計算機網路概論

Lab3

111060013 EECS 26' 劉祐廷

Catalog

1. Self Check……...……P3

2. Code Explanation…..……..…..……P3

3. Output Result……..……..P6

4. What I Have Learned…..P6

1. Self Check

- **Built by the Make tool ---------- done**
- **Use stop-and-wait method ------- done**
- **Write the new file correctly ----- done**

2. Code Explanation

A. server.c: sentFile()

```
// Seek to the "current" position in the file
fseek(fd, current, SEEK_SET);
```

使用 fseek()將檔案的指標移到正確的位址，current 表示已經讀過的檔案大小，在 fseek()第三個參數中填入 SEEK_SET 表示 fseek()會從頭開始數 current 個位址，然後將 fd 指向那個位址。

```
// Read 1024 bytes from the file into the data field of the packet we will send
// fscanf(fd, "%s", send.data);
memset(send.data, 0, sizeof(send.data));
fread(send.data, 1, 1024, fd);
```

使用 fseek()找到要讀取的位址之後，先將 send.data 全部設成 0，這樣在最後一個封包大小小於 1024 的時候，剩下的 bytes 都是 0，接著使用 fread()，從 fd 讀取 1024 個 bytes 並存入 send.data。

```
// Check if the current position indicates that the last packet is to be sent
    // If it is, set the packet size to the remaining bytes in the file
    // Set the isLast flag to true
// Otherwise
    // Set the packet size to 1024
if (filesize - current <= 1024) {
    send.header.size = filesize - current;
    send.header.isLast = true;
}
else {
    send.header.size = 1024;
}
```

當 filesize – current <= 1024 表示這是最後一個封包了，因此要把 send.header.isLast 設成 true，且由於最後一個封包的大小不固定，準確來說是 filesize – current，因此也要對 send.header.size 做一些修正。若不是最後一個封包的話，則將 send.header.size 固定在 1024，send.header.isLast 會預設成 0 因為前面有先用 memset()設定過一次了。

```
// Send the packet to the client
if (sendto(sockfd, &send, sizeof(send), 0, (struct sockaddr *)&clientInfo, sizeof(struct sockaddr_in)) == -1) {
    perror("sendto()");
    exit(EXIT_FAILURE);
}
```

將 send 的所有變數都設定好之後，使用 sendto()將檔案傳送給 client。

```
// Wait for a response from the client using poll(..., TIMEOUT) with the POLLIN event
// Alternatively, set the timeout with setsockopt(..., SO_RCVTIMEO, ...) after creating the socket

struct pollfd pfd;
pfd.fd = sockfd;
pfd.events = POLLIN;
pfd.revents = 0;

if (poll(&pfd, 1, TIMEOUT) <= 0) {
    printf("Timeout! Resend!\n");
    continue;
}
```

使用 poll()實現 stop-and-wait 的機制，將 pfd.events 設成 POLLIN 表示程式會等到有資料可以讀(收到 ack)時才會傳並繼續執行下去，若有錯誤會回傳 0，若超過第三個參數給定的時間(ms)則會回傳 0。

```
if (recvfrom(sockfd, &recv, sizeof(recv), 0, (struct sockaddr *)&clientInfo, (socklen_t *)&addrlen) == -1) {
    perror("recvfrom()");
    exit(EXIT_FAILURE);
}

printf("Received ACK = %u\n", recv.header.ack);
```

若沒超時，使用 recvfrom()收取 client 傳過來的檔案，並將 ACK number 輸出到終端機以便 debug。

```
// Update the current position in the file
current += 1024;

// Update the sequence number
send.header.seq = current / 1024;
```

收到 ack 之後要更新 current 與 send.header.seq。

B. client.c: recvFile()

```
// Receive a packet first, then use isLoss()
// to simulate if it has packet loss
/*printf("%s\n", buffer);
pause();*/
if (recvfrom(sockfd, &packet, sizeof(packet), 0, (struct sockaddr *)&serverInfo, (socklen_t *)&addrlen) == -1) {
    perror("recvfrom()");
    exit(EXIT_FAILURE);
}
```

使用 recvfrom()收取來自 server 的檔案。

```
// Send an acknowledgement for the received packet
Packet ack_packet;
memset(&ack_packet, 0, sizeof(ack_packet));
ack_packet.header.seq = packet.header.seq;
ack_packet.header.isLast = packet.header.isLast;
ack_packet.header.ack = seq;
if (sendto(sockfd, &ack_packet, sizeof(ack_packet), 0, (struct sockaddr *)&serverInfo, sizeof(struct sockaddr_in)) == -1) {
    perror("sendto()");
    exit(EXIT_FAILURE);
}
```

新建一個 ack_packet，並按照 stop-and-wait 的規則設定 ack_packet.header 裡面的各項變數，並用 sendto()將 ack 送回給 server。

```
// Copy the packet data into the buffer
// Use memcpy() instead of strncpy() since the file
// may contain 0x00 (interpreted as a null terminator)
memcpy(buffer + seq * 1024, packet.data, packet.header.size);
```

