

計算機網路概論

期末專案研究報告 -

FTP-Proxy Rate Control

Group 2

104062111 游欣為

104080002 林子馨

104062225 徐唯瑄

January 14, 2017

摘要

在這次課程中我們學習到 FTP Proxy 為中間控制 client 和 server 中間的媒介，而這次的 Final project 就是要讓我們利用助教提供的 FTP Proxy sample code 作為參考，加上自己的 rate control function 實作出控制上傳及下載速度的 Proxy。

首先介紹如何使用本次的 Project，接著講解我們所理解的 FTP Proxy 運作方法，像是如何讓 Client 端和 Server 端溝通，和怎麼做平行處理的。再來描述我們是如何實作出速度控制的 Function，使用到什麼原理以及演算法。

最後講解所遭遇到的困難以及實驗數據，跟工作分配。

關鍵字：FTP Proxy、Socket programing、Rate control、Fork and child

目錄

摘要	2
目錄	3
圖表目錄	2
I. 緒論	5
A. 研究目標	5
B. 開發平台介紹 - Filezilla	5
C. 程式執行方法 - Filezilla	6
II. 程式架構	7
A. Main	7
B. Connect FTP	9
C. Proxy Function	9
D. Create Server	10
III. 速度控制實作	12
A. 速度計算	12
B. 限制方法	13
IV. 問題解決	14
A. 速度暴衝	14
B. Address already in use	14
V. 實驗數據	15
VI. 工作分配	17

圖表目錄

圖1.1 Filezilla 介面圖	5
圖1.2 terminal 資料夾開啟	6

圖 1.3 terminal cpp 編譯	6
圖 1.4 terminal Open FTP proxy	6
圖 1.5 Filezilla 站台管理員	6
圖 2.1 TCP Socket workflow	7
圖 2.2 unistd.h header.....	7
圖 2.3 main - fork function 部分程式碼	8
表 2.1 Fork() 回傳值	8
表 2.2 Connect FTP - 程式運作細節.....	9
圖 2.4 Create Server - bind and listen 程式碼	9
圖 2.5 Main Function - create_server 程式碼	9
表 2.3 select 函式原型.....	10
表 2.4 select 函式原型描述.....	10
表 2.5 FD_SET macro	11
圖 2.6 Proxy_function - select 程式碼.....	11
圖 2.7 get the start time 程式碼	12
圖 2.8 end time 程式碼.....	12
圖 2.9 Rate Control 程式碼	13
圖 2.10 proxy_func - rate_control 程式碼	13
圖 4.1 Filezilla 站台管理員設定.....	14
圖 4.2 Address already in use 示意圖	14
表 6.1 工作分配表	17

I. 緒論

A. 研究目標

我們想要利用 Rate Control Function 控制 在 Filezilla 上下載以及上傳的速度，盡量將其控制在誤差值 3 以內，單位為 KByte。完成後不僅能更加理解 FTP Proxy 得運作方式，更能自己做出速限，方便生活中的檔案讀寫。

