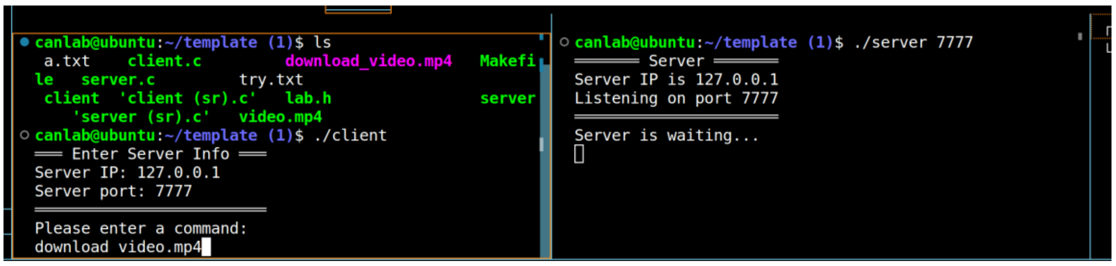
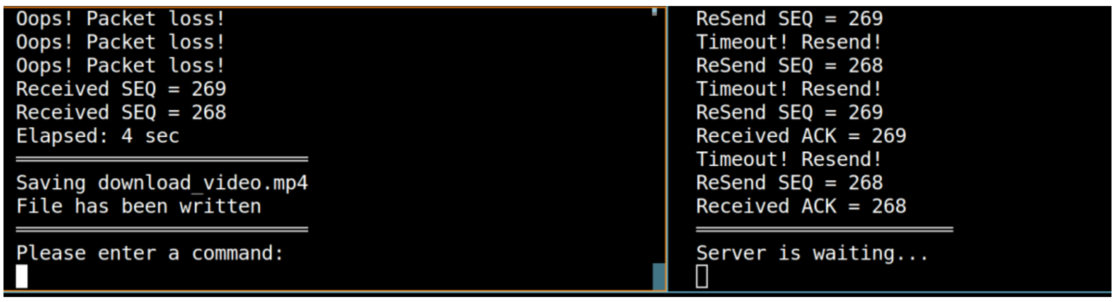
111062108 許凱棋

編譯結果:



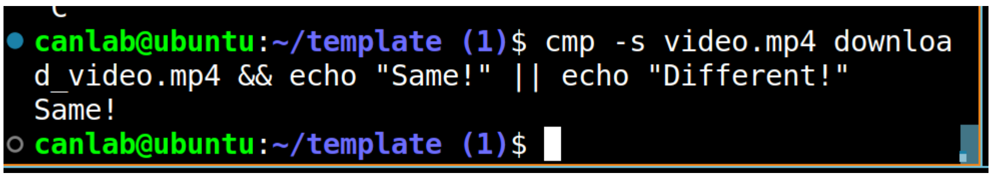
```
● canlab@ubuntu:~/template (1)$ ls
  a.txt    client.c          download_video.mp4   Makefi
le   server.c          try.txt
  client  'client (sr).c'   lab.h               server
     'server (sr).c'   video.mp4
○ canlab@ubuntu:~/template (1)$ ./client
══ Enter Server Info ══
Server IP: 127.0.0.1
Server port: 7777
═══════════════════════════
Please enter a command:
download video.mp4
```

```
○ canlab@ubuntu:~/template (1)$ ./server 7777
═══════ Server ═══════
Server IP is 127.0.0.1
Listening on port 7777
══════════════════════
Server is waiting...
```



```
Oops! Packet loss!
Oops! Packet loss!
Oops! Packet loss!
Received SEQ = 269
Received SEQ = 268
Elapsed: 4 sec
═══════════════════════
Saving download_video.mp4
File has been written
═══════════════════════
Please enter a command:
```

