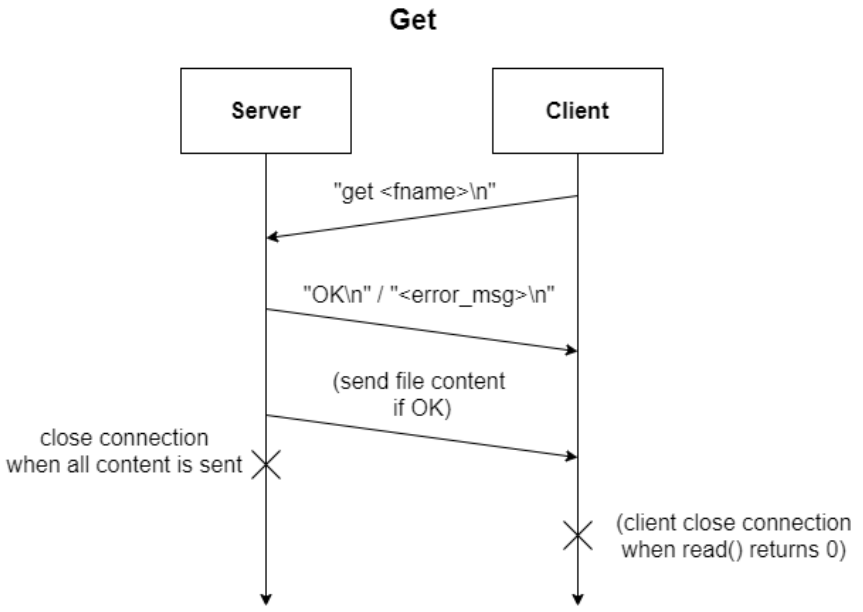
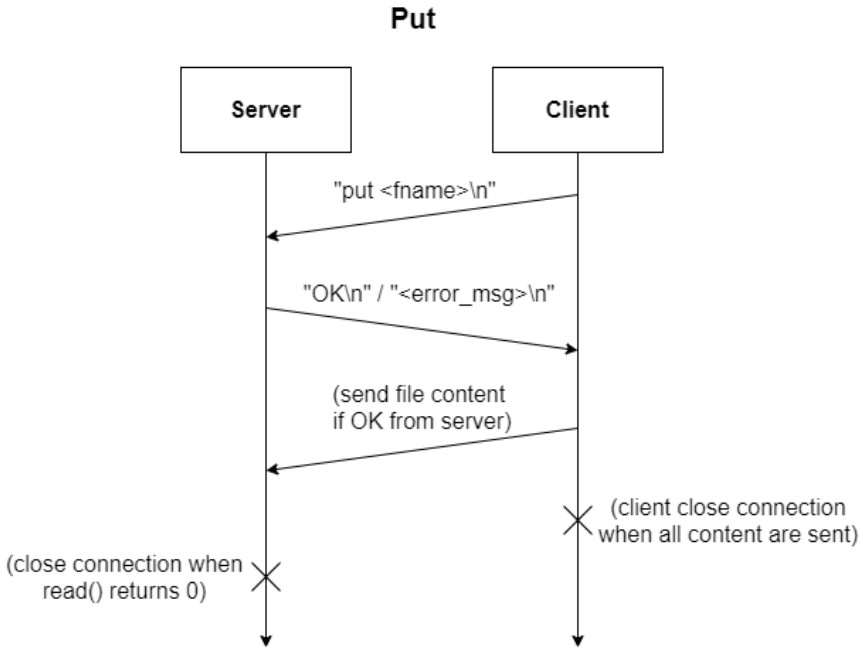
計算機網路 HW2

1.





