Yousef Jarrar, Jose Perez

Professor Tong Yu

CSE 461

March 12, 2019

Points: 30/20

Lab 5: RPC

Part 1: Twenty Random Numbers using rpcgen

Running the command man rpcgen, we are given the documentation:

$man rpcgen

NAME

rpcgen - an RPC protocol compiler

SYNOPSIS

rpcgen infile

rpcgen [-Dname[=value]] [-T] [-K secs] infile

rpcgen -c|-h|-l|-m|-M|-t [-o outfile] infile

rpcgen [-I] -s nettype [-o outfile] infile

rpcgen -n netid [-o outfile] infile

DESCRIPTION

rpcgen is a tool that generates C code to implement an RPC protocol. The input to rpcgen is a language similar to C known as RPC Language (Remote Procedure Call Language).

rpcgen is normally used as in the first synopsis where it takes an input file and generates up to four output files. If the infile is named proto.x, then rpcgen will generate a header file in proto.h, XDR routines in

proto_xdr.c, server-side stubs in proto_svc.c, and client-side stubs in proto_clnt.c. With the -T option, it will also generate the RPC dispatch table in proto_tbl.i. With the -Sc option, it will also generate sample code

which would illustrate how to use the remote procedures on the client side. This code would be created in proto_client.c. With the -Ss option, it will also generate a sample server code which would illustrate how to write

the remote procedures. This code would be created in proto_server.c.

The server created can be started both by the port monitors (for example, inetd or listen) or by itself. When it is started by a port monitor, it creates servers only for the transport for which the file descriptor 0 was

passed. The name of the transport must be specified by setting up the environmental variable PM_TRANSPORT. When the server generated by rpcgen is executed, it creates server handles for all the transports specified in NET-

PATH environment variable, or if it is unset, it creates server handles for all the visible transports from /etc/netconfig file. Note: the transports are chosen at run time and not at compile time.

When built for a port monitor (rpcgen -I), and that the server is self-started, it backgrounds itself by default. A special define symbol RPC_SVC_FG can be used to run the server process in foreground.

The second synopsis provides special features which allow for the creation of more sophisticated RPC servers. These features include support for user provided #defines and RPC dispatch tables. The entries in the RPC dispatch table contain:

- pointers to the service routine corresponding to that procedure,
- a pointer to the input and output arguments
- the size of these routines

A server can use the dispatch table to check authorization and then to execute the service routine; a client library may use it to deal with the details of storage management and XDR data conversion.

The other three synopses shown above are used when one does not want to generate all the output files, but only a particular one. Some examples of their usage is described in the EXAMPLE section below. When rpcgen is executed with the -s option, it creates servers for that particular class of transports. When executed with the -n option, it creates a server for the transport specified by netid. If infile is not specified, rpcgen accepts the standard input.

The C preprocessor, cc -E [see cc(1)], is run on the input file before it is actually interpreted by rpcgen. For each type of output file, rpcgen defines a special preprocessor symbol for use by the rpcgen programmer:

RPC_HDR   defined when compiling into header files

RPC_XDR   defined when compiling into XDR routines

RPC_SVC   defined when compiling into server-side stubs

RPC_CLNT   defined when compiling into client-side stubs

RPC_TBL   defined when compiling into RPC dispatch tables

Any line beginning with `%' is passed directly into the output file, uninterpreted by rpcgen.

For every data type referred to in infile, rpcgen assumes that there exists a routine with the string xdr_ prepended to the name of the data type. If this routine does not exist in the RPC/XDR library, it must be provided.

Providing an undefined data type allows customization of XDR routines.

- With this newly installed package, we create the rand.x that was requested

rand.x

/* rand.x */

```
program RAND_PROG {
    version RAND_VERS {
        void INITIALIZE_RANDOM ( long ) = 1;
        double GET_NEXT_RANDOM ( void ) = 2;

    } = 1;
} = 0x30000000;
```

- Use $rpcgen -C -a rand.x to create the rest of the files.
- Use $make -f Makefile.rand to compile all the files to run the ./rand_server and ./rand_client with the host name added. It should not output anything as the rand_server.c and rand_client.c files do not have added in the code needed.

