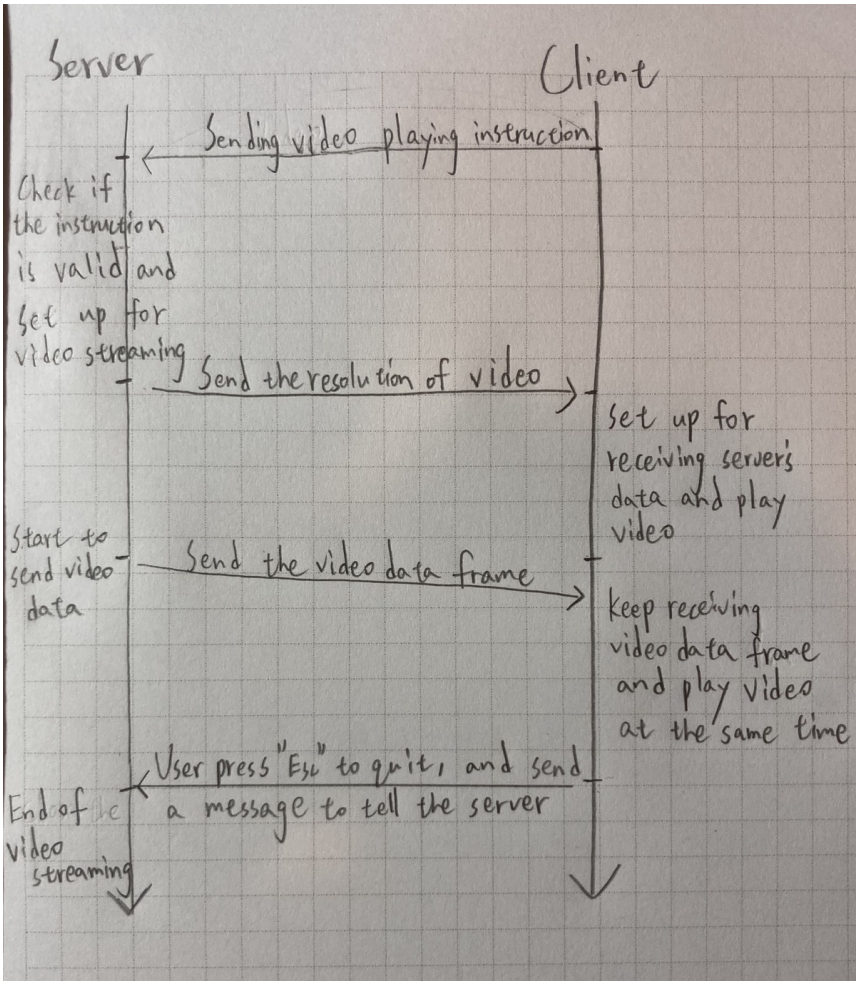
Computer Network HW2 Report

b07902054 資工三 林子權

Draw a flowchart of the video streaming and explains how it works in detail



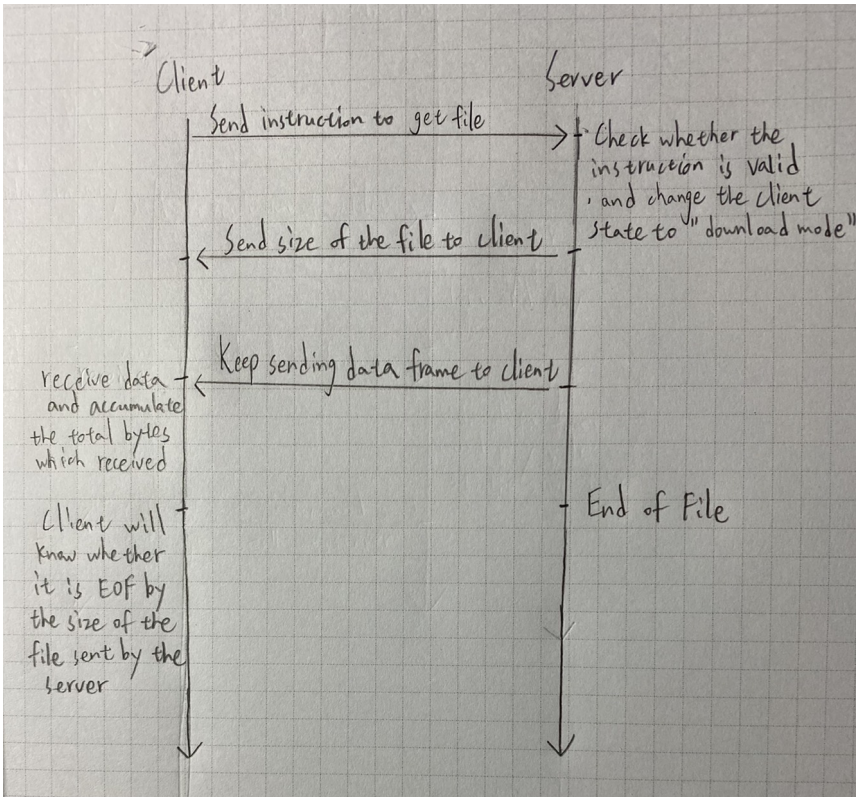
Server接收來自client的video playing instruction，並檢查該instruction是否invalid，如果file path不存在或是檔案不是mpg file，就告訴client這個指令非合法。如果valid，則初始化傳送video data frame會用到的東西，並且把video resolution傳給client，讓client初始化client端的資料。接著server便開始傳送video data frame給client，client一邊receive這些data frame，一邊拿已經收到的data播放影片。client若是想結束影片必須按下ESC。

