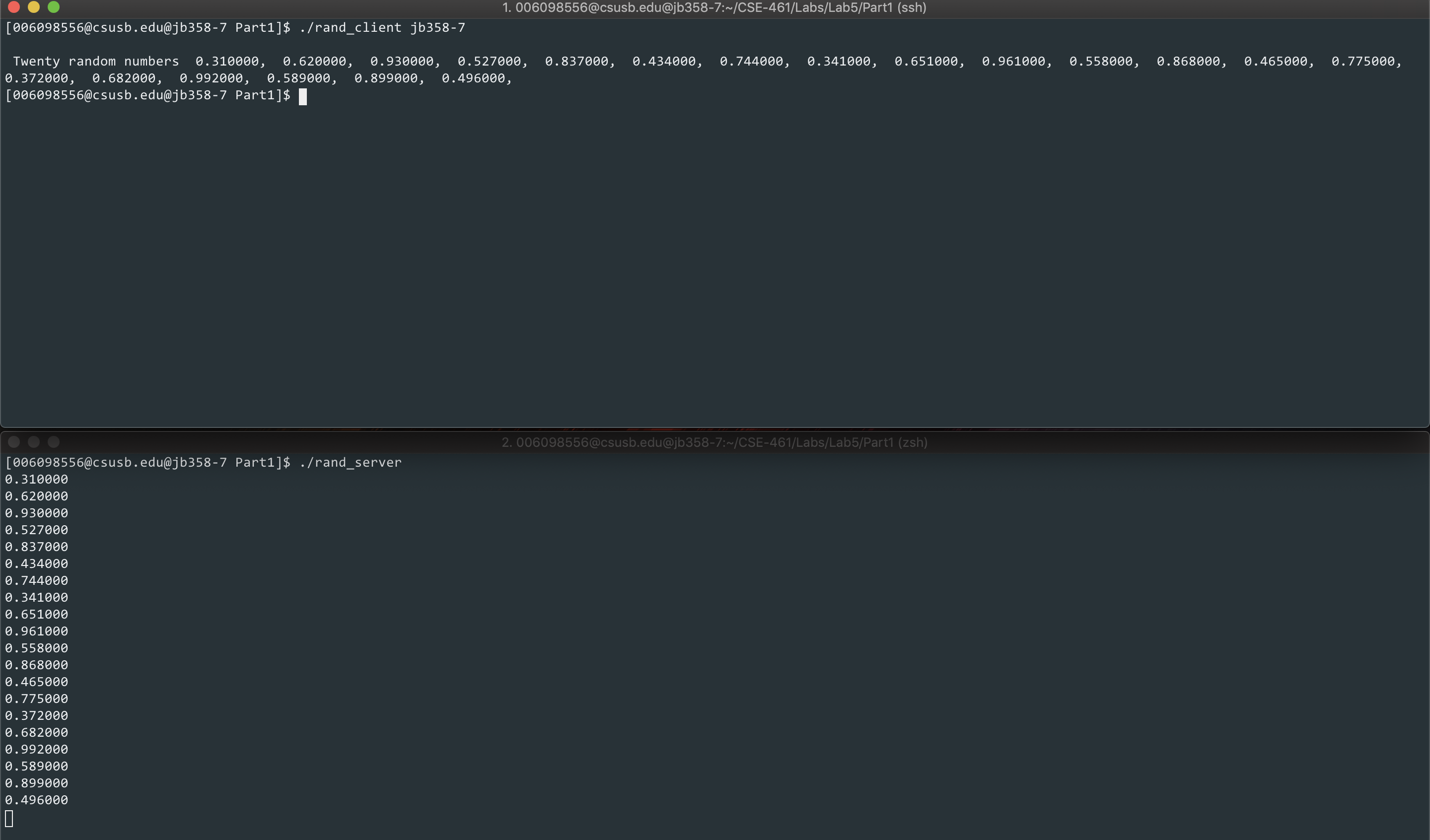
}

printf("\n");

exit (0);

}

**Output**



Part 2) Using the equation:

**rand\_sever.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 "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 + 5 \* xr ) % 31;

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

return &result;

}

**rand\_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 <SDL/SDL.h>

#include <SDL/SDL\_thread.h>

#include "rand.h"

#define N 3

char \*hosts[N];

SDL\_mutex \*mutex;

SDL\_cond \*barrierQueue;

int count = 0, 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 k, i\_minus\_1, i\_plus\_1, id, xleft, xright;

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

printf("Thread: %d ", id );

printf("\n");

