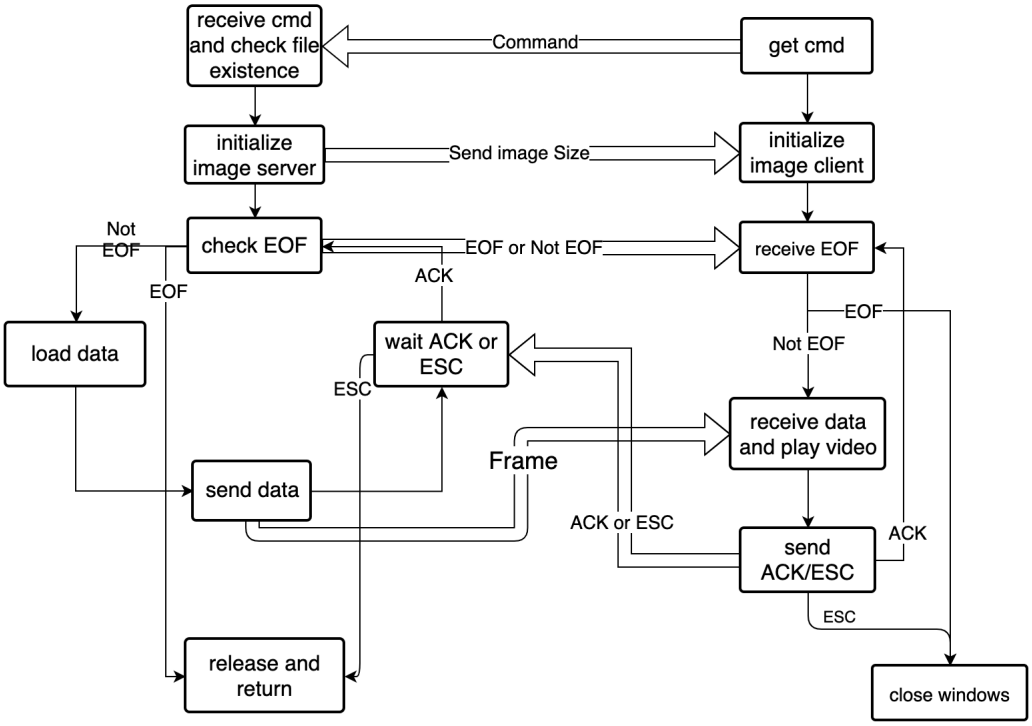
CN homework#2 report

b07902076
資工三 許世儒

*** note that the left side is the server and the right side is the client**

1

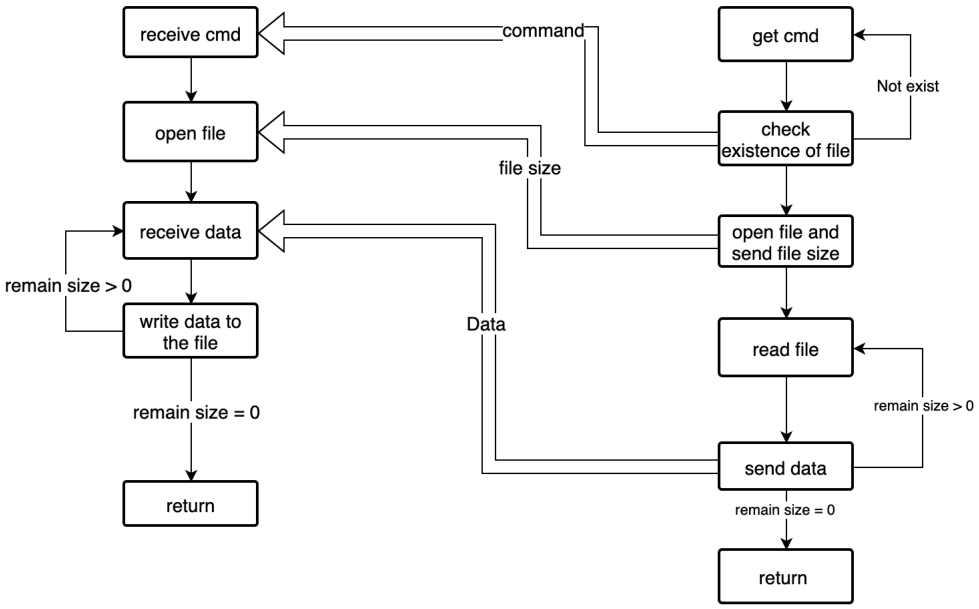


After client receives command, it will send the command to the server. The server will check whether the format is correct and whether file exists. Then, the server will initialize the image server, and the client will initialize the image client at the same time. After that, the server will check whether the file is EOF and sends the state to clients. If it's not, and it will start to copy data from image server and then send to clients. If it's EOF, it will return and stop sending. After sending data to clients, the server will wait for ACK sent from clients. In the meanwhile, clients receive data and show the video and then check whether user presses "ESC" key. If user presses "ESC" key, then the clients will send ESC instead of ACK to the server. Otherwise, the clients will send ACK to the server. If the server

receives ESC, it will stop sending and return, and the client will close the windows and wait for the next command.

