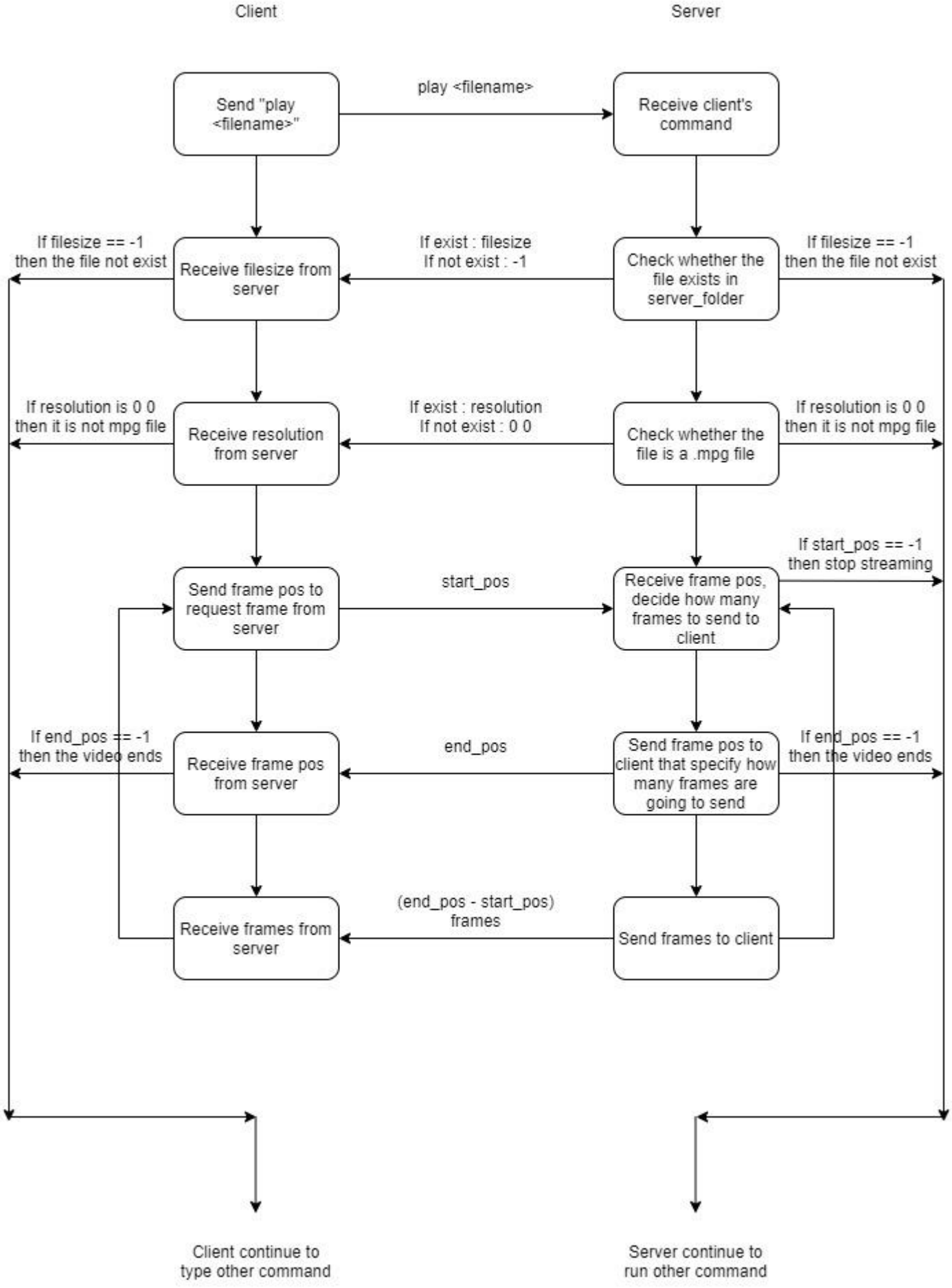
Computer Network HW2 report B07902116 資工三 陳富春