使用 memcpy()將收到的 packet.data 存入 buffer，其中第一個變數為複製進 buffer 的開頭位址，因為每個 packet 的 payload 大小為 1024，所以低一個變數要設為 buffer + seq * 1024。

```
// Increment the sequence number
seq++;

// If the packet is the last one, break out of the loop
if (packet.header.isLast) break;
```

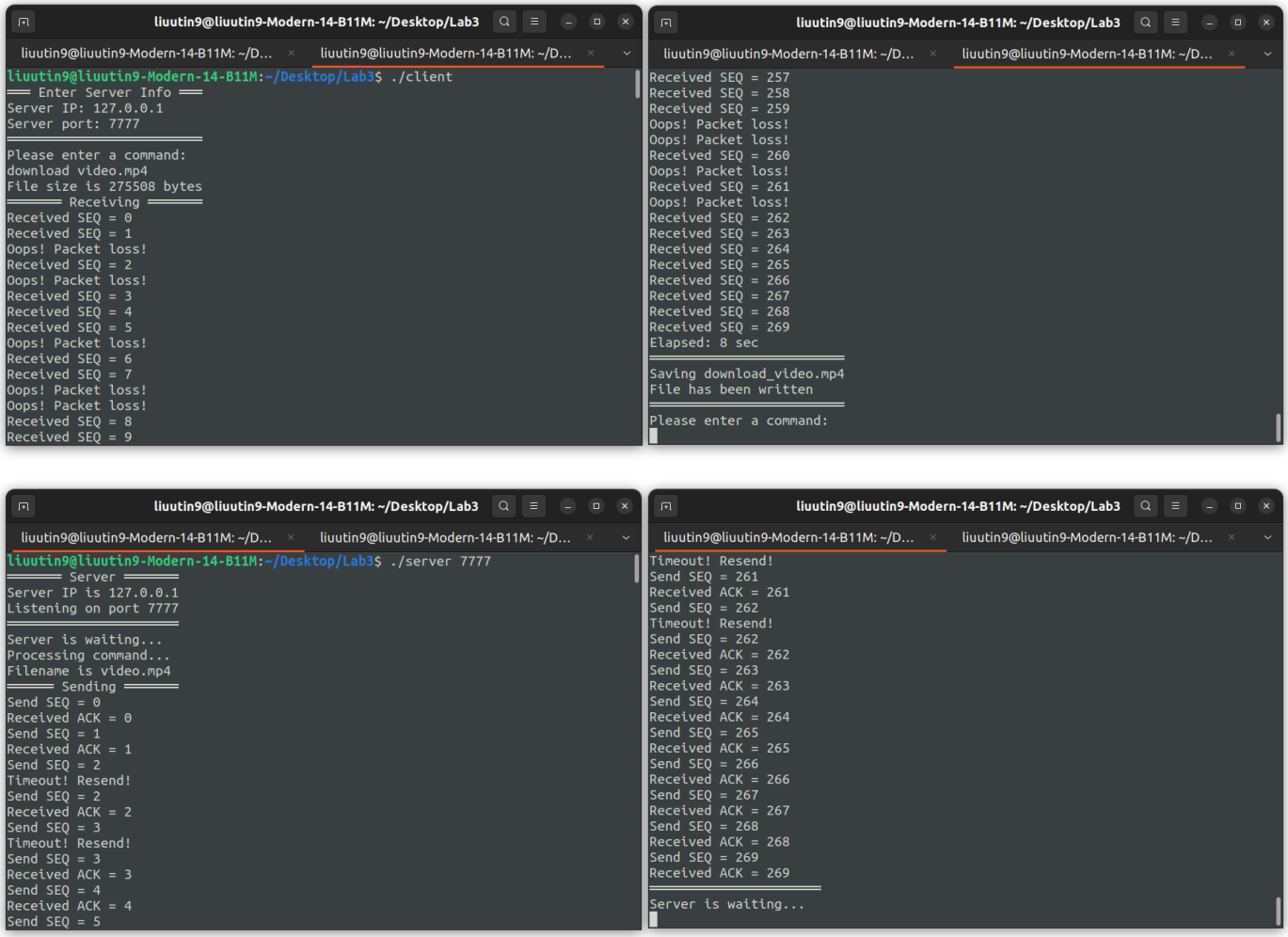
做完所有事情後要更新 seq，如果收到最後一個封包(packet.header.isLast == true)，則要跳出迴圈停止傳輸。

C. client.c: writeFile()

```
// Create a file descriptor
FILE* newFile;

// Name the file as newFilename and open it in write-binary mode
newFile = fopen(newFilename, "wb");

// Write the buffer into the file
// fprintf(newFile, "%b", buffer);
fwrite(buffer, 1, filesize, newFile);

// Close the file descriptor
fclose(newFile);

// Set the file descriptor to NULL
newFile = NULL;
```

將 buffer 所存的資料寫入新開的檔案裡面。

3. Output Result



4. What I Have Learned

在這次的 lab 裡面，我學到了如何實現一個 stop-and-wait 的傳輸機制，透過 server 與 client 的互相確認，達成補救封包 loss 的目的，另外在這次 lab 中我還學到了許多 file handling 的 function，像是 `fread()`, `fseek()`, `fwrite()`，希望未來我能在以後的其他 project 裡面，用上這次我學到的東西。