Introduction and implementation of Remote Procedure Call

Member: Chih-Hai, Su, oceani.c@nycu.edu.tw
An-Chih, Liu, andrewm711es@gmail.com

Number of pages: 12 pages in total (including cover page)

Summary:

In this report, we will introduce the concepts of Remote Procedure Call (RPC), Docker networking, how to build the entire system, and how to use the service. In addition, we made the project image public on Docker Hub, so that everyone can test whether the service is worked well. During the implementation process, we used alpine docker image as the project environment, used “libtirpc” Linux package to implement the RPC service, and built two containers with network bridge based on this image as client and server. Finally, we expect the service can work normally and prepare for the further implementation and improvement.

Introduction and Implementation of Remote Procedure Call

1. Introduction (Introduction, Preface)

- Motivation

  During the class, we learned Docker and network management concepts. We wanted to apply the knowledge to the term project, and there happened to be a class in the department, "Introduction to Network Programming", which teaches a bonus topic, Remote Procedure Call (RPC). This topic can be integrated with Docker and networking. Therefore, we thought that it is suitable as the term project of the class.

- Introduction to the study

  In this study, we will introduce the overall design in detail, including writing an interface description language (IDL), using rpcgen tool to compile and generate RPC-related files, and finally splitting a simple Add program into two different parts, client and server, to finish the RPC implementation. Next, we will create a Docker network so that both Docker containers have a network bridge and can understand each other's hostname. Finally, we deploy the built system to two Docker Containers and test whether the requirements are met.

- Results

  We implemented the RPC system and set up the server and client containers. These two containers are connected by a network bridge. They know each other's hostname, so that the client can request a simple add procedure from the server. It works normally and can be displayed in the terminal.

- How the team members divide the work
  - An-Chih, Liu: Overview, introduction and implementation of Docker network, and related reports.
  - Chih-Hai, Su: Built the Docker image and RPC service of the project, reported RPC service and how to set up the system.

2. Background Discussion

- Introduction to related applications (systems)
  - Remote program call

    A software communication protocol that one program can use to

request a service from another located in another computer on a network without having to understand the network's details. It is an example of Client/Server for distributed computing.

The process is shown in the figure, in which the Client Stub plays the role of proxy and is responsible for converting the request of the client program into an XML file for network transmission, while the Server Stub is responsible for converting the XML file back to the program request, looks up the corresponding function in the Server program, and requests the function. After the result is obtained, it is converted into XML and passed to the Client Stub, and the Client Stub converts the file into its program requirements and sends it back to the Client program to complete a request.

- Introduction to relevant application suites
  - Docker

    An open-source projects similar to virtual machines, but using operating system virtualization technology, which has the advantages of easily managing software environment, small size, fast startup, and easy integration.
  - ONC RPC

    ONC RPC is used to make RPC service, originated from Sun, and is a part of the Network File System (NFS) project. NFS' main purpose is to share file directories for different hosts to mount and use, and the intermediate requests are implemented through ONC RPC.
  - Libtirpc

    Libtirpc is a kind of ONC RPC, which can be used on Linux-like systems. It has a dynamic link library of C/C++, which is used to set up RPC services in this project.
  - Interface definition file, (IDL)

    IDL is a generic term for a language which lets a program, or an object written in one rule description file. It specifies the system version of RPC, the function name, parameters, and the return values in the system. It can be compiled by the IDL compiler like rpcgen.
  - Rpcgen

Rpcgen is an IDL compiler whose main function is to generate Client/Server Stubs and related header files. In addition, it can also generate examples of Client/Server, and the Makefile of the RPC system.

- Rpcbind

  Rpcbind can convert the program number specified by IDL to the host network address and reversely. When the client wants to use the RPC service of the server, the rpcbind tool on the server must be in the running state.

3. Main content of the topic

- Principles

  We use Libtirpc to implement the entire RPC system and use the network bridge to deploy the Docker network. Because the bridge network customized by Docker has a hostname query function, our Client can use the hostname of the server to request a procedure call to meet the requirements.

- How to construct
  - Folder settings are as follows

```
AdderRPC/
├── Dockerfile
└── files
    ├── add_client.c
    ├── add_clnt.c
    ├── add.h
    ├── add_server.c
    ├── add_svc.c
    ├── add.x
    ├── add_xdr.c
    ├── Makefile.add
    └── test1
```

  - RPC Step 1. Write Interface definition file (add.x)

```
program ADDPROG {
        version ADDVERS {
                int ADD(int, int) = 1;
        } = 1;
} = 0x23451111;
```

- RPC Step 2. Compile IDL

```
rpcgen -aNC add.x
```

- RPC Step 3. Write C language programs for Client and Server

```
/* add_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<stdio.h>
#include "add.h"

int *
add_1_svc(int arg1, int arg2,  struct svc_req *rqstp)
{
        static int  result;

        printf("Got request: add(%d, %d)\n", arg1, arg2);
        result = arg1 + arg2;

        return &result;
}
```

```
/* add_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<stdio.h>
#include<stdlib.h>
#include "add.h"

void
addprog_1(char *host, char* testfile)
{
	CLIENT *clnt;
	int  *result_1;

#ifndef DEBUG
	clnt = clnt_create (host, ADDPROG, ADDVERS, "tcp");
	if (clnt == NULL) {
		clnt_pcreateerror (host);
		exit (1);
	}
#endif  /* DEBUG */

	FILE* fptr;
	fptr = fopen(testfile, "r");

	int tot;
	fscanf(fptr, "%d", &tot);
	for(int i = 0; i < tot; i++) {
		int x, y;
		if(fscanf(fptr, "%d%d", &x, &y) != 2) {
			printf("Wrong input type of format!\n");
			break;
		}
		result_1 = add_1(x, y, clnt);
		if (result_1 == (int *) NULL) {
			clnt_perror (clnt, "call failed");
```

```
                }
        else {
                printf("Got Ans: %d\n", *result_1);
        }
    }
#ifndef DEBUG
    clnt_destroy (clnt);
#endif   /* DEBUG */
}


int
main (int argc, char *argv[])
{
    char *host;
    host = (char*)"rpc-server";

    if (argc != 2) {
        printf ("usage: %s ${testfile}\n", argv[0]);
        exit (1);
    }
    addprog_1 (host, argv[1]);
exit (0);
}
```