(1) Video streaming



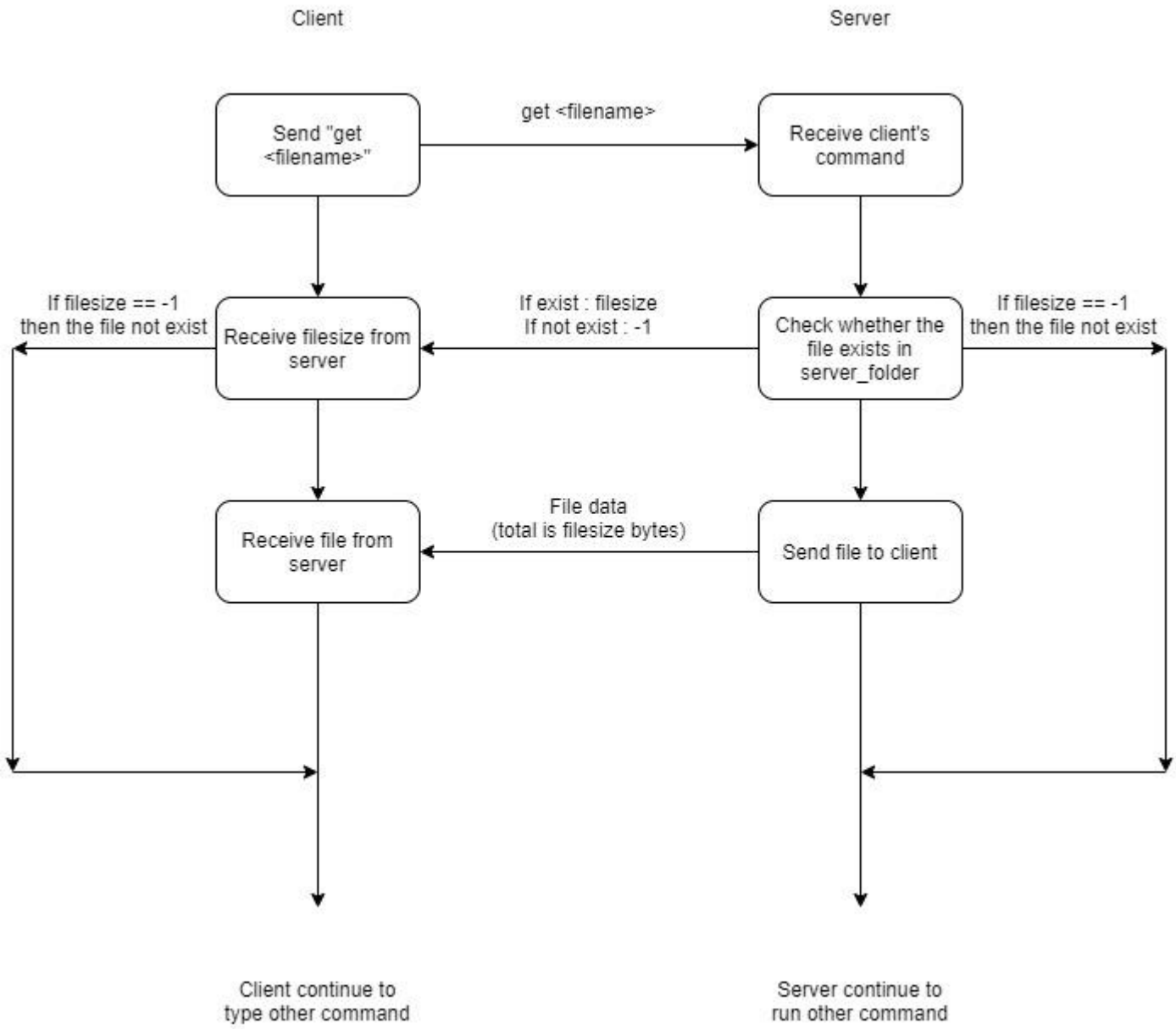
Client sends command to server, then server check whether the file exist.
If the file does not exist, server sends “-1” to notify client that there is no such file in server_folder.
If the file exists, server sends filesize to client and then check whether the file is a mpg file.
If the file is not a mpg file, server sends “0 0” to notify client that the file is not a mpg file.
If the file is a mpg file, server sends resolution to client.

After receiving the resolution, client will send start_pos to tell server from where to read the video’s frames, then server will send end_pos to tell client where the frames will play to and send (end_pos – start_pos) frames to client.
When client finish playing received frames, it will send another start_pos to server to continue play. If server noticed that the start_pos exceeds total frame number, it then send “-1” to notify client that the video ends. And then both client and server finish their play command.
If client wants to stop streaming in the middle, then it can press ESC to stop. Client will send “-1” to server to notify server that the client wants to stop streaming, and then client will flush the socket buffer by simply receiving those yet-to-play frames.

By letting server only send some frames to client enables server to response to other client simultaneously, and thus achieve the function of multiple connection.

(2) File transferring

Get:



Client sends command to server, then server check whether the file exist.
If the file does not exist, server sends “-1” to notify client that there is no such file in server_folder.
If the file exists, server sends filesize to client.

After sending filesize, server starts to send file data to client and client starts to receive file data. Because both client and server know how many bytes to receive/send, they know when to stop receiving/sending data.

By letting server only send some file data to client enables server to response to other client simultaneously, and thus achieve the function of multiple connection.

put is similar to get except that it is client that needs to check whether the file exists or not and file data flow from client to server.

(3) When client tries to write to a socket that is connected to server, but the socket in server is already closed, client will receive a RST packet. If client tries to write to the socket closed again, client will receive SIGPIPE.

If client and server executed normally, that is, client only terminates after user type “exit” and server does not receive any interrupt signal, then there will be no SIGPIPE. Because both client and server know how many bytes are going to send/receive, the connection will not close in the middle of transmission.

I did not implement a signal handler for SIGPIPE, so when SIGPIPE occurs, my process will terminate by default.

(4) Synchronous I/O is not equal to blocking I/O. Synchronous I/O can be blocking or non-blocking.

Blocking I/O means a process will start to wait after call the blocking I/O.

Synchronous I/O means a process will have to wait kernel to finish the I/O operation.

1. Blocking and synchronous I/O : take recv() for example

   When a process call recv(), the process starts to wait no matter whether there is data to receive. Kernel will wait for some data come in and then copy the data into process’s buffer and then return. So the process is blocked since it calls recv() until recv() return.
2. Non-blocking I/O and synchronous I/O : take read() in non-blocking mode for example

   When a process call read(), if the data is not ready yet, read() will return some error message immediately that tell the process no available data now. If the data is ready, then after process call read(), the process starts to wait until kernel copy the data into process’s buffer.

   So the process is blocked when kernel is copying data into process’s buffer but not blocked if the data is not ready.