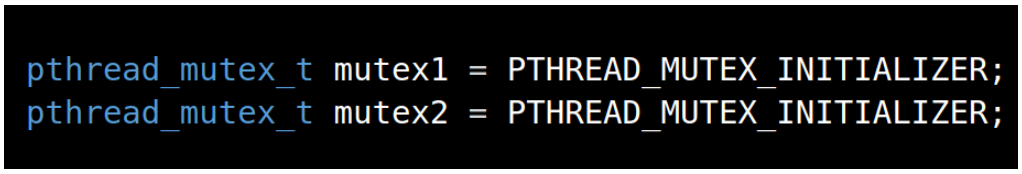
```
ReSend SEQ = 269
Timeout! Resend!
ReSend SEQ = 268
Timeout! Resend!
ReSend SEQ = 269
Received ACK = 269
Timeout! Resend!
ReSend SEQ = 268
Received ACK = 268
═══════════════════════
Server is waiting...
```



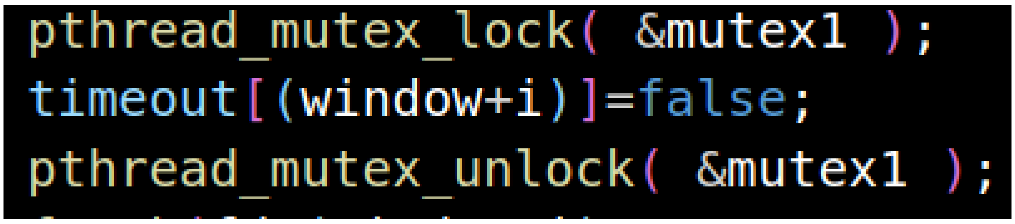
```
● canlab@ubuntu:~/template (1)$ cmp -s video.mp4 downloa
d_video.mp4 && echo "Same!" || echo "Different!"
Same!
○ canlab@ubuntu:~/template (1)$
```

程式碼解釋:

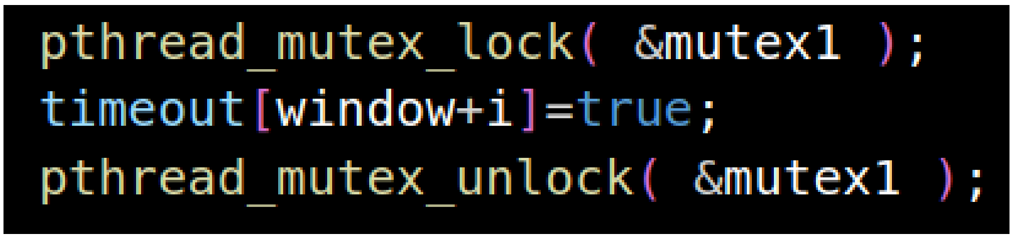
建立 mutex，使平行程是不會同時改變同一個值，如下圖的 timeout[n]，timeout[n]==1 表示超時須重送，以及 ack[n]，ack[n]==1 表示收到 ack(n)



```
pthread_mutex_t mutex1 = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t mutex2 = PTHREAD_MUTEX_INITIALIZER;
```