B. 開發平台介紹 - Filezilla

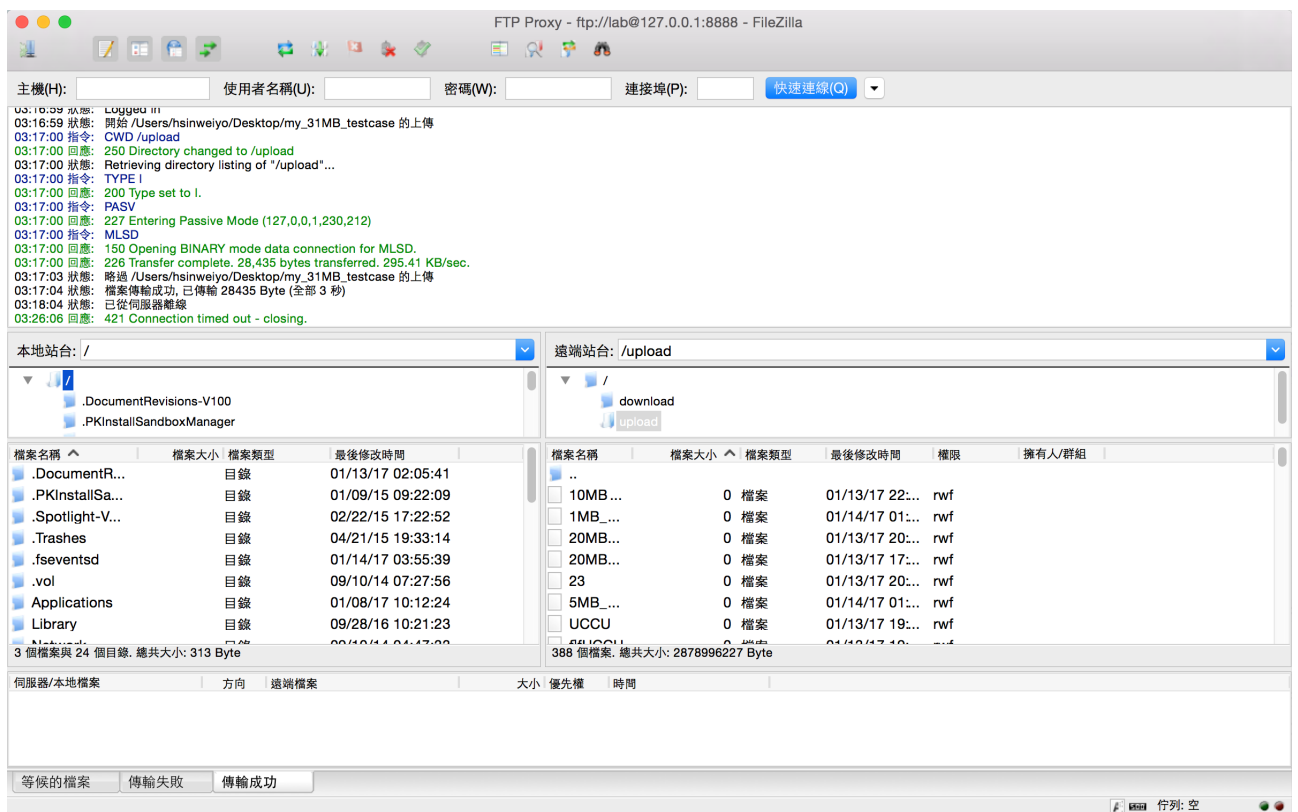


圖1.1 Filezilla 介面圖

依據 Filezilla 官方介紹，Filezilla 是免費的 FTP solution。不管是當作 Client 端或是 Server 端都是可行的，Filezilla 是免費的開放資源的軟體，並且是有 GNU General Public License 的。

C. 程式執行方法

1. 開啟 terminal (mac 上)，進入 Ftp_proxy 所在資料夾

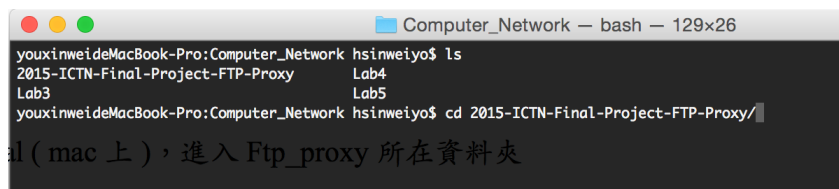


圖 1.2 terminal 資料夾開啟

2. 編譯 c 或 cpp 檔



圖 1.3 terminal cpp 編譯

3. 執行 FTP Proxy 並且輸入

<Proxy IP> <Proxy Port> <Download Rate> <Upload Rate>

```
youxinweideMacBook-Pro:2015-ICTN-Final-Project-FTP-Proxy hsinweiyo$ ./here 127.0.0.1 8888 25 25
```