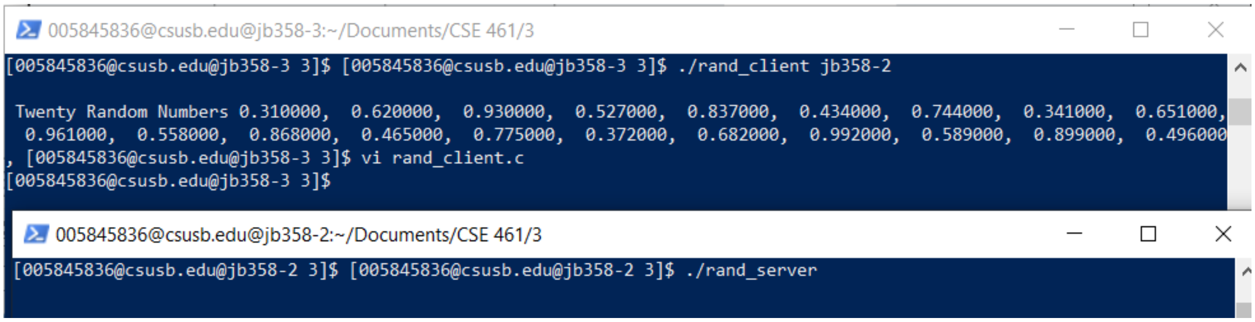
rand_server.c (with added code)

```
#include "rand.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;
    result +=0.31;
    if(result >= 1.0)
        result -= 0.713;
    return &result;
}
```

rand_client.c (with added code)

```
/*
 * 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;

#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 Twenty Random Numbers");
    for ( i = 0; i<20; ++i){
        x = rand_prog_1(host);
        printf(" %f\n, ", x);
    }
exit (0);
}
```

- $make -f Makefile.rand again and run both the server and client to get the 20 random numbers.



005845836@csusb.edu@jb358-3:~/Documents/CSE 461/3

```
[005845836@csusb.edu@jb358-3 3]$ [005845836@csusb.edu@jb358-3 3]$ ./rand_client jb358-2

 Twenty Random Numbers 0.310000,  0.620000,  0.930000,  0.527000,  0.837000,  0.434000,  0.744000,  0.341000,  0.651000,
  0.961000,  0.558000,  0.868000,  0.465000,  0.775000,  0.372000,  0.682000,  0.992000,  0.589000,  0.899000,  0.496000
, [005845836@csusb.edu@jb358-3 3]$ vi rand_client.c
[005845836@csusb.edu@jb358-3 3]$
```

005845836@csusb.edu@jb358-2:~/Documents/CSE 461/3

```
[005845836@csusb.edu@jb358-2 3]$ [005845836@csusb.edu@jb358-2 3]$ ./rand_server
```

Part 2: Parallel Random Number Generator

rand.x (for part 2)

```
/*rand.x*/
struct params{
    int xleft;
    int xright;
};

program RAND_PROG{
    version RAND_VERS {
     int GET_NEXT_RANDOM ( params ) = 1;   /*Service #1*/
    } = 1;
} = 0x30000000;    /* program # */
```

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"

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;

    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 //Number of hosts/threads
```

```
char *hosts[N]; //servers
SDL_mutex *mutex;
SDL_cond *barrierQueue; //Condition Variable

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;

//#ifndef   DEBUG
    clnt = clnt_create (host, RAND_PROG, RAND_VERS, "udp");
    if (clnt == NULL) {
        clnt_pcreateerror (host);
        exit (1);
    }
//#endif DEBUG

    result_1 = get_next_random_1(&get_next_random_1_arg, clnt);
    if (result_1 == (int *) NULL) {
        clnt_perror (clnt, "call failed");
    }
//#ifndef   DEBUG
    clnt_destroy (clnt);
//#endif  DEBUG
    return *result_1;
}

void barrier(){
    int myEra; //a local variable
    SDL_LockMutex ( mutex );

    count ++;
    if ( count < N ){
        myEra = era;
        while ( myEra == era )
            SDL_CondWait ( barrierQueue, mutex );
    } else {
        count = 0; //reset the count
        era ++;
        SDL_CondBroadcast ( barrierQueue ); // Signal all threads in queue
      }
    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);

    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();
    }
    return 0;
}


int
main (int argc, char *argv[])
{
    int i, j;
    SDL_Thread *ids[N];

    //char *host;