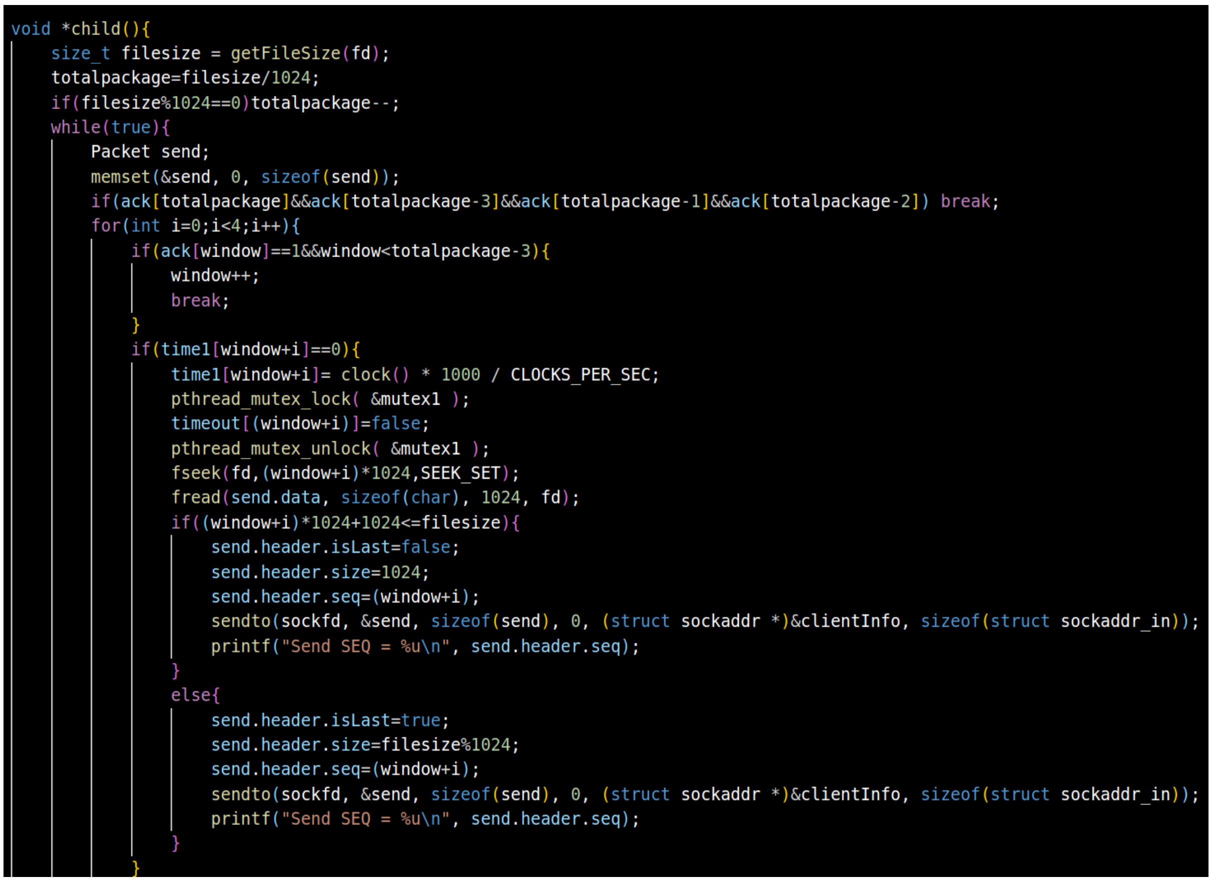


```
pthread_mutex_lock( &mutex1 );
timeout[(window+i)]=false;
pthread_mutex_unlock( &mutex1 );
```



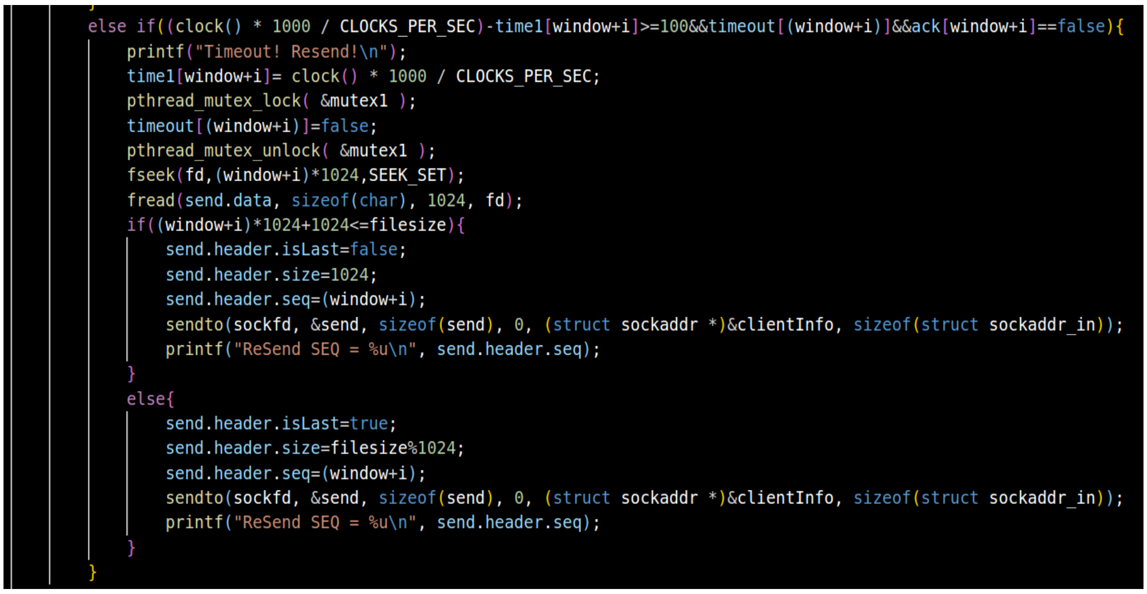
```
pthread_mutex_lock( &mutex1 );
timeout[window+i]=true;
pthread_mutex_unlock( &mutex1 );
```

