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.

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

   - By adding the -c flag when compiling.

3. 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;
}
```

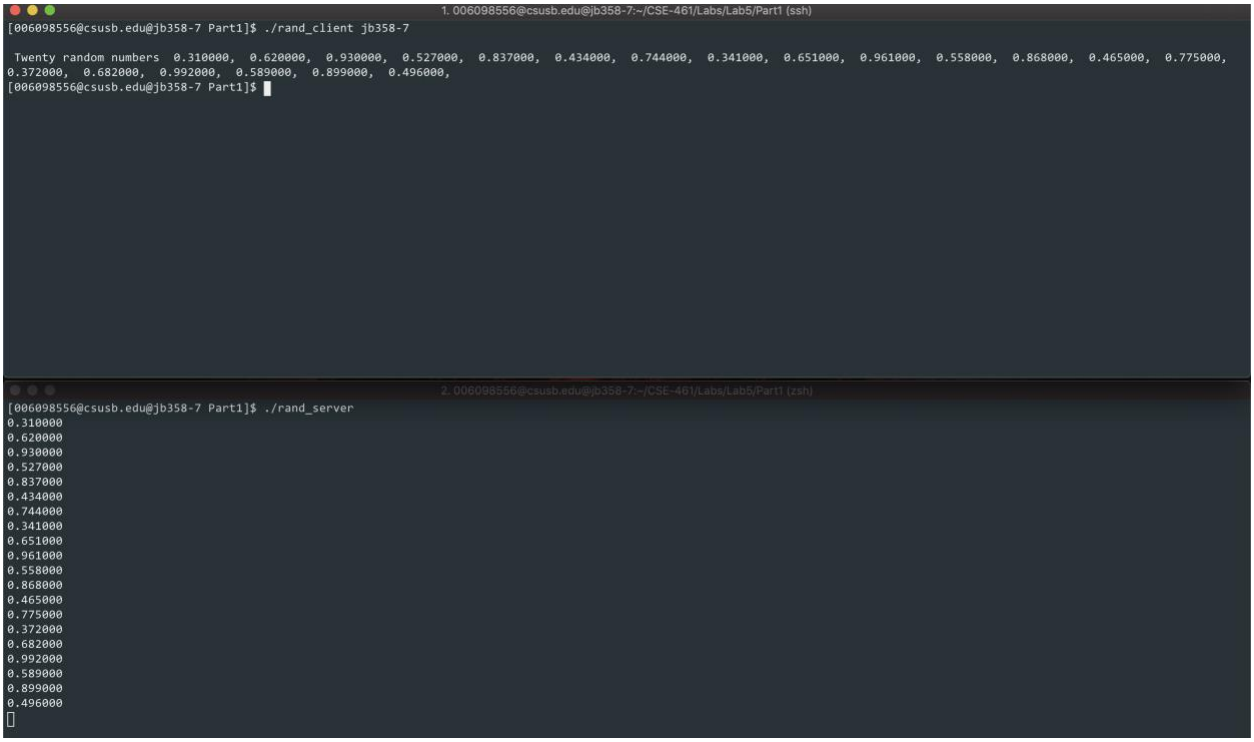
```
double *
get_next_random_1_svc(void *argp, struct svc_req *rqstp)
{
      static double  result;

      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: $x_i(t+1) = 13x_{(i-1)}(t) + 11x_i(t) + 5x_{(i+1)}(t) \; mod \; 31$

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

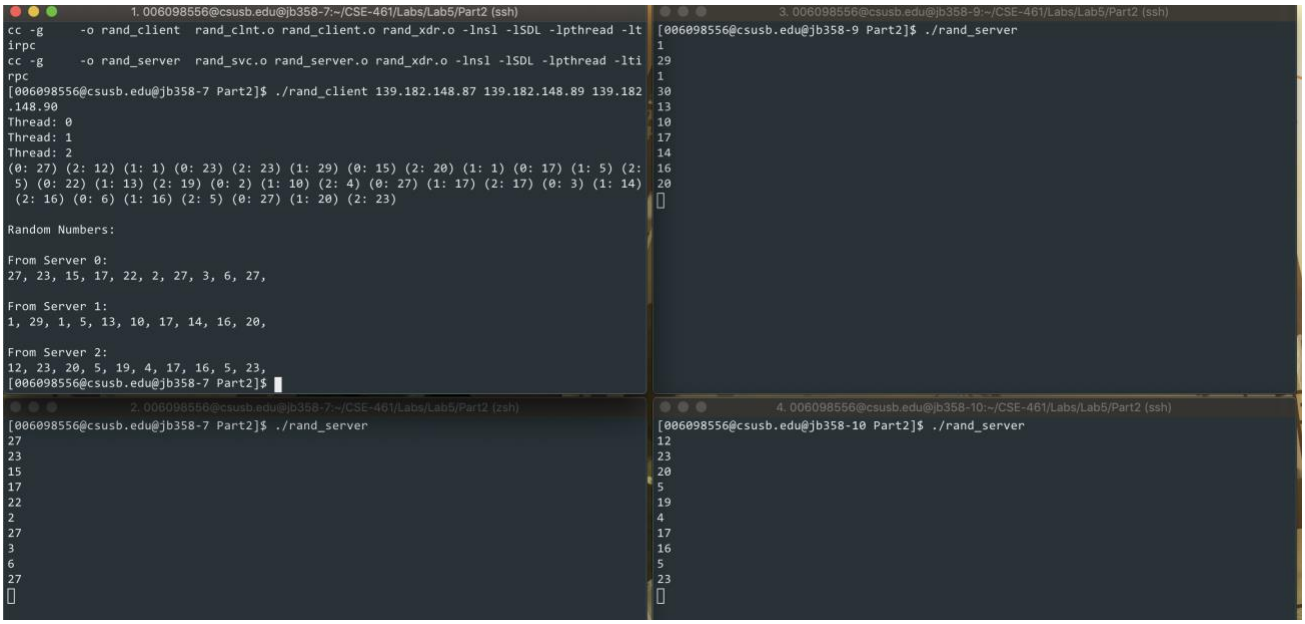
```
/*
 * This is sample code generated by rpcgen.
 * These are only templates and you can use them
 * as a guideline for developing your own functions.
 */

#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**.