Christian Mesina
Linxuan Liu
CSE 461

Lab 5
Distributed Computing

1. Remote Procedure Call

Questions:

What is XDR and what is it for? How do you compile an input file into XDR routines?

- Extended detection and response (XDR) delivers visibility into data across networks, clouds, endpoints, and applications while applying analytics and automation to detect, analyze, hunt, and remediate today's and tomorrow's threats. Using “rpcgen -c [infile]” to compile an input file into XDR routines.

What are the purposes of the switches **-C** and **-a?**

- C is to generate ANSI C code and also generates code that could be compiled with the C++ compiler. -a is to generate all the files including sample code for client and server side.

Source Code after following the instructions/demo:

rand.x

```
/* rand.x */

program RAND_PROG {
 version RAND_VERS {
   void INITIALIZE_RANDOM ( long ) = 1;         /* service #1 */
   double GET_NEXT_RANDOM ( void ) = 2;         /* service #2 */
 } = 1;
} = 6151141;          /* program # */
```

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.
*/
```

```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;
```

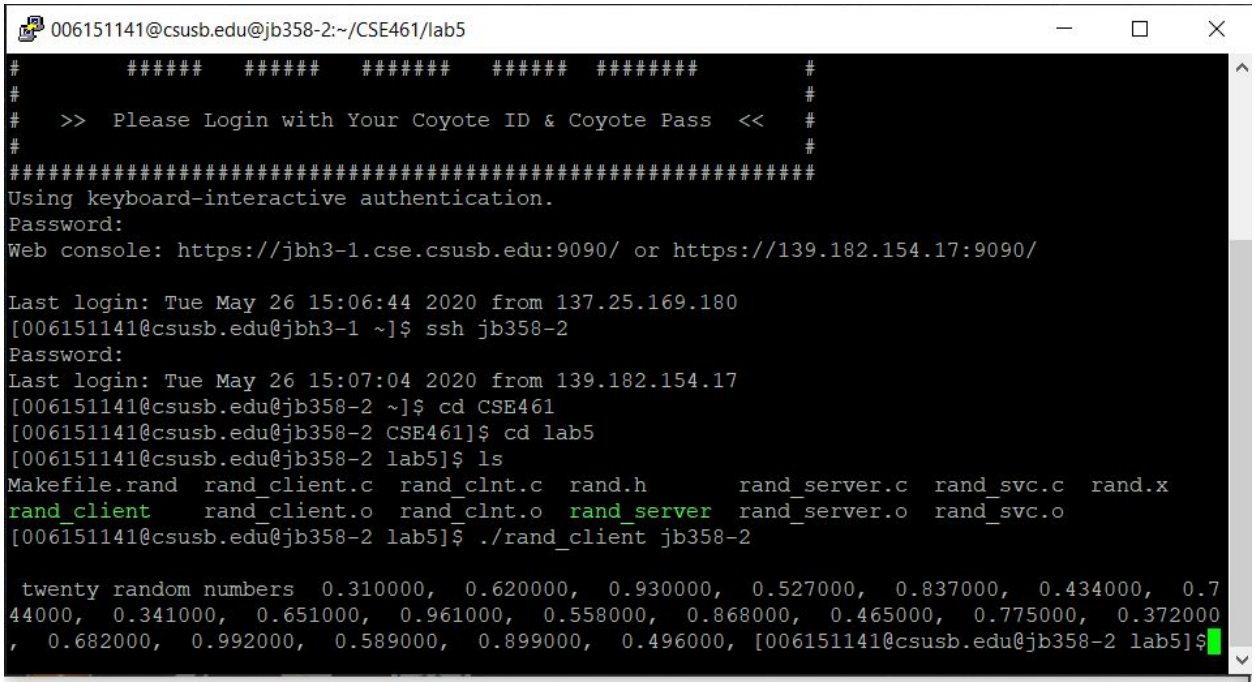
```
        printf("\n twenty random numbers ");
        for ( i = 0; i < 20; ++i ){
                x = rand_prog_1 (host);
                printf(" %f, ", x );
        }
exit (0);
}
```

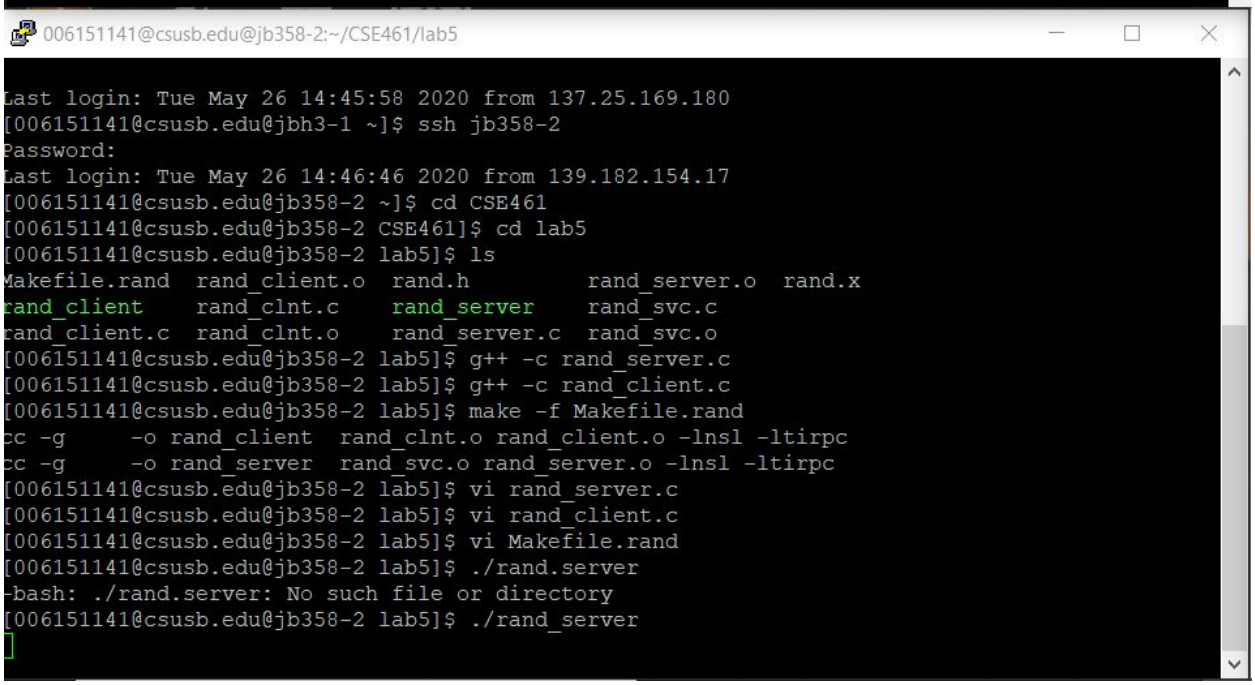
rand_server.c