Server:
以下是第一個平行程式的上半部分，主要控制當 time[n]為零時需傳送封包，並將 time 的值更新。還有控制 window，當當前 window 最小值以 ack 時就會將 window 滑動。會有一個 totalpackage 表示最後一個封包的 seq。



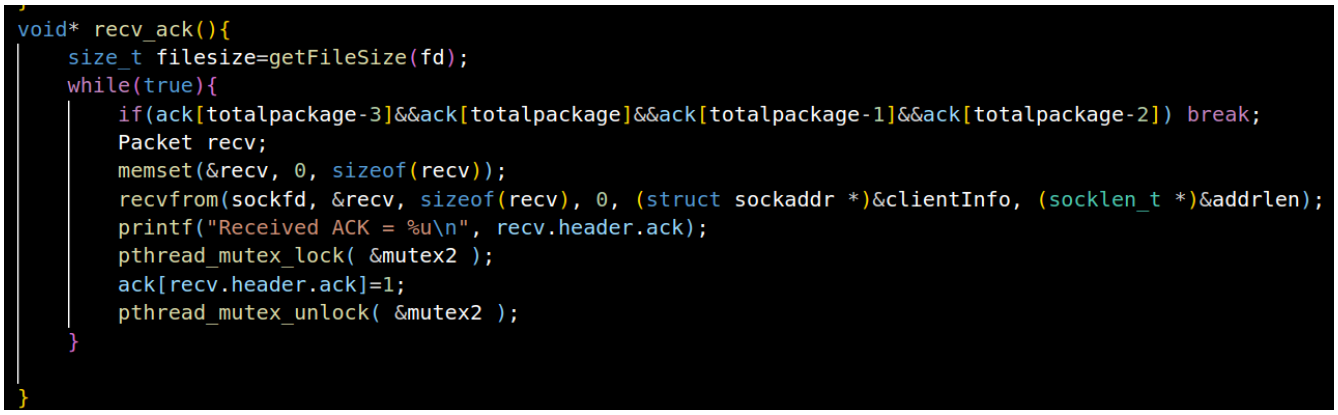
```c
void *child(){
    size_t filesize = getFileSize(fd);
    totalpackage=filesize/1024;
    if(filesize%1024==0)totalpackage--;
    while(true){
        Packet send;
        memset(&send, 0, sizeof(send));
        if(ack[totalpackage]&&ack[totalpackage-3]&&ack[totalpackage-1]&&ack[totalpackage-2]) break;
        for(int i=0;i<4;i++){
            if(ack[window]==1&&window<totalpackage-3){
                window++;
                break;
            }
            if(time1[window+i]==0){
                time1[window+i]= clock() * 1000 / CLOCKS_PER_SEC;
                pthread_mutex_lock( &mutex1 );
                timeout[(window+i)]=false;
                pthread_mutex_unlock( &mutex1 );
                fseek(fd,(window+i)*1024,SEEK_SET);
                fread(send.data, sizeof(char), 1024, fd);
                if((window+i)*1024+1024<=filesize){
                    send.header.isLast=false;
                    send.header.size=1024;
                    send.header.seq=(window+i);
                    sendto(sockfd, &send, sizeof(send), 0, (struct sockaddr *)&clientInfo, sizeof(struct sockaddr_in));
                    printf("Send SEQ = %u\n", send.header.seq);
                }
                else{
                    send.header.isLast=true;
                    send.header.size=filesize%1024;
                    send.header.seq=(window+i);
                    sendto(sockfd, &send, sizeof(send), 0, (struct sockaddr *)&clientInfo, sizeof(struct sockaddr_in));
                    printf("Send SEQ = %u\n", send.header.seq);
                }
            }
```

