學期專案書面報告

遠端程序呼叫介紹與實作

題目： A Study on Remote Procedure Call

成員： 蘇智海(110550141), oceani.c@nycu.edu.tw
劉安之(110550133), andrewm711es@gmail.com

繳交時間： 2022.1.16
頁數： 共 11 頁(含封頁)

摘要內容：

我們用上課講述的 Docker 概念，延伸加上橋接式網路讓 Docker 容器能互相溝通，並使用它在 Docker 上實作遠端程序呼叫。遠端程序呼叫是一種通過網絡從遠端電腦程序上請求服務，而不需要了解底層網絡技術的通訊協定，也就是說，我們可以架設跨主機的服務但不用寫太多關於 Socket 底層的概念。這個專案我們將會從頭到尾介紹建設 RPC 與 Docker 網路的方法，並實際展示我們的成果，並公開分享 Docker image 給需要的人使用。我們的 RPC 系統架構在 alpine docker，因為 alpine docker 比較輕巧，效率高，具安全性，且基本功能與 Ubuntu 類似，是一種適合開發輕量級專案的工具。最後在這次的實作中，我們對軟體管理與遠端程序呼叫有了更深入的了解。

遠端程序呼叫介紹與實作

1. Introduction (概論, 前言)

- 動機

  在上課時我們有學到 Docker 使用和部屬，也有學到網路管理的相關觀念，於是我們想找一個可以實作在 Docker 容器彼此網路通訊的專題，剛好系上有一堂課「網路程式設計概論」在教 socket 使用和寫服務，後面有一個延伸課題是遠端呼叫程序 (Remote Procedure Call, RPC) ，他不用寫太複雜的底層網路規範，適合用於快速部屬在分散的系統上。於是我們想建設這個系統，並配合 Docker 網路，讓兩個 Docker 容器能彼此溝通。

- 研究的簡介

  這個研究我們將詳細的跑過建設 RPC 系統的流程，包含從編寫一個介面描述語言 (IDL)，使用 rpcgen 編譯生成 RPC 相關的文件，到最後將一個簡單的 Add 程序拆成 client 和 server 兩個不同的程序完成實作。再來我們會建設 Docker network，讓兩個 Docker 容器皆有橋接網路，並彼此知道對方的 hostname。最後我們把建設好的系統部屬到兩個 Docker Container 上，並測試是否達到需求。

- 成果

  我們實作了 RPC 系統，及架設了 server 及 client 容器，這兩個容器有橋接網路連接，且彼此知道對方的主機名稱，能讓 client 從 server 請求簡單的 add 程序，正常運作，並成功在終端機顯示。

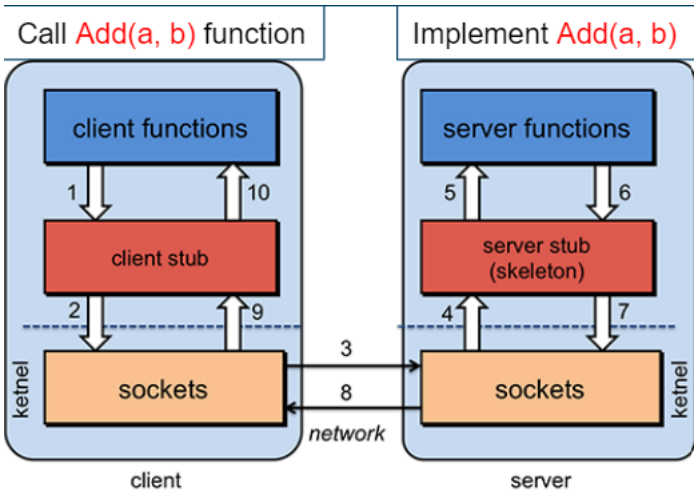
- 組員如何分工
  - 劉安之：概述、Docker network 介紹與實作，及相關報告。
  - 蘇智海：建構專案的 Docker image、遠端程序呼叫(RPC)介紹與實作，及相關報告。

2. 背景探討

- 相關應用 (系統) 簡介
  - 遠端程序呼叫：

    一種通過網絡從遠端電腦程序上請求服務，而不需要了解底層網絡技術的通訊協定，是一個分散式計算的 Client/Server 例子。

    流程如圖，其中 Client Stub 扮演 Proxy 的角色，負責將 client 程序的請求轉成 XML 文件利於網路傳輸，而 Server stub 則負責將 XML 文件轉回程式請求，並尋找 Server 程序中有沒有對應的函式可供呼叫。得到結果後將其轉成 XML 傳給 Client Stub，而 Client Stub 將文件轉成其程序需求傳回 Client 程序，完成一次請求。



- 相關應用套件簡介
  - Docker

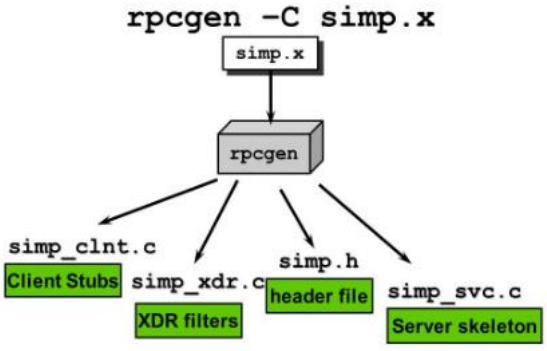
    開源專案，類似於虛擬機器，但使用作業系統虛擬化技術，優點是軟體環境管理容易，體積小，啟動快，且易於整合。
  - ONC RPC