首先，我把SIGPIPE直接設SIG_IGN，所以write出問題只會回傳-1而不會自爆。

最剛開始在傳訊息的時候，都以 `\n` 當作end of line決定訊息長度。
Client會傳argument給server 。在put/get的情況，client會把所有檔案拆分成一個個形式 `get/put <fname>` 的command，一個一個檔案分別跟server拿 / 放。i.e. 每個檔案都重開一個socket跟server連線。

Server那邊如果回傳 `"OK\n"` 的話才會繼續傳檔案，如果是任何其他東西的話client會當作那是錯誤訊息，然後轉貼給stdout並終止連線。Server傳完error message後也會自己關掉socket。這樣的好處是：

- 可以把Server端發生的錯誤給client看，e.g. `File '<fname>' does not exist.` 之類的。

- 可以更動一下 `OK` 的內容，順便傳一些資訊給client，等等play會用到

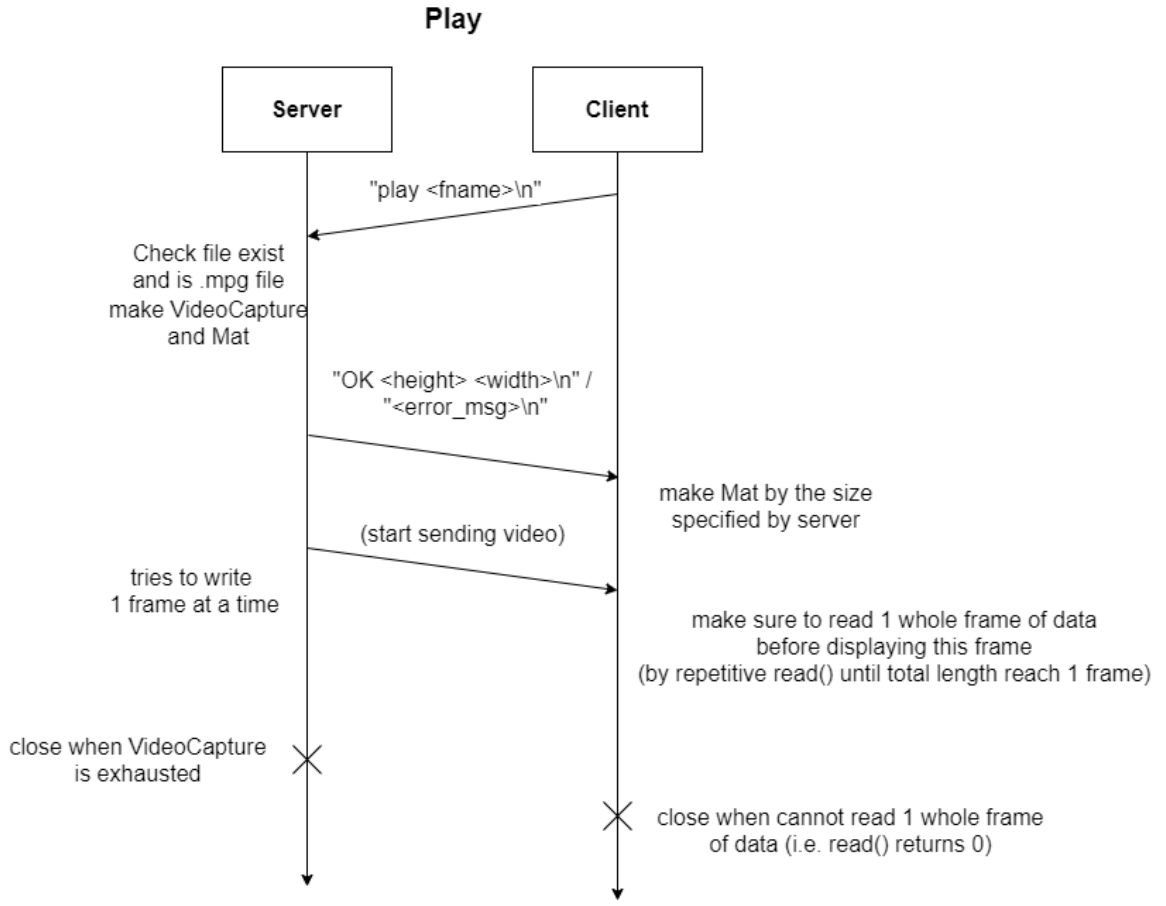
注意如果是error message的話就會關閉連線。接下來的事情都是假設server回傳 `OK\n` 。

