Assignment: Project#04
Name: Christen Ford/Daniel Izadnegahdar
Csu Id#: 2741896/2420596
Turnin Id#: 2741896(chford)
Turnin Date: 04/28/20

1. **Summary:**
   1. This project exercises distributed computing using the remote procedure call(rpc). Distributed computing is a de-centralized method of computing that uses several computers to coordinate with each other to complete a common task. The advantage of this is that the computation is de-centralized and failure of independent components don't affect the whole. Another advantage is that it mitigates the risk of overloading a single computer.
   2. The RPC protocol is a type of distributed computing technique that allows programs to make requests with other programs located on other computers. It is a synchronous protocol that runs in sequence instead of in parallel. It also does not require complicated information about a network to succeed.
   3. This project will be exercised through 3 files: (1)rcalc.x, (2)rcalc.c, and (3)rcalc_svc_prov.c. The rcalc.x file is used to store specifications such as the RPC program number. The rcalc.c file is the client file that will serve as the user interface to the user for processing user requests. The rcalc_svc_prov.c file is the server file that contains 2 services: (1)sumsqrt_range() and (2)update_list(). The sumsqrt_range() service takes 2 arguments, and sums the square root of each integer in between, including the provided arguments. The update_list() service takes a list of numbers and returns F * F/10.0 on each entry. These 3 files are described in more detail below.

2. **RPC Sequence:**
   1. **Summary:**
      1. An RPC procedure call works by requesting a message from the client to a server, receiving the request on the server, processing the executables, sending a reply back to the client, and resuming execution on the client. Since RPC is synchronous, the calling environment is often suspended to maintain sequence. Stubs are used on the client and server side to maintain the interface and provide transparency.
      2. The 8 steps below are a generalized summary of building distributed applications using this RPC protocol. They are based of the book "Internet working With TCP/IP", volume 3, by Comer and Stevens.
   2. **Step#01:**
      1. Build an RPC application that takes basic commands from the user.
   3. **Step#02:**
      1. Choose a set of procedures that will run on the remote computer and divide the program.
   4. **Step#03:**
      1. Create a rpcgen specification for the remote program. Make sure it includes the remote procedure names, numbers, and arguments.
   5. **Step#04:**
      1. Run the rpcgen function to check the specification and, if it is valid, generate 4 source code files that will be used in the client and server.
   6. **Step#05:**
      1. Modify each routine's interface on the client and server side.
   7. **Step#06:**
      1. Compile and link 3 files together for the client program. Those files include the original application, the client-side stub, and the xdr procedures. The executable output becomes the client.
   8. **Step#07:**
      1. Compile and link 3 files together for the server program. Those files include the procedures taken from the original application, the server-side stub, and the xdr-procedures. The executable output becomes the server.
   9. **Step#08:**
      1. Run the server executable on the remote computer and then run the client

executable on the local computer.

3. **File rcalc.x:**
   1. **Summary:**
      1. This file contains specifications required for the network and calculations to work properly. It includes structs, program definitions, and the RPC number.
   2. **Code Description:**
      1. The file begins by describing 2 structs that are used for the 2 arithmetic functions of the program. They store variables such as the lower and upper limits of the sum of square roots function and the list for the update list function.
      2. Next, the program definitions are specified. It defines the RPC program number in hexadecimal format. The last 4 digits of 0x2000XXXX is the last 4 digits of the turning csuId#.