以下是上面的後半部分控制重送的部分，根據 timeout[n]若為 1 就將該封包重送。除了送出的條件不一樣，其餘都同。



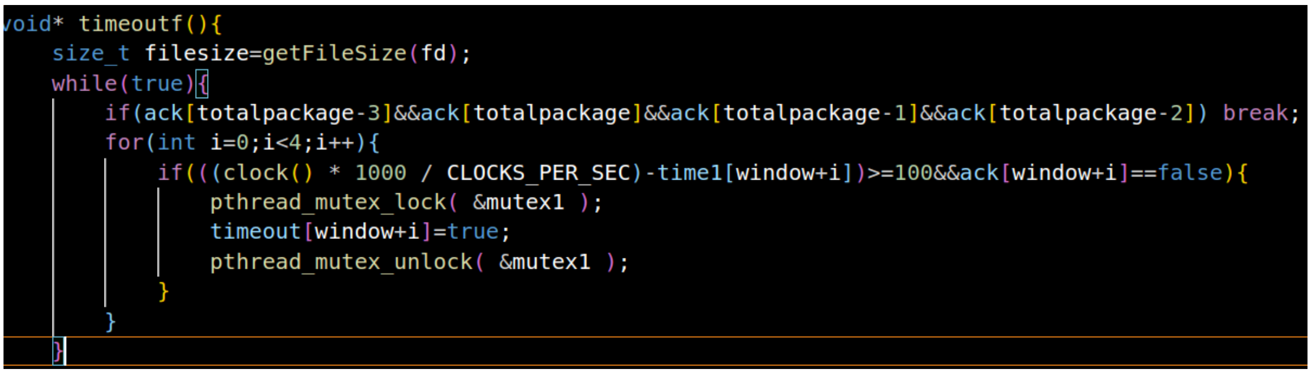
```c
            else if((clock() * 1000 / CLOCKS_PER_SEC)-time1[window+i]>=100&&timeout[(window+i)]&&ack[window+i]==false){
                printf("Timeout! Resend!\n");
                time1[window+i]= clock() * 1000 / CLOCKS_PER_SEC;
                pthread_mutex_lock( &mutex1 );
                timeout[(window+i)]=false;
                pthread_mutex_unlock( &mutex1 );
                fseek(fd,(window+i)*1024,SEEK_SET);
                fread(send.data, sizeof(char), 1024, fd);
                if((window+i)*1024+1024<=filesize){
                    send.header.isLast=false;
                    send.header.size=1024;
                    send.header.seq=(window+i);
                    sendto(sockfd, &send, sizeof(send), 0, (struct sockaddr *)&clientInfo, sizeof(struct sockaddr_in));
                    printf("ReSend SEQ = %u\n", send.header.seq);
                }
                else{
                    send.header.isLast=true;
                    send.header.size=filesize%1024;
                    send.header.seq=(window+i);
                    sendto(sockfd, &send, sizeof(send), 0, (struct sockaddr *)&clientInfo, sizeof(struct sockaddr_in));
                    printf("ReSend SEQ = %u\n", send.header.seq);
                }
            }
```