接著他們會開始傳檔案 / 接檔案。Client會拼命 `read() / write()` ，而server因為我選擇用blocking IO (`+ select()`)，所以他每次 `read() / write()` 都只會讀/寫一個小 `BUFF_SIZE` 的資料而已，以免其他 `select()` 到的file descriptor沒辦法及時處理到。照理來說這樣應該不會有太多delay，因為 `select()` 到代表那個file descriptor馬上可以讀寫。

如果server寫完了會自己關socket，client會 `read() == 0` 然後也會關socket；如果client寫完關socket，那server會 `select()` 到這個socket並且發現 `read() == 0` ，也會關socket。

如果中間 `read() / write()` 出任何問題 (i.e. 回傳-1或0，可能是未知的錯誤或是client被關掉等等)，那直接close掉相關file descriptor並關掉連線。

2.



Client下達指令，server確定檔案沒問題之後會回傳OK以及影片的width / height。Client收到之後會做對應大小的 `cv::Mat` 。

接著server也是每次select都只試著傳一個frame，而client會試著 `read()` 資料，等累積到一個frame大小的資料後才會顯示。

等到server傳完之後自己close socket，client看到 `read() == 0` 之後也會自己close socket + destroy window。

至於如果client出問題的話，server `write()` 的時候自然會回傳-1。

3.

SIGPIPE是當你write到一個沒有reader的pipe / socket時會出現的signal，同時write回傳-1且errno設成EPIPE。

在我的程式裡如果沒有特別處裡的話的確可能會被送SIGPIPE，因為 `select()` 的時間和真正 `write()` 的時間有差異，導致如果對方在這段時間斷掉的話， `write()` 就會寫到沒有reader的socket。不論是client或server都有可能發生。

我handle的方法就是開頭就 `signal(SIGPIPE, SIG_IGN)` ，因為如果真的發生的話， `write()` 會回傳-1，原本就已經出錯該斷掉連線了，是因為SIGPIPE還是因為其他原因已經不太重要了。

4.

ref.

- https://stackoverflow.com/questions/8416874/whats-the-differences-between-blocking-with-synchronous-nonblocking-and-asynch

- Blocking: 一切(i.e. 整個thread)會停止，直到某件事完成才繼續。

- Synchronous: 等到對方回應後才會繼續。

他算是兩種東西：blocking指的是「某個動作會block整個thread的運行」的事實，而synchronous單純指「這個動作不做完的話不會做下一步」。

e.g. 如果一個server有一個socket，接著一個client。現在假設client想傳東西給server，那為了達到synchronous，我的socket可以是blocking或是nonblocking：

- blocking：單純讀，他會block，等到返回時我已經有東西了。
- nonblocking：我可以busy waiting，一直嘗試 `read()` 他直到不噴錯為止 (i.e. `read() != -1` , `errno != EAGAIN` 等等)。