4. **File rcalc.c(client):**
   1. **Summary:**
      1. This file is used to take input from the user, send it to the server for processing, and display the results on the output, using the rpc protocol.
   2. **Code Description:**
      1. The code begins with a series of libraries required for string manipulation(string.h) and rpc network communication(rpc/rpc.h).
      2. When the program is executed, it starts with the main() function. The main() function first error checks to make sure the server is running so the client can be handled. Next, it checks if the provided arguments are either -r or -u, and sets a key variable accordingly. The key variable is used as an identifier for running specific functions through a switch statement. The switch statement contains 3 cases, an 'SUMSQRT_RANGE', 'UPDATE_LIST', and default case.
      3. The 1st case i.e. 'SUMSQRT_RANGE' first checks to make sure that the user provided the right number of arguments. If the proper number was provided, the code continues, by prepping to run the sum of square roots function. First, it defines the proper struct from the variable definitions of the rcalc.x file. Then, it converts the 3rd and 4th arguments to integers using the atoi() function. Finally, it runs the sumsqrt_range() function which will be described in more detail under the server section. The results are then printed to the screen using printf().
      4. The 2nd case i.e. 'UPDATE_LIST' also first checks to make sure that the user provided the right number of arguments. If the right number is provided, the code continues, by prepping to run the update list function. To begin, the list data size is allocated in the heap using the malloc() function. Next, the list runs through the update_list() function, to calculate F * F / 10.0 on each item. The result of each item is then printed to the screen.
      5. The last case is the default case. The default case runs when the key was neither of the 2 expected inputs described earlier. If this is the case, an error message is displayed and the switch statement exits out with an error argument of 1.

5. **File rcalc_svc_proc.c(server):**
   1. **Summary:**
      1. The server file is used to continuously run in the background, listen, and execute commands from the user. The 2 main commands it runs are the sumsqrt_range() and update_list() functions.
   2. **Code Description:**
      1. The code begins with a series of libraries required for math (math.h) and

rpc network communication(rpc/rpc.h).
2. The sumsqrt_range() function first allocates the proper space in the heap using malloc(). It then checks to make sure that the provided lower number is lower than the upper number. Next, it performs the calculation, by running a ‘for’ loop that iterates from the lower value to the upper. For each iteration, ‘rsqrt’ is incremented with the square root of the iterator. The iterator should be a number between the provided lower and upper values. Finally, the final sum of ‘rsqrt’ is returned.
3. The update_list() function first sets the current node to the head, and runs a while loop, to scan through the list, and run the F * F / 10.0 operation on each item of the list. After the while loop is complete, the result of the head is returned to the user.

6. File makefile description:

1. Summary:

1. This file defines a set of rules for use by the make program. It is used to execute commands with a simple pseudonym such as ‘make all’ or ‘make clean’. The 3 commands it is designed to run are ‘make’, ‘clean’, and ‘submit’. The ‘make’ command is used to compile and link the code, ‘clean’ is used to remove object and executables, and ‘submit’ is used to turn in the project to the grading server.

2. Code Description:

1. The file starts with a series of marcos such as CC, CFLAGS, and IFLAGS.
2. The file then lists the compile commands, client/server build, and watch commands.
3. The server compile command requires the ‘-lm’ link command because the server program needs to use the math library to run sqrt().
4. The file then lists the clean command. This command removes all of the generated object and executable files from the directory.
5. The file then lists the submit command. This command will run the turnin command, to submit all .c files, .x files, the makefile, and report files to a directory where the project will be graded.

7. Results:

1. Run Instructions:

1. Open a terminal, ssh to degas, browse to the directory of the project, and type ‘make’.
2. After the executables are generated, type ./rcalc_svc.
3. Open another terminal, ssh to spirit, and type ./rcalc degas -r 2 6.
4. After the output is displayed, on that same terminal, type ./rcalc degas -u 5 10 13 14 25.

2. **Run Output:**

    1. **make:**

```
daizadne@degas:~/Desktop/submission$ make
rpcgen rcalc.x
echo > rcalc_shared.h
gcc -c -o rcalc.o -DDEBUG rcalc.c
gcc -DDEBUG   -c -o rcalc_clnt.o rcalc_clnt.c
gcc -DDEBUG   -c -o rcalc_xdr.o rcalc_xdr.c
gcc -DDEBUG -o rcalc rcalc.o rcalc_clnt.o rcalc_xdr.o \

gcc -c -o rcalc_svc_proc.o -DDEBUG rcalc_svc_proc.c
gcc -DDEBUG   -c -o rcalc_svc.o rcalc_svc.c
gcc -DDEBUG -o rcalc_svc rcalc_svc_proc.o rcalc_svc.o \
rcalc_xdr.o  -lm
```