接下來是判斷 ack 的部分，此段程式會不斷接收直到最後四個封包的 ack 都收到了。



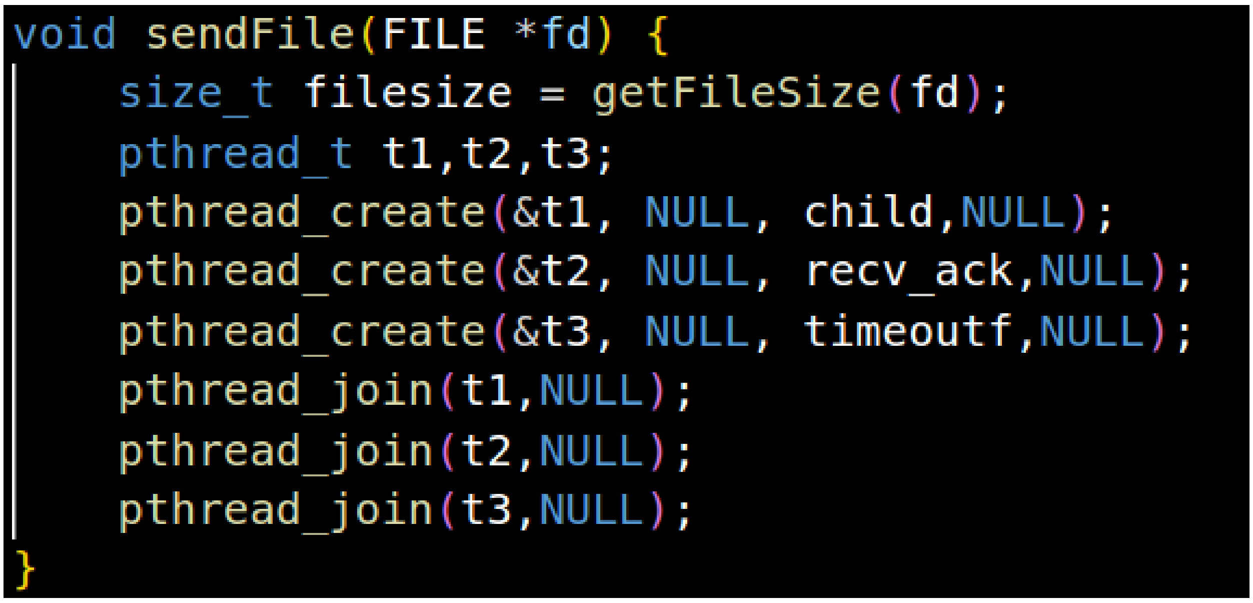
```
void* recv_ack(){
    size_t filesize=getFileSize(fd);
    while(true){
        if(ack[totalpackage-3]&&ack[totalpackage]&&ack[totalpackage-1]&&ack[totalpackage-2]) break;
        Packet recv;
        memset(&recv, 0, sizeof(recv));
        recvfrom(sockfd, &recv, sizeof(recv), 0, (struct sockaddr *)&clientInfo, (socklen_t *)&addrlen);
        printf("Received ACK = %u\n", recv.header.ack);
        pthread_mutex_lock( &mutex2 );
        ack[recv.header.ack]=1;
        pthread_mutex_unlock( &mutex2 );
    }

}
```

再來是 timeout，此 function 會不斷檢查 window 內的封包是否 timeout，若超出就會將 timeout[n]改為 1，進行重傳。



```
void* timeoutf(){
    size_t filesize=getFileSize(fd);
    while(true){
        if(ack[totalpackage-3]&&ack[totalpackage]&&ack[totalpackage-1]&&ack[totalpackage-2]) break;
        for(int i=0;i<4;i++){
            if(((clock() * 1000 / CLOCKS_PER_SEC)-time1[window+i])>=100&&ack[window+i]==false){
                pthread_mutex_lock( &mutex1 );
                timeout[window+i]=true;
                pthread_mutex_unlock( &mutex1 );
            }
        }
    }
```