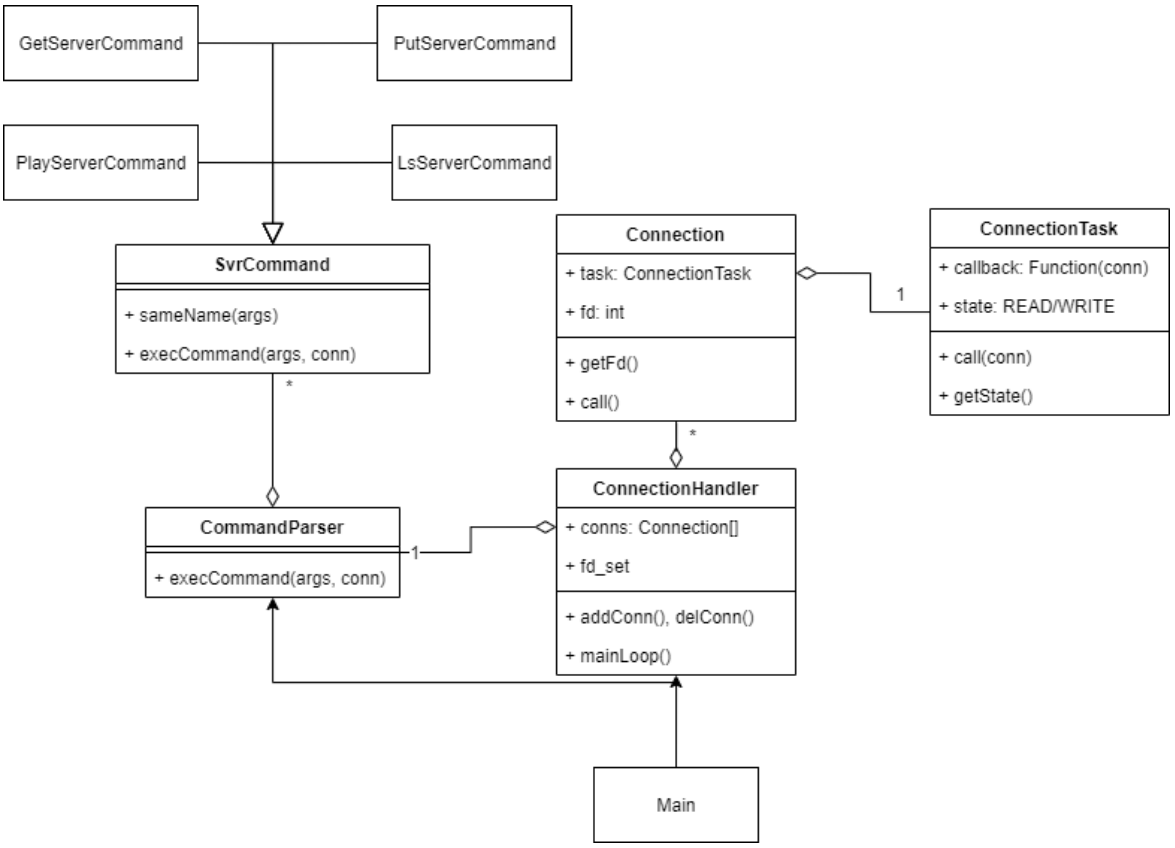
我還是會有synchronous的效果 (因為不讀到我的流程就不繼續走下去)，但是我可以用blocking或nonblocking socket。

所以他們兩個還是有差別。Blocking基本上代表synchronous，但是synchronous也可以用nonblocking做到 / 模擬blocking的效果。