```
/*
* 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;

   /*
    * insert server code here
    */
   result += 0.31;
   if(result >= 1.0)
       result -= 0.713;

   return &result;
}
```

Output:

- This is the output when you follow the instructions/demo based on what the lab manual says. It doesn't output a random number generator.





Source Code after creating our own Random Number Generator:

rand.x

```
/* rand.x */

program RAND_PROG {
 version RAND_VERS {
   void INITIALIZE_RANDOM ( long ) = 1;        /* service #1 */
   double GET_NEXT_RANDOM ( void ) = 2;        /* service #2 */
 } = 1;
} = 6151141;          /* program # */
```

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;

#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);
```

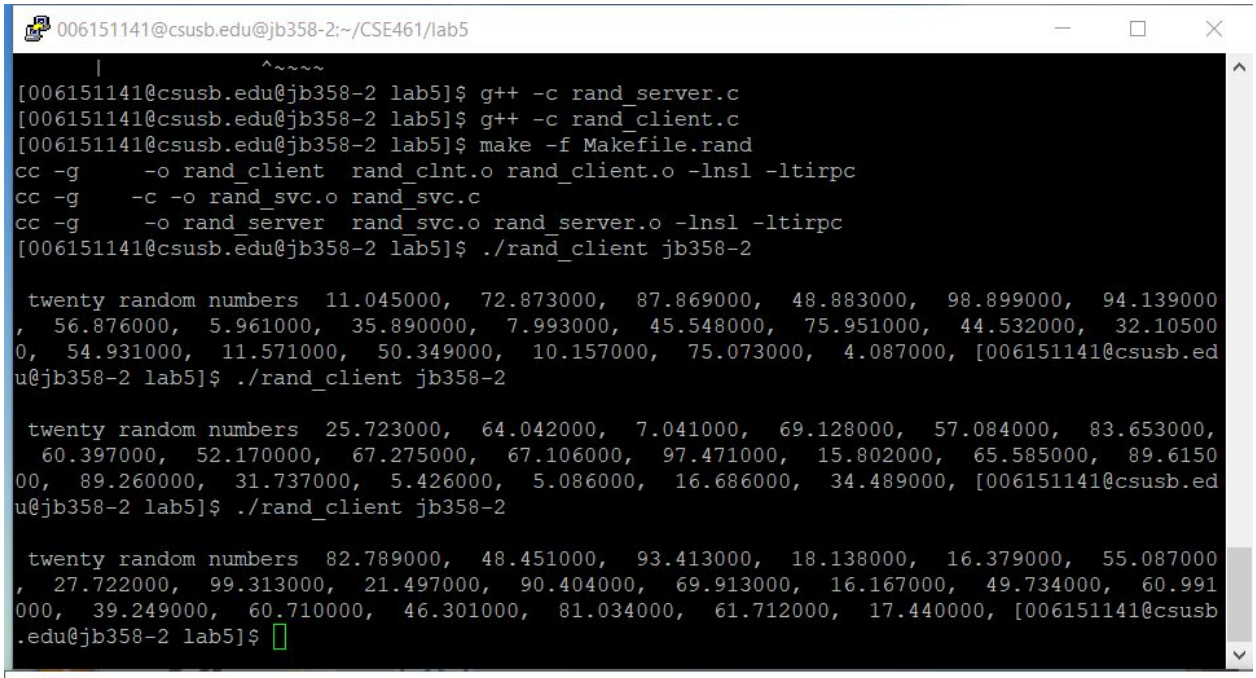
```
    if (result_2 == (double *) NULL) {
        clnt_perror (clnt, "call failed");
    }
#ifndef DEBUG
    clnt_destroy (clnt);
#endif    /* DEBUG */
    return *result_2;
}

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, ", x );
        }
exit (0);
}
```