圖 1.4 terminal Open FTP proxy

4. 打開 Filezilla 輸入所有所需資訊按下連線



圖 1.5 Filezilla 站台管理員

II. 程式架構

A. Main

1. 得到 4 個變數 (FTP Port / FTP IP / Download Rate / Upload Rate)，少於 4 個會印出 **[v] Usage: ./executableFile** 並且回傳 -1 結束 main function。
2. 前半段的 Main Function 讓是在做 socket 連接，這部分的程式碼我們在 Lab 2 以及 Lab 3 已經熟悉過了，就不加以贅述：

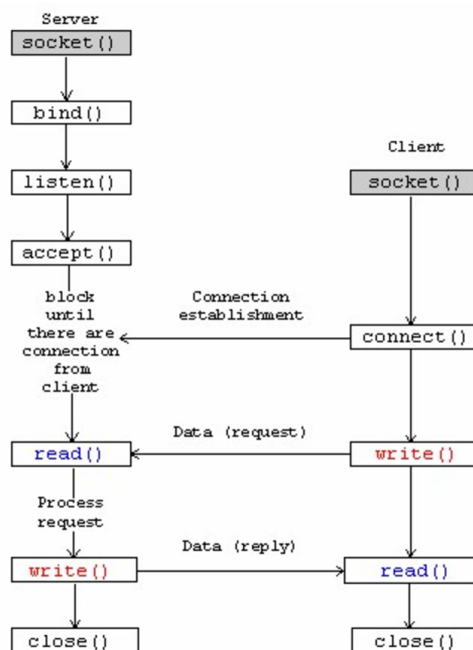


圖2.1 TCP Socket workflow

3. fork () 運作原理：

看似不起眼短短一句的 `childpid = fork()` 的背後其實用到了很複雜的觀念，我們現在來解釋我們所理解的 `fork()` 運作方式。

```
#include <unistd.h>
```

圖2.2 unistd.h header

首先先 include `unistd.h` 我們會用到裡面的 `fork()` 跟 `getpid()`，接下來宣告我們的 `pid_t` (short 類型變量) `childpid` 用來存取 `fork()`。接下來呼叫 `fork()`。

```

if ((childpid = fork()) == 0) {
    printf("Child: my pid is %d, parent pid is %d\n",
        getpid(), getppid());
}

```

圖2.3 main - fork function 部分程式碼

利用 childpid 等於零時印出他的行程辨識元 (process identifier)，此時我們應該在他的 child process 裡面。而 fork() 回傳值有 -1, 0, 1，而這些分別代表了不同的意思，為了理解 fork 跟 pid 的原理我們參考了文獻。

“整個作業系統是為了分配有限的硬體資源給 Program 使用，但是 OS 也視為一個 Program，放進 CPU 中執行，為了讓 OS Kernel 辨識每個 Process，就會有所謂的行程辨識元 (Process Identifier, PID)，其中特別的是 pid 0 和 1，0 是 Kernel 用來 Swap，交換分頁使用；而 1 則是初始化 Process，也是 Kernel 建立的第一個 Process，其他的 Process 基本上都是透過 pid 1 的 init Process 來產生。”

所以我們理解我們在進入子行程後才能進入 Proxy_funciton。

表2.1 Fork() 回傳值

Fork() 回傳值	Process	meaning
0	child process	we're in the child process
1	parent process	we're in the parent process
-1	parent process	in parent process with error

當我們進入 child process 就關掉 listening socket 然後在現在對 accepted 的 socket 做事情。接著進入 proxy_func 。

B. Connect FTP

跟我們 Lab3 做的事一樣，但是我們這裏在連接並把 port number 還有 IP address 拿來使用。

表2.2 Connect FTP - 程式運作細節

Function	description
socket	創造新的 socket
bzero	把 FTP port 跟 address 存起來
inet_pton	把字串格式的 IP address 封裝到 struct sockaddr_in
connect	把main 裡面建好的 clifd 跟現在的 ser_port 連結

