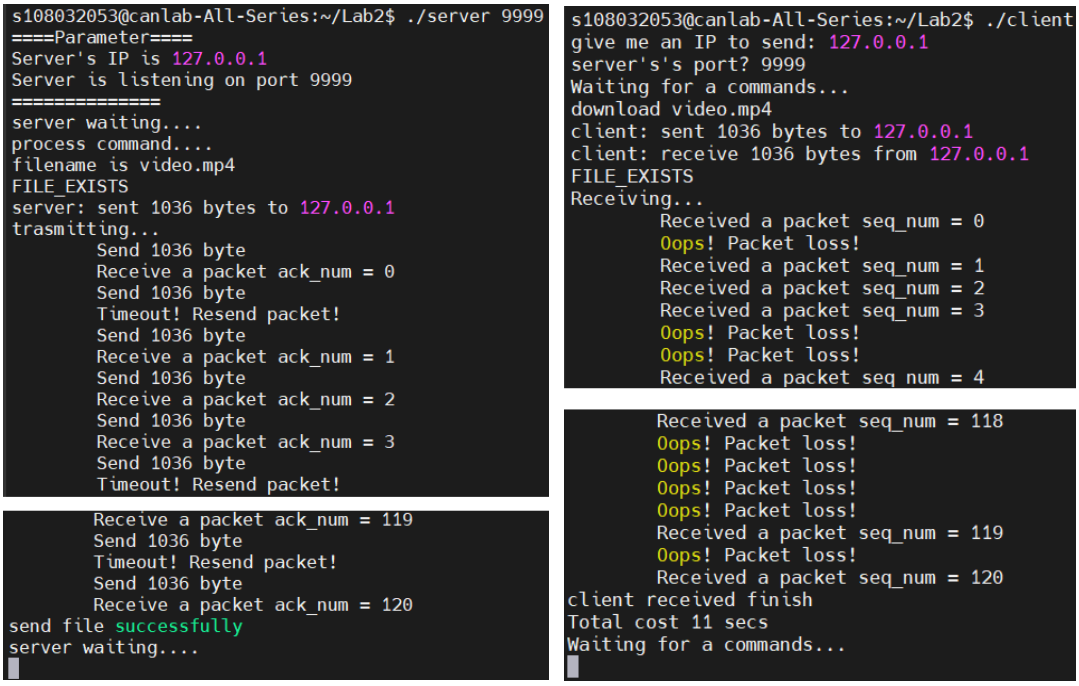
Conputer Network Lab2

學號：108032053 姓名：陳凱揚

1. 執行結果

下圖 1 為在 Linux machine 上編譯、執行後的輸入、輸出結果。首先先執行 server，並輸入 port 為 9999，接著執行 client，輸入 IP 為 127.0.0.1、port 為 9999，就可以輸入指令 ”download video.mp4” 來下載檔案。下圖可以看到 stop&wait、packet loss、timeout 等功能，並在最後成功傳遞檔案，檔案也能順利開啟。



▲ 圖 1

2. 程式功能

(1) server.c

在 receive_thread()中會有個無限迴圈不斷接收封包（下圖 2），並更新 seq 值為 ack_num+1，其中 seq 為 gloabal variable，代表下一次要傳的封包編號。

```
while(recvfrom(sockfd, &rcv_pkt, sizeof(rcv_pkt), 0, (struct sockaddr *)
{
    printf("\tReceive a packet ack_num = %d\n", rcv_pkt.header.ack_num);
    seq = rcv_pkt.header.ack_num+1;
}
```

▲ 圖 2

在 sendFile()中（下圖 3），我首先先傳遞一個封包，儲存檔案大小，讓 client 先知道檔案的大小，以便決定 buffer 的大小。接著將 seq 設為 0、檔案位置移回最前面，就開始一個個傳送封包。

當剩餘檔案大小大於 0 時，會以 fread()讀取資料至 snd_pkt.data、設好 header，並更新剩餘檔案大小。接著以無限迴圈的方式傳送封包，每次傳送後會在 100ms 內檢查是否收到 ack，若未收到表示發生 timeout，重新傳送封包，直到成功收到 ack。當所有資料傳完後，跳出迴圈且關閉檔案。

```
// send the filesize
sprintf(snd_pkt.data, "%d\0", filesize);
if(sendto(sockfd, &snd_pkt, sizeof(snd_pkt), 0, (struct sockaddr*)&client_info
    printf("sendto error\n");
    return -1;
}

seq = 0;
rewind(fd);
while(filesize > 0){
    int readnum = fread(snd_pkt.data, sizeof(char), sizeof(snd_pkt.data), fd);
    snd_pkt.header.seq_num = seq;
    snd_pkt.header.isLast = (filesize==readnum?1:0);
    filesize -= readnum;
    while(1){
        printf("\tSend %ld byte\n", sizeof(snd_pkt));
        if(sendto(sockfd, &snd_pkt, sizeof(snd_pkt), 0, (struct sockaddr*)&cli
            printf("sendto error\n");
            return -1;
        }
        clock_t expireTime = (clock()*1000)/CLOCKS_PER_SEC+TIMEOUT;
        while((clock()*1000)/CLOCKS_PER_SEC < expireTime){
            if(seq == snd_pkt.header.seq_num+1) break;
        }
        if(seq == snd_pkt.header.seq_num+1) break;
        printf("\tTimeout! Resend packet!\n");
    }
}
printf("send file successfully\n");
fclose(fd);
```

▲ 圖 3

(2) client.c

對應到 server 一開始就先傳送檔案大小的資料，在 recvFile()中一開始會先接收這個封包，並動態分配對應大小的 buffer（下圖 4）。

```
// Get the filesize and dynamic allocate buffer
if(recvfrom(sockfd, &rcv_pkt, sizeof(rcv_pkt), 0, (s
    printf("recvfrom error\n");
    return -1;
}
int filesize = atoi(rcv_pkt.data);
char *buffer = (char*)malloc(filesize*sizeof(char));
```

▲ 圖 4

首先 index 代表 buffer 存到哪裡了，接著便以無限迴圈不斷接受封包，每次接到後就模擬封包是否遺失，遺失的話直接 continue 重新接收封包，成功的話就將收到的資料存進 buffer，而在這邊會確保不會寫超過 buffer 的大小，且 index 最大只到 filesize。最後回傳 ack，再檢查是不是最後一個封包，是的話就將 buffer 的資料以 fwrite()寫進檔案裡，並結束迴圈且關閉檔案（下圖 5）。

```
int index = 0;
memset(snd_pkt.data, '\0', sizeof(snd_pkt.data));

printf("Receiving...\n");
while(1)
{
    if(recvfrom(sockfd, &rcv_pkt, sizeof(rcv_pkt), 0, (struct sockaddr*)&i
        printf("recvfrom error\n");
        return -1;
    }
    if(isLoss(0.5)){
        printf("\tOops! Packet loss!\n");
        continue;
    }
    printf("\tReceived a packet seq_num = %d\n", rcv_pkt.header.seq_num);
    memcpy(buffer+index, rcv_pkt.data, min(DATA_SIZE, filesize-index));
    index = min(index+DATA_SIZE, filesize);

    snd_pkt.header.ack_num = rcv_pkt.header.seq_num;
    if(sendto(sockfd, &snd_pkt, sizeof(snd_pkt), 0, (struct sockaddr*)&inf
        printf("sendto error\n");
        return -1;
    }
    if(rcv_pkt.header.is_last == 1){
        fwrite(buffer, sizeof(char), filesize, fd);
        break;
    }
}
printf("client received finish\n");
fclose(fd);
free(buffer);
```

▲ 圖 5