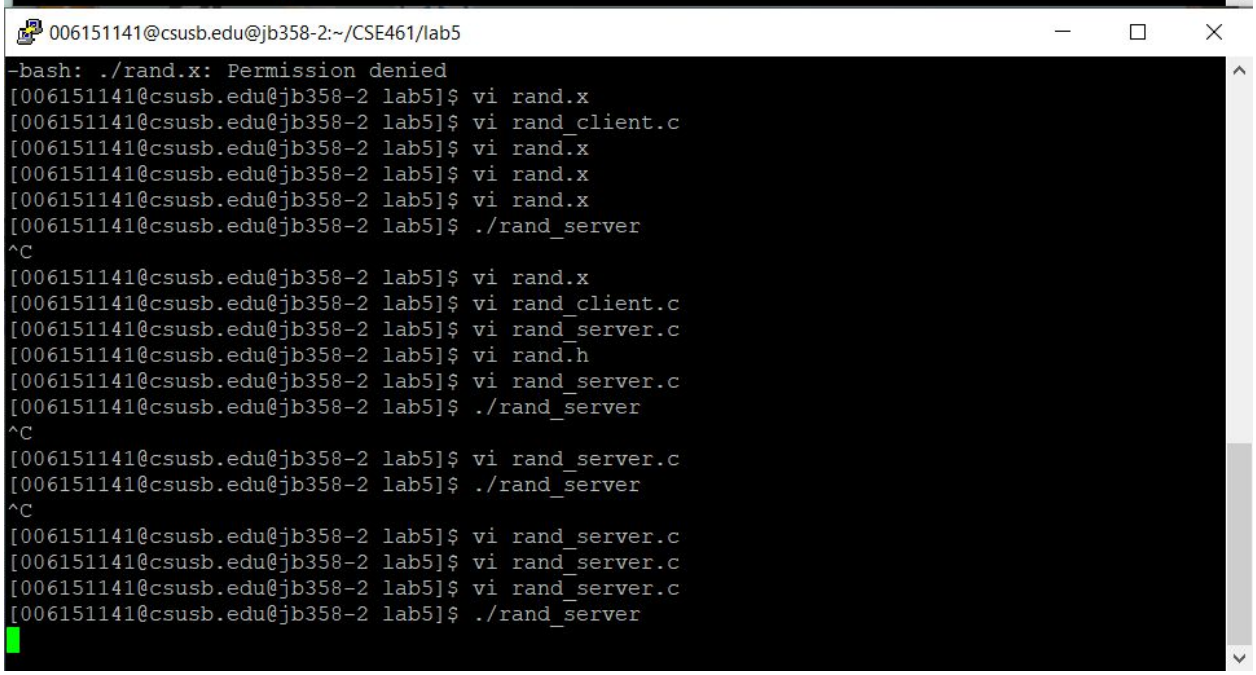
rand_server.c

```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"
#include "ctime"

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;

   /*
    * insert server code here
    */
   static int count = 0;
   count += 1;
   srand(time(NULL) + count);
   result = (double)(rand()%100000)/1000;

   return &result;
}
```

Outputs:

- This shows our outputs after implementing our own random number generator.