    ONC RPC 源自於昇陽電腦，是網路檔案系統 (NFS) 計畫的一部分，他的主要目的是共享檔案目錄給不同主機掛載使用，而中間的請求則透過 RPC 實作，也就是上述遠端程序呼叫的功能。
  - libtirpc

    libtirpc 是 ONC RPC 的一種，可於類 Linux 系統上使用，是 C/C++的動態連接庫，用於架設 RPC 服務。
  - Interface definition file, (IDL)

    IDL 是遠端程序介面的規則說明文件，其規範 RPC 的系統版本，系統中函式名稱、參數、及回傳值，可供後續其編譯器使用。
  - rpcgen

    rpcgen 是一種 IDL 的編譯器，其主要功能是生成 Client stub/Server stub 及其相關標頭檔。此外，它也可以生成 Client/Server 的範例，及最終編譯整個專案的 Makefile。



- rpcbind

  rpcbind 工具可以將 IDL 規定的程序號碼和主機網路地址互相轉換，當 client 要使用 server 的 RPC 服務時，該 server 上的 rpcbind 必須處於運行狀態。

3. 專題主要內容

- 運作原理

  我們使用 Tirpc 架構整個 RPC 系統，以及使用橋接網路部屬 Docker network。因為 Docker 自訂的橋接網路有 hostname 的查詢功能，所以我們的 Client 能利用 server 的主機名稱請求程序呼叫，達成需求。

- 如何建構
  - 資料夾設定如下

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

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

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

- RPC Step 2. 編譯 IDL

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

- RPC Step 3. 撰寫 Client 和 Server 的 C 語言程式

```
/* 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. 在 Makefile.add 加上動態連結庫。

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

- RPC Step 5. 撰寫 testfile (test1)，第一行代表有幾個請求，之後每一行皆有兩個數字或字串代表要相加的參數，如果皆為數字則會輸出相加結果，否則會輸出 Input Format Error.

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

- Docker Step 1. 撰寫 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. 創建專案映像檔 (位置在 AdderRPC 資料夾外)

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

- Docker Step 3. 創立一條橋接式網路

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

- Docker Step 4. (此為最後步驟)，建立 Client/Server 容器

```
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
```

- 實際運作範例
  - Step 1. 進入 server 中將服務於背景模式開啟並退出容器

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

- Step 2. 進入 client 中測試結果

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

- Step 3. 刪除所有產生的相關文件

```
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
```

- 備註：口頭報告的測試連結

  為了口頭報告方便，我們有將原始專案的映像檔 push 到 Docker hub 供公開存取。其中我們事先編譯完了 Client 和 Server 兩個程序，故口頭報告時省去了 make 步驟。以下是其測試連結。

  https://hackmd.io/@oceanic/SyhC7Co5i

4. 問題與討論

- 重點: 遭遇困難 & 如何克服

  我們的系統在 Ubuntu 建設，然而將檔案放置在 Docker image 時，會發現很多套件都要自己安裝，且會有軟體相容的問題。後來我們歸納出應該在一開始建設 Docker 容器時就把軟體裝好與編譯，因為 Ubuntu 的執行檔不能直接在 alpine docker 執行。所以要寫一個適合專案的 Dockerfile，並設定好資料夾位置，以方便軟體管理與部屬。

- 問題與討論
  - 為什麼要設計遠端呼叫程序系統

    當單一程序使用需求增加時，我們可以將部分程式分散到另一台機器執行，且其不用注重網路協定等規則，實作容易。
  - 在眾多 Docker 中為什麼使用 alpine docker

    Alpine docker 的優點在於它檔案小，(僅僅只有 5Mb)，下載速

度快，與提高安全性，適合系統部屬使用。

- 為什麼使用橋接模式連接容器

  橋接模式可以提供對外與對內連結，且 Docker 的橋接模式還有 hostname 查詢功能，方便使用。

- 除了 ONC RPC 外，還有哪些 RPC 規範與特性

  有 JAVA, JSON, Hessian, Protobuf 等等，Java 的系統兼容性較好，但性能較差，單一語言；JSON 的可讀性較好，且支援跨語言，但性能與空間開銷較大；Hessian 的性能及空間開銷較少，有多語言支持，但兼容性不佳；Protobuf 同樣性能及空間開銷較少，支援跨語言，效率高，但其 IDL 規範較為複雜。

- 心得

  在這次架設系統中，我們培養了自學以及問題解決的能力，熟悉了軟體管理的方法，且對遠端程序呼叫有了進一步的了解。

5. 結論 (與將來研究的想法)

這次專案我們成功在兩個 Docker 容器建立遠端程序呼叫的服務，其中有關遠端程序的優點是可以分散負擔單一程序的大量請求。我們希望之後能設計一個平衡負載的系統，讓 Client 可以將請求分散到多個不同的 Server 上，減少程序負擔。

參考資料

- 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] Bridge Network 簡介 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.