C. Create Server

跟上面很雷同，在 bind () 之後 listen socket 並把 listen 到的 socket 回傳到剛剛 main function 裡面來做 connection。

```
if (bind(listenfd, (struct sockaddr *)&servaddr, sizeof(servaddr)) < 0) {  
    //print the error message  
    perror("bind failed. Error");  
    return -1;  
}  
  
listen(listenfd, 3);  
return listenfd;
```

圖2.4 Create Server - bind and listen 程式碼

```
ctrlfd = create_server(port);
```

圖2.5 Main Function - create_server 程式碼

D. Proxy Function

這是所有 function 裡面最重要的一個，所有封包傳送以及接收都在這裡進行，client 端跟 server 端透過 Proxy 在互相溝通，我們也在這裡發現 sample code 的 bug。

1. select () :

其中用到了 select 這個 function，select () 授予我們同時監視多個 sockets 的權力，它會告訴我們哪些 sockets 已經有資料可以讀取、哪些 sockets 已經可以寫入，如果我們真的想知道，還可以告訴我們哪些 sockets 觸發了例外，這個函式以 readfds、writefds 及 exceptfds 分別表示要監視的那一組 file descriptor set (檔案描述符集合)。

表2.3 select 函式原型

```
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>

int select(int maxfdp, fd_set *readfds, fd_set *writefds,
           fd_set *exceptfds, struct timeval *timeout);
```

表2.4 select 函式原型描述

select 參數	Description
int maxfdp	參數應該要設定為 file descriptor 的最高值加 1
fd_set *readfds	指向 fd_set 結構的指針，用來監視這些文件描述符是否有可讀的文件，也可以傳入NULL值表示不關心任何文件的讀變化。
fd_set *writefds	指向 fd_set 結構的指針，用來監視這些文件描述符是否有可寫的文件，也可以傳入NULL值表示不關心任何文件的寫變化。
fd_set *exceptfds	同上面兩個參數，用來監視文件的錯誤和異常，也可傳入NULL值。
struct timeval *timeout	大於 0 就是等待的超時時間，select 在 timeout 時間內阻塞就會 return 0

這裏稍微簡介一下所用到的 FD_SET。

表2.5 FD_SET macro

FD_SET	(int fd, fd_set *set)	將 fd 新增到 set
FD_CLR	(int fd, fd_set *set)	從 set 移除 fd
FD_ISSET	(int fd, fd_set *set)	若 fd 在 set 中，傳回 true
FD_ZERO	(fd_set *set)	將 set 整個清為零

select return 值：

- 負數：select() 發生錯誤。
- 正數：某些文件描述符可讀寫或出錯。
- 0：等待超時，沒有可讀寫的文件。

```
nready = select(maxfdp1 + 1, &rset, NULL, NULL, NULL);
if (nready > 0) {
    // check FTP client socket fd

    if (FD_ISSET(clifd, &rset)) {
        memset(buffer, 0, MAXSIZE);
        if ((byte_num = read(clifd, buffer, MAXSIZE)) <= 0) {
            printf("[!] Client terminated the connection.\n");
            break;
        }
    }
}
```

圖2.6 Proxy_function - select 程式碼

2. 判斷 upload 跟 download

我們對 upload 以及 download 分別限速，所以要先分辨出何時作 upload 何時做 download，我們知道

- 第一個讀入與寫出是上傳，因為是對 client 端做讀取，然後對 server 端做寫入。
- 第二個讀入與寫出是下載，因為是對 server 端做讀取，然後對 server 端做寫入。

III. 速度控制

A. 速度計算

我們知道，既然要做速度控制，代表我們要讓平均速度保持在我們想要的速限內，而要如何做到這點，首先我們要先知道現在上傳或是下載的速度。

1. 經過時間計算：

我們在遞迴呼叫 proxy_func 之前先用 gettimeofday 函式得到當時的時間。

```
#include<sys/time.h>
int gettimeofday( struct timeval*tv, struct timezone *tz )

struct  timeval{
    long  tv_sec;   秒
    long  tv_usec;  微秒
};

struct  timezone{
    int tz_minuteswest; 和greenwich 時間差了多少分鐘
    int tz_dsttime;      type of DST correction
}
```

