1. Architecture of the project. (ex: Flow of program)

我們這次的程式，主要使用進程的方式來處理。

proxy 開啟後，程式就會開啟一個 socket 來 listen，當每次有 client 欲跟 proxy 建立 connect 的時候，我們就建一個子進程來處理。每一個子進程都會分別跟 client 以及 server 建立 socket 連線。

也就是 proxy 是 client 端的過渡 server(create_server)，是 server 端的過渡 client(connect_FTP)。

2. Trace Code: The understanding of the proxy

一般的 proxy，是指 proxy server 在接受使用者的 request 之後，會先檢查自己的 server 上有沒有一份 client 端要的資料(可快取)，如果沒有，則 proxy 會代理 client 端，去截取一份需要的資料，這份資料除了給 client 端之外，proxy server 這也存放一份。而同樣的下一個 client 端使用者來做 request 時，proxy server 便會一樣先在 server 中檢查看看，如果有的話，則再檢查與目的端的資料是否相符，若相符則由 proxy server 直接給要求的 client 端即可。**簡言之，類似建立個哨站觀察記錄操作封包**。

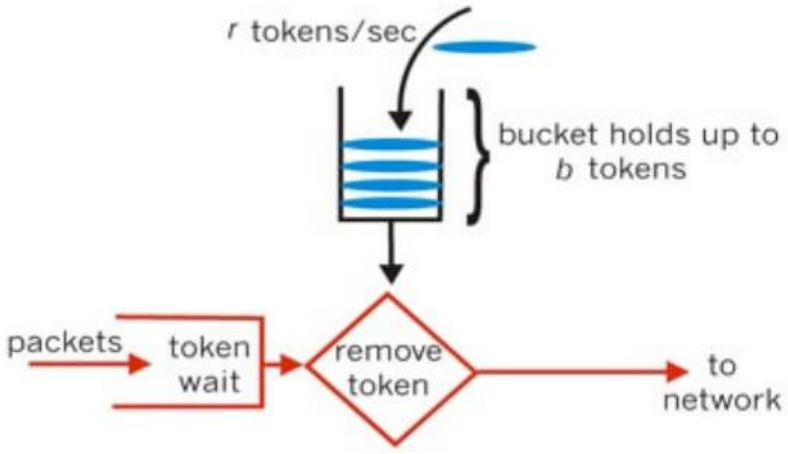
而我們這也差不多，主要就是建立個 client 端的過渡 server(create_server)，是 server 端的過渡 client(connect_FTP)，這快取不是我們的目的，而是在中間觀察其流量，封包數及消耗時間，近來郎

幫助我們方便實現接下來的速度限制。

3. How do you implement the approach of controlling transmission rate?

這次的程式我們**原本要用漏桶算法外加一些速率系數調變**(在此我們叫做**波浪法)**，後來在低速上傳上**遇到極容易 timeout 的問題**，調整後**有其改善的極限**(此部分留到 Problems you confronted and how did you solve them? 細講)，後來，**我們主要是以令牌桶演算法外加波浪法來控制傳輸速度**。

令牌桶演算法的原理是系統會以一個恆定的速度往桶裡放入令牌，而如果請求需要被處理，則需要先從桶裡獲取一個令牌，當桶裡沒有令牌可取時，則拒絕服務。



實作方式演算法——令牌法：

①token generation：每个 1/Rate 時間往 token bucket 中放入一个 token（token 也就是傳輸的最小單位）

②packet queue：建立一個 queue 來暫時儲存要傳送的 packet，當

packet 傳入 proxy 後，如果這 queue 尚未放滿，我們便將資料暫時保存在 queue 裡，若 queue 已經放滿，那我們暫且就不接收這筆資料。

③通過消耗 token 來傳輸資料：每次從 packet queue 的 front 取出 data，如果此時 token bucket 中 token 的數量比 packet 的大小大（由速度單位決定），那我們就消耗與 packet 大小相對應大小的 token 數量，若沒有比較大，那我們就繼續等待 token 增長到足夠的大小，不轉送 data。