在server端，我把每個clients初始化OpenCV傳送影片會用到的東西都buffering起來，然後用select每一次while迴圈只傳送一個video data frame，如此一來就可以實現許多個clients同時播放影片的需求，並且達到公平的分配網路資源。

在client端，我把downloading video frames跟playing video這兩個工作放在不同的thread上面去執行。因為downloading的速度會快很多，所以我把多下載下來的frame放在buffer裡面，等待被播放。播放影片時做buffering，除了可以讓影片播的更流暢之外，如果網路突然出了問題，還可以用之前預先下載下來的frame撐著。

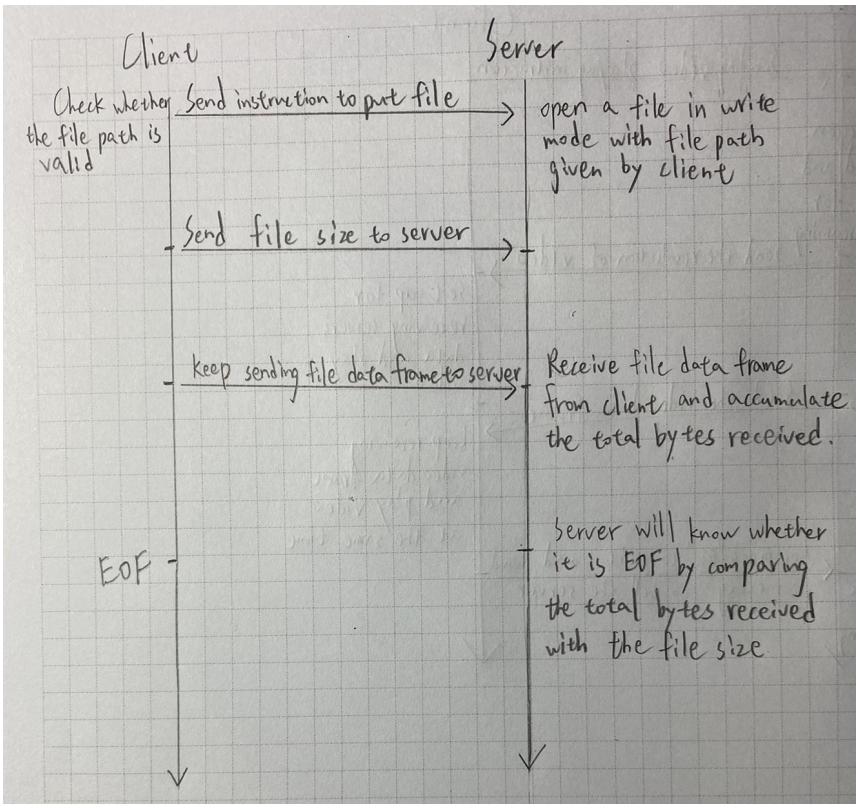
Draw a flowchart of the file transferring and explains how it works in detail

Get file from server:



Client發送get file的指令給server，server檢查該指令是否valid，如果valid的話就先把file size傳給client，再開始傳file data frame給client；invalid的話就發訊息告訴client這個指令不合法。client會持續記錄自己總共收到了多少bytes的data，跟file size比對，如果傳完了就可以繼續發下一個指令給server。

Put file to server:



Client自行檢查put file指令中的file path有沒有存在於client folder內，有的話則發送指令給server，server收到指令後在server folder中創一個檔名一樣的file。接著client會再發送file size給server。server收到了file size之後，便開始接收client傳過來的file data frame。server會持續記錄自己總共收到了多少bytes的data，跟file size比對，如果傳完了client就可以繼續發下一條指令給server。

What is SIGPIPE? Is it possible to happen to your code? If so, how do you handle it?

如果client和server的連線中斷了，server還嘗試要對client的socket做讀或者寫的話，那就會收到SIGPIPE的signal。在我的程式裡面基本上不會遇到這樣的問題，因為我每次迴圈都會判斷是不是有clients連線中斷了，如果有的話要把他們的socket關起來以免不小心對他們做了讀或者寫。判斷的方式為：如果一個client在read_set裡面，但在fetch instruction時recv回傳了0，則代表該client已經斷開連線了。

Is blocking I/O equal to synchronized I/O? Please give me some examples to explain it.

不完全一樣，可以說blocking I/O是synchronized I/O的一種表現形式。
更精確地說，synchronized I/O可以是blocking也可以是non-blocking的。
synchronized I/O在執行I/O operation時會將process給block住，這邊I/O operation的定義是真實的I/O操作(物理上)。在non-blocking I/O中，process會一直重複地確認他要的東西好了沒，若是還

沒好就先不管他，若是好了的話，process還是必須呼叫一個system call，把data從kernel挪回來，此時的這個挪移資料的步驟就必須是synchronized的了。因此non-blocking I/O其實算是一種synchronized I/O。synchronized I/O是process層面的，代表process會被block住直到資料準備好；blocking I/O是kernel層面的，代表kernel不會立即return結果給process。