    2. **server:**

```
daizadne@degas:~/Desktop/submission$ ./rcalc_svc
```

    3. **client:**

```
daizadne@spirit:~/Desktop/submission$ ./rcalc degas -r 2 6
9.831822
daizadne@spirit:~/Desktop/submission$ ./rcalc degas -u 5 10 13 14 25
2.5 10.0 16.9 19.6 62.5
daizadne@spirit:~/Desktop/submission$
```

3. **Description:**
    1. The 1st screen shot shows the outputs for compiling the program and the 2nd screen shot shows running the server on the degas Linux computer. After the executables are created, the server is initiated, so it can now take commands from the user and run through the steps of the RPC protocol.
    2. The 2nd terminal shows the outputs for running the client commands. The 1st command shows the results for totaling the square roots of the 2 provided numbers. The 2nd command shows the results for running F * F/10.0 on each entry of the provided list.

4. **Rcpinfo:**
   1. **Output:**

```
daizadne@spirit:~/Desktop/submission$ rpcinfo -p degas
   program vers proto   port  service
    100000    4   tcp    111  portmapper
    100000    3   tcp    111  portmapper
    100000    2   tcp    111  portmapper
    100000    4   udp    111  portmapper
    100000    3   udp    111  portmapper
    100000    2   udp    111  portmapper
    100007    2   udp    909  ypbind
    100007    1   udp    909  ypbind
    100007    2   tcp    910  ypbind
    100007    1   tcp    910  ypbind
 536909897    1   udp  44052
 536909897    1   tcp  55359
 536892565    1   udp  41908
 536892565    1   tcp  49961
 572522497    1   udp  45836
 572522497    1   tcp  58449
 536908896    1   udp  58823
 536908896    1   tcp  48769
 536870929    1   udp  53289
 536870929    1   tcp  48179
 536876596    1   udp  44893
 536876596    1   tcp  36527
2576980377    1   udp  60899
2576980377    1   tcp  51859
 536909449    1   udp  43580
 536909449    1   tcp  42421
 572675111    1   udp  46252
 572675111    1   tcp  42977
 536892455    1   udp  43390
 536892455    1   tcp  54705
 536893585    1   udp  34112
 536893585    1   tcp  52811
 591730964    1   udp  60367
 591730964    1   tcp  48075
 591728913    1   udp  46342
 591728913    1   tcp  48521
 536877332    1   udp  44668
 536877332    1   tcp  45901
 536906121    1   udp  48117
 536906121    1   tcp  53291
 536870913    1   udp  51446
 536870913    1   tcp  54331
 536877206    1   udp  55610
 536877206    1   tcp  54261
daizadne@spirit:~/Desktop/submission$
```

2. **Description:**
    1. The rpcinfo() command is used to display RPC information, by calling an RPC server and reporting what it finds. By default, it displays the program, version, protocol, port#, and service#.
    2. The 1st column in this RPC information output is the decimal equivalent of the program number. The program number on rclac.x is 0x20001896, but it is in hexadecimal format. It is equivalent to 536877206 in decimal format, which is shown on the last 2 entries of the RPC information output. Under 536877206, the udp port# of 55610 and the tcp port# of 54261 is displayed.

8. **Experiences in testing/debugging:**
    1. **Error#01:**
        1. **File:**
            1. makefile
        2. **Message:**
            1. sumsqrt_range.c:(.text+0x70): undefined reference to `sqrt'
        3. **Solution:**
            1. The -lm link command was not used in the compile command to link the math library to the server file. In the C programming language, the math library is linked through the compile command.
    2. **Error#02:**
        1. **File:**
            1. rcalc_svc_proc.c
        2. **Message:**
            1. segmentation fault (core dumped) (after running sum of square roots)
        3. **Solution:**
            1. The malloc() function was required to allocate the space of the double data type in the heap section of the memory stack. Dynamic lists require this in the C programming language but not for higher-level languages such as Java or C#.
    3. **Error#03:**
        1. **File:**
            1. rcalc.c
        2. **Message:**
            1. result is 24677.35 instead of 9.83 after running ./rcalc degas -r 2 6
        3. **Solution:**
            1. The input arguments from the user were not properly converted to integer using atoi(). The atoi() function is used to convert a string representation of an integer into its value. The lower and upper variables are defined as integers in the rcalc.x file.