2. 資料傳輸量

我們知道 byte_num = read(clifd, buffer, MAXSIZE) 就是 byte 數，所以在每次進入上傳或下載檔案的時候就用 data_sum += byte_num 做運算。

```
/* start time counter */
gettimeofday(&t, NULL);
prev_moment = t.tv_sec * 1000.0 + t.tv_usec / 1000.0;
```

圖2.7 get the start time 程式碼

```
/* Rate Control */
data_sum += byte_num;
gettimeofday(&t, NULL);
cur_moment = t.tv_sec*1000.0 + t.tv_usec/1000.0;
current_rate = (1.0*data_sum) / (cur_moment-prev_moment);
rate_control(byte_num, up_rate);
```

圖2.8 end time 程式碼

B. 限制計算

因為既然速度過快，那我們就利用讓系統 `usleep` 的方式去控制上傳或下載的速度，`usleep` 的用法就是 `usleep (microseconds)`; 為了配合我們上傳下載的速度，我們直接使用毫秒在微調速度。

先算出目前速度下所花費的時間，因為要轉換成 KByte 所以我們直接乘上 1024 並且除以剛剛得到速度，分別用目前的速度跟目標速度去計算時間應該要花費多少。得到的結果相減就是時間差，但是因為單純只睡兩者差距的時間是不夠的，必須乘上一個特定的係數讓他降低平均速度。

```
void rate_control(int byte_sum, int rate) {
    double current_cost, target_cost;

    target_cost = ( 1.0 * byte_sum) * 1024.0 / rate;
    current_cost = ( 1.0 * byte_sum) * 1024.0 / current_rate;

    double difference = target_cost-current_cost;

    if(difference < 0) return;
    usleep(difference * (rate <= 50 ? 39 : rate <= 200 ? 38 : rate <= 400 ? 37 : 36));
}
```

圖2.9 Rate Control 程式碼

就這樣在上傳與下載之中，遞迴呼叫 `fork()` 時，我們不斷計算它的現在速度跟目標速率的差別，然後每傳送一個封包就進入 `rate control` 控制他現在的速度。

```
// Rate Control
data_sum += byte_num;

gettimeofday(&t, NULL);
cur_moment = t.tv_sec * 1000.0 + t.tv_usec / 1000.0;

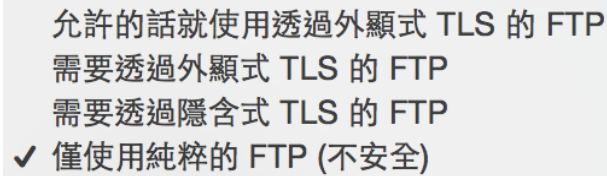
current_rate = (1.0*data_sum) / (cur_moment - prev_moment);
rate_control(byte_num, down_rate);
```

圖2.10 proxy_func - rate_control 程式碼

IV. 問題解決

A. 速度暴衝

我們發現剛開始測試時，速度完全沒有被控制下來，結果發現是因為我們在 Filezilla 連線時沒有將設定調整正確，讓系統自己走別條路而不使用我們的 FTP_Proxy。

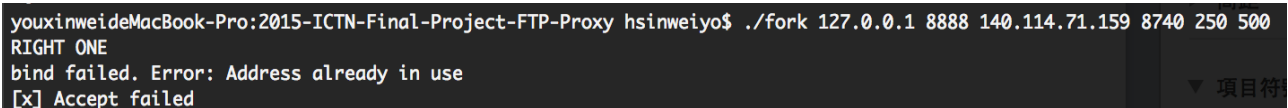


允許的話就使用透過外顯式 TLS 的 FTP
需要透過外顯式 TLS 的 FTP
需要透過隱含式 TLS 的 FTP
✓ 僅使用純粹的 FTP (不安全)