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

{

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][k] = x[id];

barrier();

}

}

int

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

{

int i, j;

SDL\_Thread \*ids[N];

if (argc < 4) {

printf ("usage: %s server\_host1 host2 host3 ...\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 );

}

// Print out results in buffers

printf("\n\nRandom Numbers: ");

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

{

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

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

{

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

}

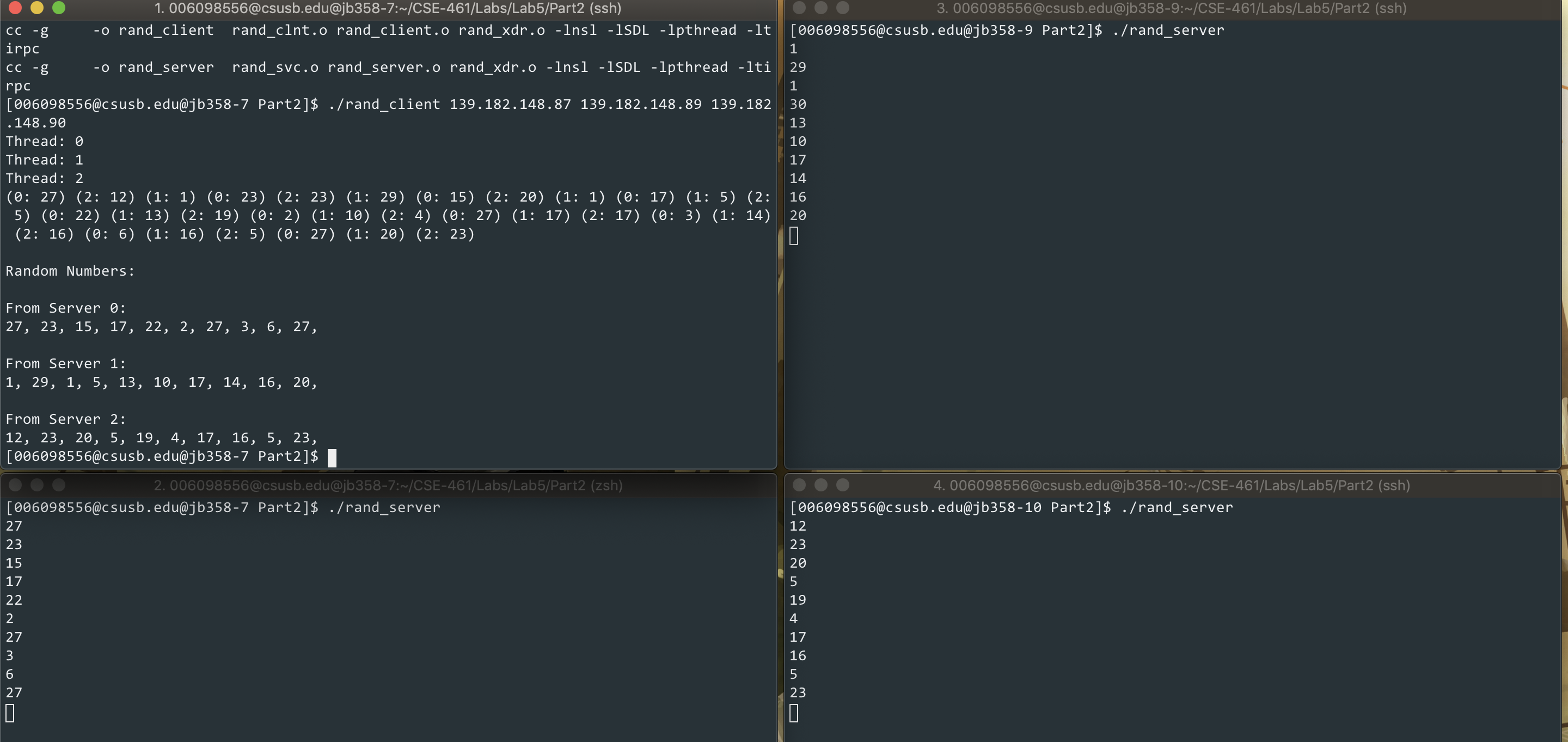
}

printf("\n");

exit (0);

}

**Output**

**

**Summary:**

We have successfully completed all the required work in this lab. We managed to run the servers successfully on 3 different machines in the computer lab. The only problem we had was compiling the program using the Makefile, but we fixed it by adding a missing flag in the Makefile. We give ourselves **20/20**.