原本還以為第1, 2題要解釋整個程式架構，不過事後發現只要講protocal即可。因為我已經打好了所以在這邊一併附上。

我用了C++，因為我需要OOP。

Server的架構大致如下：



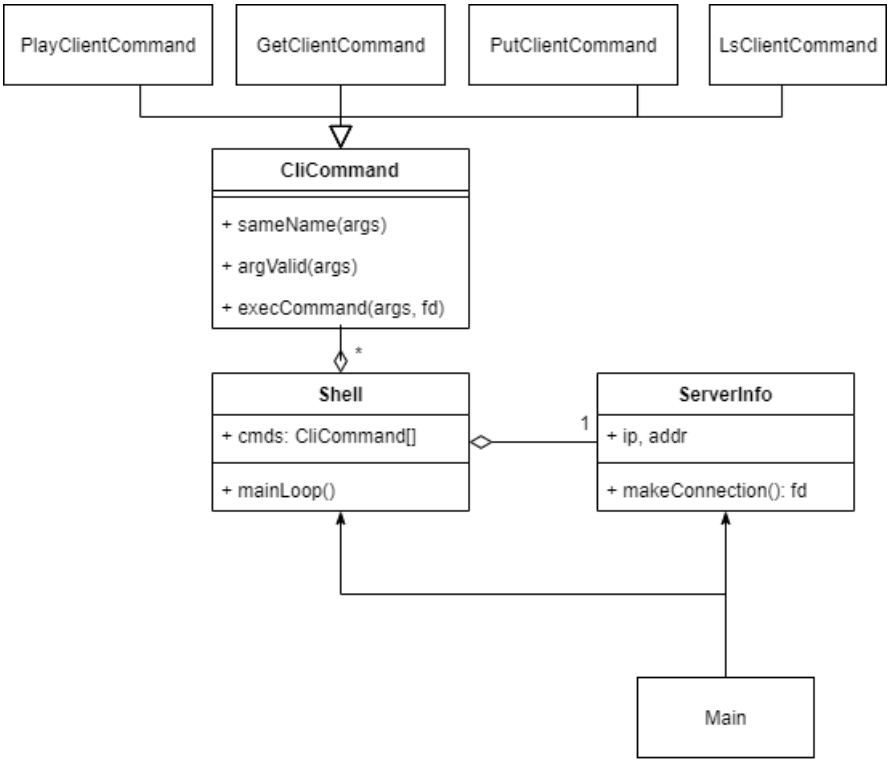
- `Connection`
  - `Connection` 為我們操作上的連線基本單位，可以說是file descriptor的再包裝，交由 `ConnectionHandler` 管理。
- `ConnectionTask`
  - 因為我們要用 `select()` ，代表我們需要每個 `Connection` 知道 `select()` 到他時需要做什麼事。
  - 他負責表示：
    - 輪到這個 `Connection` 被執行的時候，要call哪個function？這個function不能block太久，只能收發一點點東西 (e.g. 1024 bytes或一個video frame) 就要return
    - 目前這個function是要read還是write？這會影響到 `select()` 的時候要放到read還是write的 `fd_set` 。
  - 至於目前傳輸進度以及資料等等的執行state是存在function裡面。i.e. 用lambda function的話需要指定 `mutable` 等等。
- `ConnectionHandler`
  - 會用 `select()` 看所有 `Connection` 的file descriptor能不能被read / write。如果能的話，會用 `Connection::call()` 交由此 `Connection` 裡的 `ConnectionTask` 決定他現在要做甚麼。
  - 所有 `Connection` 的file descriptor由 `ConnectionHandler` 負責關閉， `Connection` 可以透過 `ConnectionTask` 的function的回傳值(bool)告訴他要不要把本 `Connection` 結束掉。
  - 有 `mainLoop` 給main負責拼命 `select()` 可以收發的file descriptor。
- `SvrCommand, CommandParser`
  - 所有種類的command (e.g. ls, get, ...) 都是 `SvrCommand` ，在main裡面被dependency inject進去 `CommandParser` 。 `CommandParser` 負責透過command arguments `args` 的第一個argument來決定要用哪一個command。

    - Argument需要透過client傳過來。 `SvrCommand` 會需要更動 `Connection` 的function和state來指定他要做的任務。

這個架構代表：

- 我們可以隨意定義及增加 `SvrCommand` 並且在main時dependency inject進去我們的code裡面。
- 我們可以在剛開始做一個特別的 `Connection` 和 `ConnectionTask` 來接收socket連線，他的callback function負責幾件事情：
    - `accept()` 以及把新的file descriptor變成一個新的 `Connection` 。
    - 指定新的 `Connection` 的 `ConnectionTask` 的function為「讀取argument並透過 `CommandParser` 與 `SvrCommand` 指定自己新的function」。
    - 把新的 `Connection` 交由 `ConnectionHandler` 來處理 `select()` 事項以及統一管理。
- `SvrCommand` 的 `execCommand(args, conn)` 基本上就是給此connection一個新的function來做事。注意不能在 `execCommand` 裡面做真正的read / write，畢竟現在其實是正在block住負責連線的socket的，必須快點結束。

Client：



- `CliCommand`
    - 事實上跟 `SvrCommand` 差不多，不過因為我們沒有多連線要管理，他們可以接file descriptor並且真的做完工作時才return，也就是client並沒有 `Connection` 以及callback function的機制。
- `ServerInfo`
    - 表示server的資訊以及負責建立跟server的連線。
- `Shell`
    - 有點像server的 `CommandParser` ，不過有 `mainLoop()` 負責接收指令。