圖4.1 Filezilla 站台管理員設定

B. Address already in use

這個問題尚未找到解決辦法，但是我們知道這是 sample code 的 bug，因為在 client 真正 closed 之前我們就 bind 了，才會造成他認為這個位址已經有人使用了，這個問題早成我們 trace code 時很大的問題，因為每關掉一次就要按很多次關閉，甚至不知道原因出在哪裡，現在我們就只能利用每次都使用不同的 Port，來避免這個問題困擾我們。



```
youxinweideMacBook-Pro:2015-ICTN-Final-Project-FTP-Proxy hsinweiyo$ ./fork 127.0.0.1 8888 140.114.71.159 8740 250 500  
RIGHT ONE  
bind failed. Error: Address already in use  
[x] Accept failed
```

圖4.2 Address already in use 示意圖

V. 實驗數據

我們在前面提及，我們控制速度的方法是讓他睡一下，但是他乘上的常數所得到的結果差很多，所以我們用窮舉法把所有大小的上傳下載檔案都來拿上傳及下載，分別乘上不同的係數，整理出實驗數據，最後決定我們的係數會隨著使用者所輸入的值而改變。

係數	上傳速限	檔案大小	實驗結果	係數	上傳速限	檔案大小	實驗結果
38	100	5MB	99.97	39	100	5MB	99.97
		10MB	100.01			10MB	99.95
		21MB	100.01			21MB	99.95

係數	上傳速限	檔案大小	實驗結果	係數	上傳速限	檔案大小	實驗結果
40	100	5MB	99.88	36	500	5MB	500.29
		10MB	99.90			10MB	500.36
		21MB	99.89			21MB	499.78
						81MB	499.71

係數	上傳速限	檔案大小	實驗結果	係數	上傳速限	檔案大小	實驗結果
37	500	5MB	498.78	38	500	5MB	497.28
		10MB	498.08			10MB	498.83
		21MB	499.59			21MB	499.02
		81MB	499.47			81MB	499.08

係數	上傳速限	檔案大小	實驗結果	係數	上傳速限	檔案大小	實驗結果
39	500	5MB	500.29	40	500	5MB	498.78
		10MB	498.83			10MB	498.47
		21MB	498.45			21MB	498.46
		81MB	496.35			81MB	498.53

係數	下傳速限	檔案大小	實驗結果	係數	下傳速限	檔案大小	實驗結果
39	50	5MB	51.87	40	50	5MB	51.71
		10MB	50.84			10MB	50.81
		21MB	50.44			21MB	50.41

係數	下傳速限	檔案大小	實驗結果	係數	下傳速限	檔案大小	實驗結果
41	50	5MB	51.67	36	250	5MB	260.07
		10MB	50.84			10MB	255.21
		21MB	50.38			21MB	252.17
						81MB	252.37

係數	下傳速限	檔案大小	實驗結果	係數	下傳速限	檔案大小	實驗結果
43	50	5MB	51.67	42	50	5MB	51.62
		10MB	50.69			10MB	50.82
		21MB	50.34			21MB	50.30

係數	下傳速限	檔案大小	實驗結果	係數	下傳速限	檔案大小	實驗結果
37	250	5MB	259.24	38	250	5MB	259.45
		10MB	255.91			10MB	253.15
		21MB	252.37			21MB	252.12
		81MB	250.52			81MB	245.18

係數	下傳速限	檔案大小	實驗結果	係數	下傳速限	檔案大小	實驗結果
39	250	5MB	259.24	40	250	5MB	259.66
		10MB	254.82			10MB	254.52
		21MB	252.08			21MB	251.83
		81MB	250.14			81MB	250.04

VI. 工作分配

其實因為在寫出控制速度的程式前必須理解報告中所敘述的所有函式，所以大部分的分工都是一起完成的，下列表單僅列出相對負責的工作領域，這是一個團隊合作的 Final project。

表6.1 工作分配表

工作項目	工作分配
程式碼追溯	林子馨，游欣為
程式碼除錯	徐唯瑄
實驗數據搜集	游欣為，林子馨
程式碼實作	游欣為，徐唯瑄
函式功能理解	林子馨
程式碼整合	林子馨，徐唯瑄
報告撰寫	游欣為