2. Parallel Random Number Generator

Source Code:

rand.x

```
/* rand.x */

struct params
{
   int xleft;
   int xright;
};

program RAND_PROG {
   version RAND_VERS {
       int GET_NEXT_RANDOM ( params ) = 1;     /* service #1 */
   } = 1;
} = 123456789;     /* program # */
```

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 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;

#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;
   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 );
   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];
    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);
}
```

rand_server.c

```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;
}
```

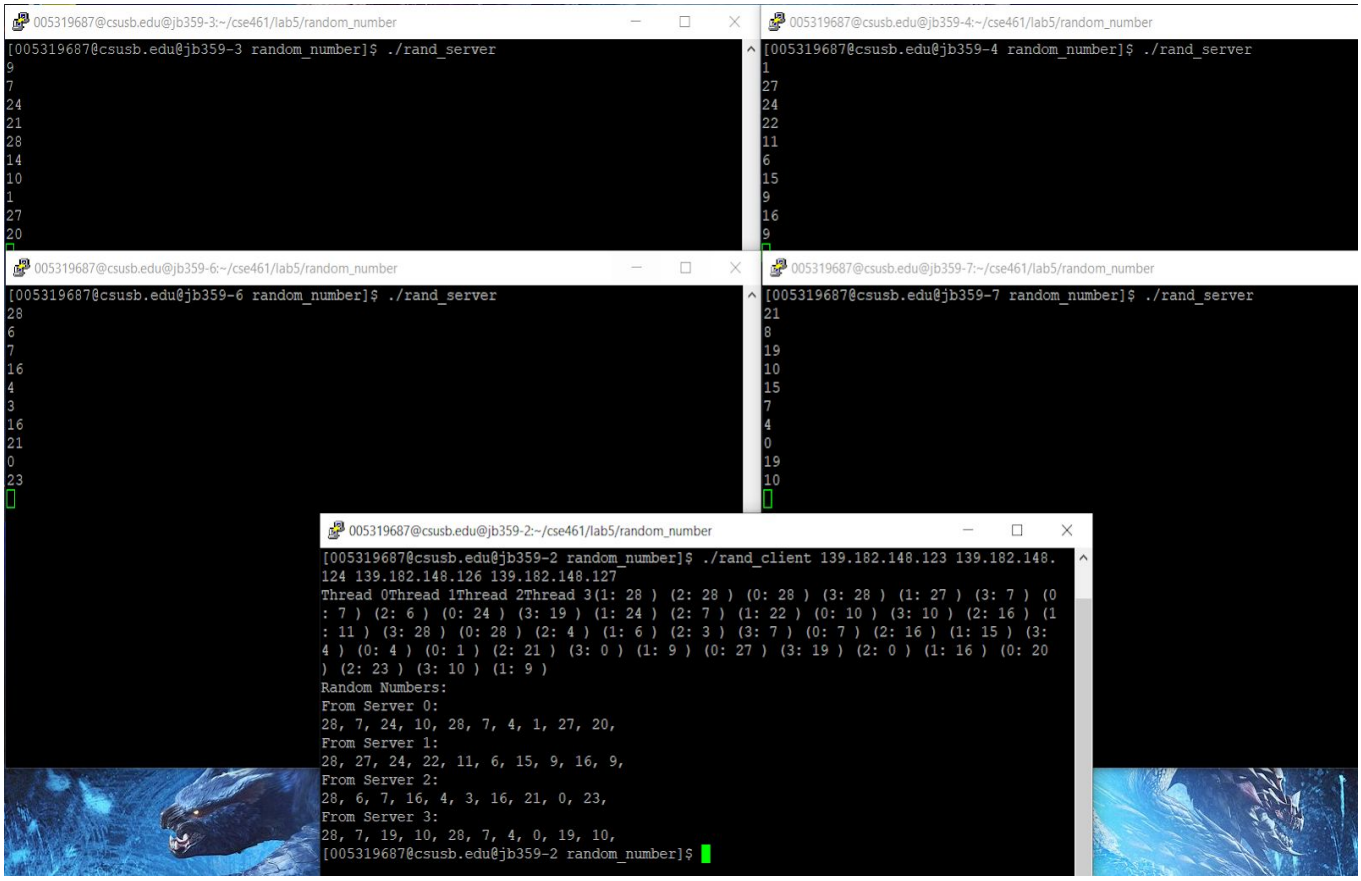
Makefile.rand

```
# 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)
```

Outputs:

- We completed the parallel random generator and also the extra credit part which is to run the program in 5 different machines/servers and it is shown below.



3. Report

- For this lab, the instructions on the website were very straight forward. However, one of the concerns/struggles that we had was implementing the random number generator to make the numbers different every time we open the server. For the Parallel Random Generator part, it’s easy to implement the whole code after watching the professor's video. Then we try to increase the machine number to get the extra credit and it works at the end. Overall, we implemented everything correctly based on the instructions on the lab manual. Therefore, we believe that we should get the full points plus the extra credit which is **30/20**.