- RPC Step 4. Add the C/C++ dynamic link library in Makefile.add.

  ```
  LDLIBS += -lnsl -ltirpc
  ```

- RPC Step 5. Write a testfile (test1). The first line represents several requests, and then each line has two numbers or strings representing the parameters to be added. If they are all numbers, the addition result will be the output, otherwise it will output the “Input Format Error”.

```
6
23 436
15 2
1 1
43 34
666666 0
2 dog
```

- Docker Step 1. Write Dockerfile

```
FROM alpine:3.17
MAINTAINER oceani.c@nycu.edu.tw

ADD files/* /usr/src/app/

WORKDIR /usr/src/app

RUN apk add build-base
RUN apk add libnsl-dev
RUN apk add libtirpc
RUN apk add rpcgen
RUN apk add rpcbind

RUN ln -s /usr/include/tirpc/rpc /usr/include/rpc
RUN ln -s /usr/include/tirpc/rpcsvc /usr/include/rpcsvc
RUN ln -s /usr/include/tirpc/netconfig.h /usr/include/netconfig.h
```

- Docker Step 2. Create a project image file (located outside the AdderRPC folder)

```
sudo docker build -t myrpc_x64 AdderRPC
```

- Docker Step 3. Create a network bridge.

```
sudo docker network create --driver bridge rpc-bridge-net
```

- Docker Step 4. (The last step), create Client/Server containers.

```
sudo docker run -dit --name rpc-server --network=rpc-bridge-net myrpc_x64 ash
sudo docker run -dit --name rpc-client --network=rpc-bridge-net myrpc_x64 ash
```

- Practical operation examples

- Step 1. Enter the server to enable the service with the background mode and exit the container.

```
sudo docker exec -it rpc-server ash
rpcbind
make -f Makefile.add
./add_server &
exit
```

- Step 2. Enter the client and test the system.

```
sudo docker exec -it rpc-client ash
rpcbind
make -f Makefile.add
./add_client test1
exit
```

- Step 3. Delete all generated related files.

```
sudo docker rm -f rpc-server
sudo docker rm -f rpc-client
sudo docker network rm rpc-bridge-net
sudo docker rmi -f myrpc_x64
```

- Remarks: test link for oral presentation

  For the convenience of the oral report, we have pushed the image file of the original project to Docker hub for public access. Among them, we have compiled the two programs of Client and Server in advance, so the make step was omitted during the oral report. The link to its test. The link to oral representation PPT.

4. Questions and Discussion

- Difficulties & how to overcome them?

  Our system had been built on Ubuntu, but when we put the files in the Docker image, we found that many packages have to be installed by ourselves, and there would be software compatibility problems. Later, we concluded that the software should be installed and compiled at the beginning of building the Docker container, because the Ubuntu executable file cannot be directly executed in alpine docker. Therefore, it is necessary to write a Docker file suitable for the project, and set the location of the

folder to facilitate software management and deployment.

- Questions and discussions
  - Why to design a remote call program system?

    When the demand for a single program increases, we can distribute part of the program to another machine for execution, and the implementation does not need to pay attention to underlying rules of network protocols, which is easy to implement.
  - Why to use alpine docker among many Dockers?

    The advantage of Alpine docker is that it has a small size (only 5Mb), fast download speed, and improved security, which is suitable for system deployment.
  - Why to use bridge mode to connect containers?

    The bridge mode can provide external and internal connections, and Docker's bridge mode also has a hostname query function, which is suitable for the RPC system.
  - In addition to ONC RPC, what are other RPC specifications and features?

    There are JAVA, JSON, Hessian, Protobuf, etc. Java has better system compatibility, but has poor performance, and needs to implement with JAVA; JSON has better readability and supports cross-platform programming languages but has a large overhead; Hessian has small overhead and multi-language support, but it has poor compatibility; Protobuf also has less small overhead, supports cross-platform languages, and has high efficiency, but its IDL specification is more complicated.
  - Experience

    In this implementation, we cultivated the ability of self-study and problem solving, became familiar with the methods of software management, and gained a better understanding of remote program calls.

5. Conclusions (and ideas for future research)

In this project, we successfully established a remote program call service in two Docker containers. The advantage of the RPC is that it can distribute a large number

of requests from a single program. We hope to design a load-balancing system in the future, so that the client can distribute requests to multiple servers, reducing the program burden.

References

- Remote Procedure Call https://www.ibm.com/docs/en/aix/7.1?topic=concepts-remote-procedure-call, Accessed on Jan 15, 2023.
- Network File System https://www.ibm.com/docs/en/aix/7.1?topic=management-network-file-system, Accessed on Jan 15, 2023.
- [Docker] Introduction to Bridge Network https://godleon.github.io/blog/Docker/docker-network-bridge/, Accessed on Jan 15, 2023
- Dockerfile reference https://docs.docker.com/engine/reference/builder/, Accessed on Jan 15, 2023.