    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; //Initial Values

    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 buffer
    printf("\n Random Numbers: ");
    for ( i = 0; i < N; i++){
        printf("\n From Server %d:\n", i);
        for (j = 0; j < 10; ++j )
            printf("%d, ", rns[i][j] );
    }
    printf("\n");

    //host = argv[1];
    //rand_prog_1 (host);
```

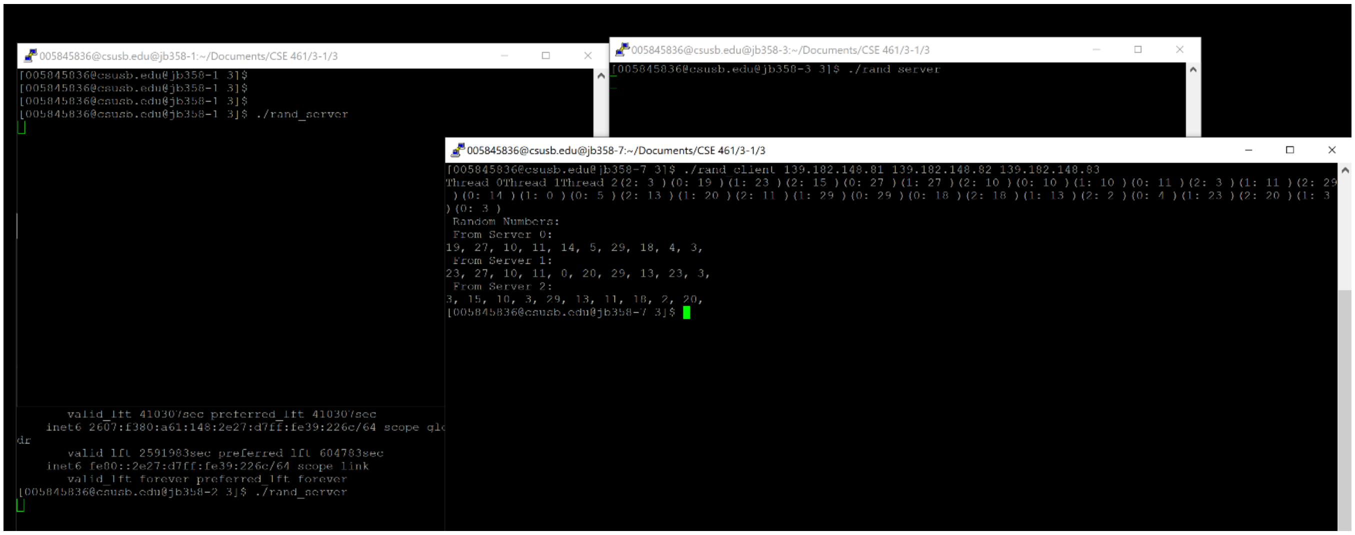
```
exit (0);
}
```

Makefile.rand

```
# This is a template Makefile generated by rpcgen
# Parameters
CLIENT = rand_client
SERVER = rand_server

SOURCES_CLNT.c =
SOURCES_CLNT.h =
SOURCES_SVC.c =
SOURCES_SVC.h =
SOURCES.x = rand.x

TARGETS_SVC.c = rand_svc.c rand_server.c rand_xdr.c
TARGETS_CLNT.c = rand_clnt.c rand_client.c rand_xdr.c
TARGETS = rand.h rand_xdr.c rand_clnt.c rand_svc.c rand_client.c rand_server.c

OBJECTS_CLNT = $(SOURCES_CLNT.c:%.c=%.o) $(TARGETS_CLNT.c:%.c=%.o)
OBJECTS_SVC = $(SOURCES_SVC.c:%.c=%.o) $(TARGETS_SVC.c:%.c=%.o)
# Compiler flags

CFLAGS += -g
LDLIBS += -lnsl -lSDL -lpthread -ltirpc
RPCGENFLAGS =

# Targets

all : $(CLIENT) $(SERVER)

$(TARGETS) : $(SOURCES.x)
	rpcgen $(RPCGENFLAGS) $(SOURCES.x)

$(OBJECTS_CLNT) : $(SOURCES_CLNT.c) $(SOURCES_CLNT.h) $(TARGETS_CLNT.c)
```

```
$(OBJECTS_SVC) : $(SOURCES_SVC.c) $(SOURCES_SVC.h) $(TARGETS_SVC.c)