④packet queue：每個 socket，在建立完 TCP 連線後，都會有兩個緩存區(也就是前面所說的 packet queue)，分別為傳送和接收。也就是指，在每個 proxy 子進程中，會有 4 個緩存區。

- client 端

  接收緩存區：先從 client 將資料緩存至此，再從此處讀取數據

  傳送緩存區：先將資料緩存至此，再從此處傳送數據至 client

- server 端

  接收緩存區：先從 server 將資料緩存至此，再從此處讀取數據

  傳送緩存區：先將資料緩存至此，再從此處傳送數據至 server

藉由這個緩存區的方式，我們傳輸資料時，需要經過的步驟為：

從接收緩存區讀取數據→消耗 token→傳送到另一端的傳送緩存區

程式內實作方式：

(1) 首先，使用 struct 來建立與 token 相關的數值。

```
struct token {//create a token data type
    pthread_mutex_t mutex;//Mutual exclusion
    int count;
    int count_loop;//the total transmit flag
    int rate;
    int token_per_time;
    bool download;//download or upload
    double t; // delay time for generating/consuming token
};
```

(2) 因為 token 的產生需要與主程式（也就是傳輸檔案）一起運行，所以我們使用 Linux 有的 pthread 來實作。在每個進程裡，都會創建一個線程來產生 token。創建 thread 的方法如下，在 proxy_func 這個函數中實作。

```
// tokens gets the messages of upload and download and count_loop by initializing
struct token *proxy_token = (struct token *)malloc(sizeof(struct token));
proxy_token->rate = rate*1024;
proxy_token->count = 0;
proxy_token->count_loop = count_loop;
proxy_token->download = download;
// to balance the speed error between consume and generate tokens
int setval =  proxy_token->rate;
if (proxy_token->rate >= (MAXSIZE*balance_factor)){
    proxy_token->token_per_time =MAXSIZE;
    setval =MAXSIZE;
}
else{
    proxy_token->token_per_time =  (proxy_token->rate/balance_factor);
}
proxy_token->t = ((double)(proxy_token->token_per_time)/proxy_token->rate);

// start a pthread to generate token
pthread_mutex_init(&proxy_token->mutex,NULL);
pthread_t pthread_id;
pthread_attr_t attribute;
pthread_attr_init(&attribute);
```

(3) 因為在使用 multi-thread 的過程中，會默認進程所產生的 thread 與原本的 thread 是互相阻塞的關係，為了避免這個情況，我們使用

pthread_detach 這個函式，將 thread 及 thread 間分離。

(4) 控制速率

在此**令牌桶演算法是控制大局速度**不要跟實際差太多，為了更精確控制速率我們**外加兩個演算法，原本在漏桶算法的波浪法以及低速強制規定法**

- 波浪法:是根據觀察總流量而制定現在理論速度跟實際速度所要放慢的時間應該要在乘上因當前已傳送的總流量而調變的變速，進而可以更有效使小檔案或過慢或過快的限速達到理論產生的速度更貼近實際速度，此方法有一好一壞，**好處是很有彈性地去修正偏離的速度，壞處就是要花大量的時間去試出 hyperparameter**。

```
int usleep_time = (int)1000000*(proxy_token->t);↓
if (download==true){//initialize download slower↓
    if ( count_loop>1000 && count_loop<=4000){//initialize download slower↓
        usleep_time = (int)1300000*(proxy_token->t);↓
    }↓
}↓

if (download==false){//initialize upload faster↓
    if ( count_loop>0 && count_loop<=2000){//initialize upload faster↓
        usleep_time = (int)1100000*(proxy_token->t);↓
    }↓
    if ( count_loop>2000 && count_loop<=5000){//initialize upload faster faster↓
        usleep_time = (int)500000*(proxy_token->t);↓
    }↓
    if ( count_loop>5000 && count_loop<=7000){//initialize upload faster faster faster↓
        usleep_time = (int)10000*(proxy_token->t);↓
    }↓
}↓
```