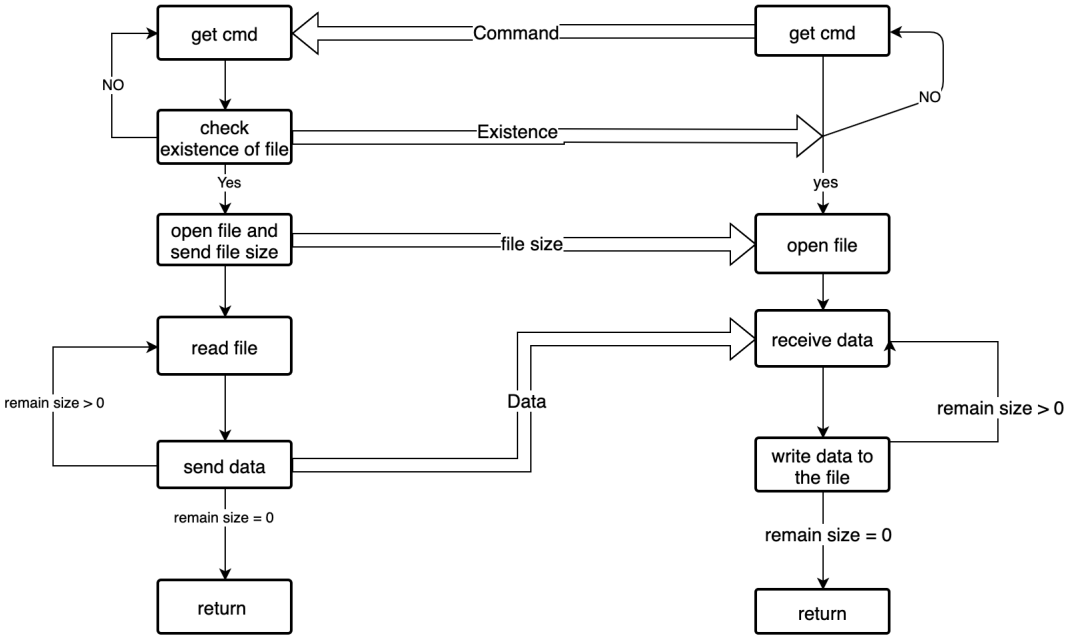
2

PUT:



After getting the command, the client will first check whether the file exists. Then, it will send command to the server. After that, the server will create a new file and the client will open the file and send the file size to the server. The client reads data from the file and sends it to the server until the remaining size equals 0. At the same time, the server receives data from the client until the remaining size equals 0. And then, both client and server return.

GET:



After getting the command, the client will send the command to the server. The server will check whether the file exists and send the result back to the client. If the file exists, the server will open the file and send the file size to the client. At the same time, the client will create a new file. After that, the server will send data until the remaining size equals 0 and the client will receive data. At the end, both client and server will finish it and return.

3

SIGPIPE is a kind of signal which is designed for processes under Unix and Unix-like system. It may happen if client or server is trying to send or write data to a closed socket. It can happen to my code when the server or client send data to the other side which has already closed. For example, in the part of playing video, the server will keep sending frames to the client. However, the client may close the socket during this time. Therefore, the SIGPIPE will happen and the process will be terminated. To solve this problem, I have my program stop receiveing or sending when the return value of send() and recv() is non-positive. At this time, if I send or recv again, the SIGPIPE would happen.

4

Blocking I/O doesn't equal to Synchornous I/O. However, they are the same sometimes. For example, in my homework 2, the function of recv() is blocking I/O. It is because when the function is called, the program will stop until it really receives data. Also, it's also Synchornous I/O. As for the Synchornous I/O instead of Blocking I/O, there is also an example in my homework 2.

```
fcntl(svr.listen_fd, F_SETFL, O_NONBLOCK);
remoteSocket = accept(svr.listen_fd, (struct sockaddr *)&remoteAddr, (socklen_t*)&addrLen);
if (remoteSocket < 0) {
    if (errno == EINTR || errno == EAGAIN) continue;  // try again
    else if (errno == ENFILE) {
        perror("out of file descriptor table\n");
        continue;
    }
```

As we can see, it is a Non-blocking for the function accept(), but it is Synchronous I/O because we will retry until it accepts a remote socket.