$(CLIENT) : $(OBJECTS_CLNT)
                $(LINK.c) -o $(CLIENT) $(OBJECTS_CLNT) $(LDLIBS)


$(SERVER) : $(OBJECTS_SVC)
                $(LINK.c) -o $(SERVER) $(OBJECTS_SVC) $(LDLIBS)


 clean:
                 $(RM) core $(TARGETS) $(OBJECTS_CLNT) $(OBJECTS_SVC) $(CLIENT) $(SERVER)
```

Output:


005845836@csusb.edu@jb358-1:~/Documents/CSE 461/3-1/3
005845836@csusb.edu@jb358-3:~/Documents/CSE 461/3-1/3
005845836@csusb.edu@jb358-7:~/Documents/CSE 461/3-1/3

```
[005845836@csusb.edu@jb358-1 3]$
[005845836@csusb.edu@jb358-1 3]$
[005845836@csusb.edu@jb358-1 3]$
[005845836@csusb.edu@jb358-1 3]$ ./rand_server
```

```
[005845836@csusb.edu@jb358-3 3]$ ./rand server
```

```
[005845836@csusb.edu@jb358-7 3]$ ./rand_client 139.182.148.81 139.182.148.82 139.182.148.83
Thread 0Thread 1Thread 2(2: 3 )(0: 19 )(1: 23 )(2: 15 )(0: 27 )(1: 27 )(2: 10 )(0: 10 )(1: 10 )(0: 11 )(2: 3 )(1: 11 )(2: 29
 )(0: 14 )(1: 0 )(0: 5 )(2: 13 )(1: 20 )(2: 11 )(1: 29 )(0: 29 )(0: 18 )(2: 18 )(1: 13 )(2: 2 )(0: 4 )(1: 23 )(2: 20 )(1: 3
)(0: 3 )
 Random Numbers:
 From Server 0:
19, 27, 10, 11, 14, 5, 29, 18, 4, 3,
 From Server 1:
23, 27, 10, 11, 0, 20, 29, 13, 23, 3,
 From Server 2:
3, 15, 10, 3, 29, 13, 11, 18, 2, 20,
[005845836@csusb.edu@jb358-7 3]$
```

```
        valid_lft 410307sec preferred_lft 410307sec
     inet6 2607:f380:a61:148:2e27:d7ff:fe39:226c/64 scope glo
dr
        valid_lft 2591983sec preferred_lft 604783sec
     inet6 fe80::2e27:d7ff:fe39:226c/64 scope link
        valid_lft forever preferred_lft forever
[005845836@csusb.edu@jb358-2 3]$ ./rand_server
```

Extra Credit:

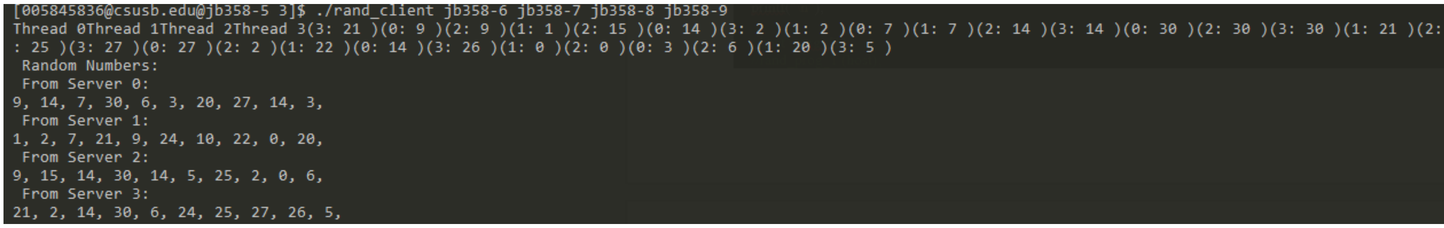
1. (Extra credit 10 points) Extend the random number generator discussed above to one with n = 4. Run the programs in 5 different machines.