- **低速強制規定法**:這個法**主要是對應魔王測資 25KB/sec 上傳而生**，主要是把 55 以下速度先劃分幾個速度區間進而各自有其新規

定速度，之所以那麼做是因為我們發覺有些低速我們已經調的蠻穩定了。可是偏離一段固定的速度差(有 bias)，**那乾脆點，你偏離那固定一段速度差我就修正回一段速度差就誤差及我們故意的速度差互相抵銷了**。

```
port = atoi(argv[2]);
rate = atoi(argv[3]);
//low speed needs to be record and fix bias via experience
if (rate > 20 && rate <= 30) {
rate=28;
}
if (rate > 30 && rate <= 40) {
rate=35;
}
if (rate > 48 && rate <= 55) {
rate=55;
}
```

(5) 為了將 token 演算法應用在上傳與下載，我們使用 select 函數。根據 IO 狀態修改 fd_set 的內容，來通知執行了 select()的進程，哪一個 socket 或文件，是可讀取或可寫入的。**此外我們也因應速率調整不單是 rate_control 有去做速率調整，連 token 在進行線程內部也有去實現波浪法**。

4. Problems you confronted and how did you solve them?

原先我們打算使用**漏桶方法外加波浪法**，直接在每次傳輸速度時判斷它的傳輸速度，再以調變 delay 延遲時間來達到限制傳輸速度的方式。但這個方式**會遇到 filezilla 的 timeout 問題，波浪法有改善但有極限**。

因此**後來我們改為使用令牌桶演算法來進行傳輸，原本遇到的傳輸問題就能解決一半**，雖然聽說別組令牌很準，架起來後沒出過這問題，也許是我們架起本身的令牌桶演算法並不完全但可運行，**後來想想可以沿用之前的波浪法，果然解決幾乎所有 timeout 問題**，大概 50KB/sec 以下偶而還是有，並且速度跟理想都很固定差一段速度差(bias)，所以就外**加強制低速規定演算法修正(bias)，不但速度更準 timeout 問題也消失了超爽的**。

但是高速(125KB/sec 以上)時速度仍不太穩定，同種設定下時準時不準，由於我們 project 起步晚(大約兩星期前) ，**然後大約一星期前才從漏桶方法改令牌桶演算法，就只好之後心有餘力再改善了**。

5. How to run your code?

先輸入：**gcc –pthread 31.c -o proxy** 形成執行檔

再執行，輸入：./proxy <ProxyIP> <ProxyPort> <Rate>，Rate 可用來設置上傳/下載速度(kB/sec)

輸入格式如：**./proxy 127.0.0.1 8888 100**

如需修改速度，請一定要先關掉 filezilla，再以 ctrl+c 終止程式後重新輸入新速度即可。

6. Show some experimental results.

下載速率:

大小 / 速度(KB/sec)	3MB	5MB	10MB	20MB	40MB
125	149.3	145.42	128.58	128.14	122.8
100	113.56	112.71	116.98	98.12	97.63
75	88.15	85.42	76.33	75.64	73.28
50	56.07	55.56	52.42	52.37	48.61
25	29.07	25.06	25.66	24.91	23.56

上傳速率:

大小 / 速度(KB/sec)	3MB	5MB	10MB	20MB	40MB
125	119.74	118.73	120.78	121.3	119.75
100	97.62	97.94	98.03	97.88	97.68
75	71.99	68.78	72.02	73.64	72.5
50	39.89	47.86	47.78	51.21	49.38
25	24.01	27.11	27.61	24.91	23.56

7. What is the responsibility of each member?

- 李明穎:架起基礎令牌桶資料型態(struct)，debug 及把基礎的令牌桶演算法上加入觀察總封包數及波浪法調變來處理低速問題。
- 林佳艾: 用基礎令牌桶資料型態去架起基礎令牌桶演算法，debug 及觀察速度回報，結報撰寫。
- 李沛倫:強制低速規定演算法來處理低速問題，debug 及觀察速度回報，結報撰寫。