以下是傳送封包得主程式，將三個平行程式合起來的地方。

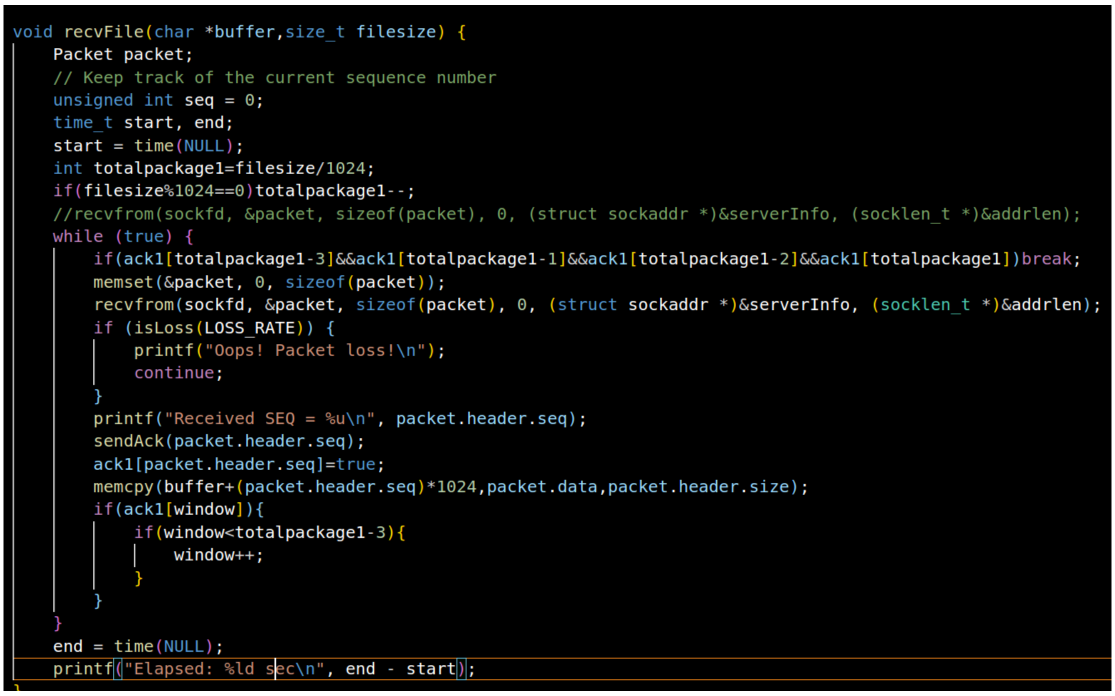


```
void sendFile(FILE *fd) {
    size_t filesize = getFileSize(fd);
    pthread_t t1,t2,t3;
    pthread_create(&t1, NULL, child,NULL);
    pthread_create(&t2, NULL, recv_ack,NULL);
    pthread_create(&t3, NULL, timeoutf,NULL);
    pthread_join(t1,NULL);
    pthread_join(t2,NULL);
    pthread_join(t3,NULL);
}
```

Client:

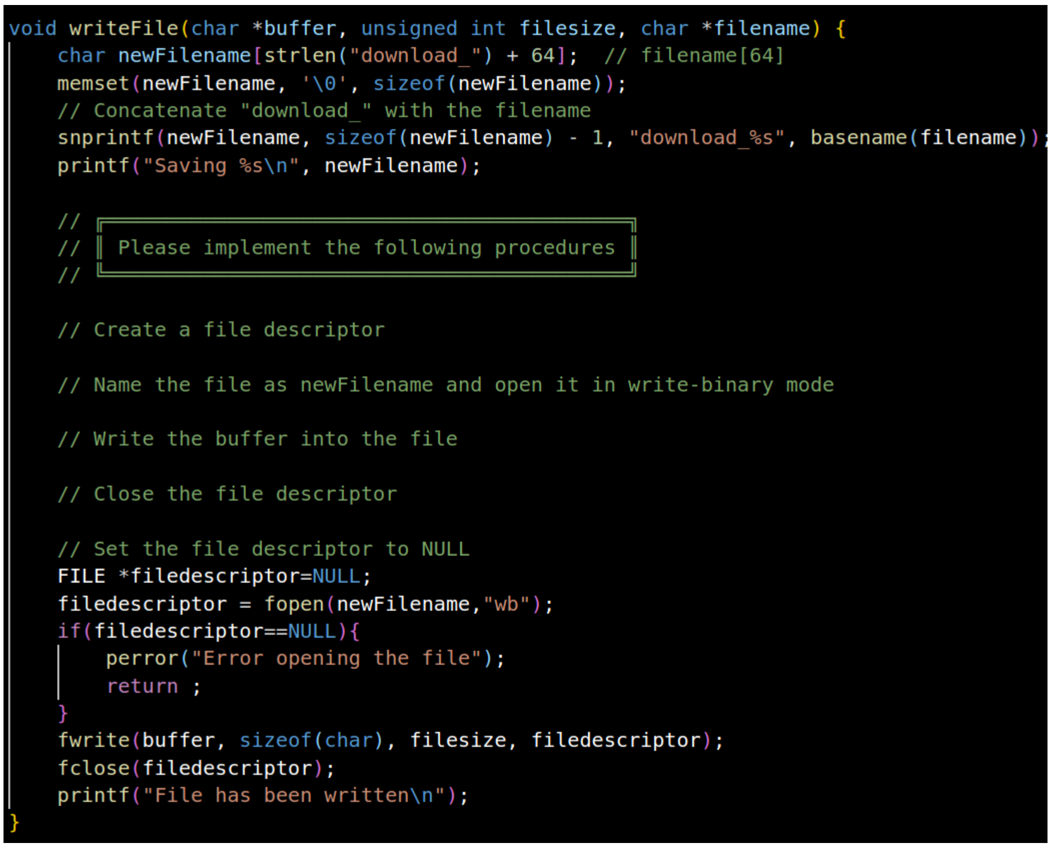
這邊是接收封包得主程式，會一直接收，直到最後四個封包都收到。

這邊會根據 sender 傳過來的 byte 樹進行運算，算出封包數量並記錄。如果當下，然後根據傳過來的 seq 進行運算並放進 buffer 中對應位置。



```c
void recvFile(char *buffer,size_t filesize) {
    Packet packet;
    // Keep track of the current sequence number
    unsigned int seq = 0;
    time_t start, end;
    start = time(NULL);
    int totalpackage1=filesize/1024;
    if(filesize%1024==0)totalpackage1--;
    //recvfrom(sockfd, &packet, sizeof(packet), 0, (struct sockaddr *)&serverInfo, (socklen_t *)&addrlen);
    while (true) {
        if(ack1[totalpackage1-3]&&ack1[totalpackage1-1]&&ack1[totalpackage1-2]&&ack1[totalpackage1])break;
        memset(&packet, 0, sizeof(packet));
        recvfrom(sockfd, &packet, sizeof(packet), 0, (struct sockaddr *)&serverInfo, (socklen_t *)&addrlen);
        if (isLoss(LOSS_RATE)) {
            printf("Oops! Packet loss!\n");
            continue;
        }
        printf("Received SEQ = %u\n", packet.header.seq);
        sendAck(packet.header.seq);
        ack1[packet.header.seq]=true;
        memcpy(buffer+(packet.header.seq)*1024,packet.data,packet.header.size);
        if(ack1[window]){
            if(window<totalpackage1-3){
                window++;
            }
        }
    }
    end = time(NULL);
    printf("Elapsed: %ld sec\n", end - start);
```

以下是撰寫檔案的部分，用 wb 寫出二進位檔案。最後要 fclose。



```c
void writeFile(char *buffer, unsigned int filesize, char *filename) {
    char newFilename[strlen("download_") + 64];  // filename[64]
    memset(newFilename, '\0', sizeof(newFilename));
    // Concatenate "download_" with the filename
    snprintf(newFilename, sizeof(newFilename) - 1, "download_%s", basename(filename));
    printf("Saving %s\n", newFilename);

    // ╔═══════════════════════════════════════════╗
    // ║ Please implement the following procedures ║
    // ╚═══════════════════════════════════════════╝

    // Create a file descriptor

    // Name the file as newFilename and open it in write-binary mode

    // Write the buffer into the file

    // Close the file descriptor

    // Set the file descriptor to NULL
    FILE *filedescriptor=NULL;
    filedescriptor = fopen(newFilename,"wb");
    if(filedescriptor==NULL){
        perror("Error opening the file");
        return ;
    }
    fwrite(buffer, sizeof(char), filesize, filedescriptor);
    fclose(filedescriptor);
    printf("File has been written\n");
}
```

學到的東西:

1. 平行程式的撰寫
2. Mutex 應用
3. Select repeat 運行方式
4. 用程式撰寫二進制檔案