```
/*
 * 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 4 //Number of hosts/threads
    char *hosts[N]; //servers
    SDL_mutex *mutex;
    SDL_cond *barrierQueue; //Condition Variable

    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;

    //#ifndef  DEBUG
        clnt = clnt_create (host, RAND_PROG, RAND_VERS, "udp");
        if (clnt == NULL) {
            clnt_pcreateerror (host);
            exit (1);
        }
    //#endif DEBUG

        result_1 = get_next_random_1(&get_next_random_1_arg, clnt);
        if (result_1 == (int *) NULL) {
            clnt_perror (clnt, "call failed");
        }
    //#ifndef  DEBUG
```

```
     clnt_destroy (clnt);
//#endif  DEBUG
     return *result_1;
}

void barrier(){
     int myEra; //a local variable
     SDL_LockMutex ( mutex );

     count ++;
     if ( count < N ){
           myEra = era;
           while ( myEra == era )
                SDL_CondWait ( barrierQueue, mutex );
     } else {
           count = 0; //reset the count
           era ++;
           SDL_CondBroadcast ( barrierQueue ); // Signal all threads in queue
        }
     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);

     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();
     }
     return 0;
}


int
main (int argc, char *argv[])
{
     int i, j;
     SDL_Thread *ids[N];

     //char *host;

```

```
        if (argc < 4) {
            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; //Initial Values

        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 buffer
        printf("\n Random Numbers: ");
        for ( i = 0; i < N; i++){
            printf("\n From Server %d:\n", i);
            for (j = 0; j < 10; ++j )
                printf("%d, ", rns[i][j] );
        }
        printf("\n");

        //host = argv[1];
        //rand_prog_1 (host);
    exit (0);
    }
```

```
[005845836@csusb.edu@jb358-5 3]$ ./rand_client jb358-6 jb358-7 jb358-8
jb358-9
Thread 0Thread 1Thread 2Thread 3(3: 21 )(0: 9 )(2: 9 )(1: 1 )(2: 15
)(0: 14 )(3: 2 )(1: 2 )(0: 7 )(1: 7 )(2: 14 )(3: 14 )(0: 30 )(2: 30
)(3: 30 )(1: 21 )(2: 14 )(3: 6 )(0: 6 )(1: 9 )(0: 3 )(2: 5 )(1: 24
)(3: 24 )(0: 20 )(1: 10 )(3: 25 )(2: 25 )(3: 27 )(0: 27 )(2: 2 )(1: 22
)(0: 14 )(3: 26 )(1: 0 )(2: 0 )(0: 3 )(2: 6 )(1: 20 )(3: 5 )
 Random Numbers:
 From Server 0:
9, 14, 7, 30, 6, 3, 20, 27, 14, 3,
 From Server 1:
1, 2, 7, 21, 9, 24, 10, 22, 0, 20,
 From Server 2:
9, 15, 14, 30, 14, 5, 25, 2, 0, 6,
 From Server 3:
21, 2, 14, 30, 6, 24, 25, 27, 26, 5,
```



```
[005845836@csusb.edu@jb358-5 3]$ ./rand_client jb358-6 jb358-7 jb358-8 jb358-9
Thread 0Thread 1Thread 2Thread 3(3: 21 )(0: 9 )(2: 9 )(1: 1 )(2: 15 )(0: 14 )(3: 2 )(1: 2 )(0: 7 )(1: 7 )(2: 14 )(3: 14 )(0: 30 )(2: 30 )(3: 30 )(1: 21 )(2:
: 25 )(3: 27 )(0: 27 )(2: 2 )(1: 22 )(0: 14 )(3: 26 )(1: 0 )(2: 0 )(0: 3 )(2: 6 )(1: 20 )(3: 5 )
 Random Numbers:
 From Server 0:
9, 14, 7, 30, 6, 3, 20, 27, 14, 3,
 From Server 1:
1, 2, 7, 21, 9, 24, 10, 22, 0, 20,
 From Server 2:
9, 15, 14, 30, 14, 5, 25, 2, 0, 6,
 From Server 3:
21, 2, 14, 30, 6, 24, 25, 27, 26, 5,
```

Reflection:

Jose and I have completed all sections of Lab 5. We managed to setup the random twenty number generator and then extended that information to creating more than one instance of the server on the JBH358 computers. The hardest part of the lab was being able to understand how the server and the client communicated with each other. Although we could’ve read the Manual of RPC-GEN. It was easier to watch the video the Dr.Yu posted and follow along with him. By completing this, we successfully implemented the second portion of the lab, and this include the extra credit that was shown to the Teacher Aid. Who marked us off.

In total we think we deserve 30/20 (This includes the extra 10 points, for Extra Credit).