PROGETTO RETI
a.a. 2010/11

Gruppo

Pola Andrea, Bovina Stefano, Daini Irene

Scelta progetto

Livello Basso

Ambiente di sviluppo

Debian e Ubuntu
(gedit)

File:

- const.h
- socket_map_list.h
- TCP_Session.c
- TCP_Session.h

+

Makefile

const.h

Contiene la definizione delle costanti del progetto inerenti le dimensioni dei pacchetti.

socket_map_list.h

Contiene la definizione delle strutture dati, di alcune macro e l'implementazione delle API di mantenimento delle liste e di gestone delle connessioni.

Descrizione campi della struttura connessione socket_map_list:

int fd_pub: è il file descriptor assegnato alla sessione.
int fd_priv: è il vero file descriptor che identifica la connessione
int fd_accept_pub: file descriptor creato dalla accept (fisso).
int fd_accept_priv: è il file descriptor originato da una accept (varia in caso di caduta della connessione, poi ripristinata).

int type: campo che specifica se la connessione considerata è CCS oppure no.

*struct socket_map_list * next*
*struct socket_map_list * prev:* puntatori necessari per spostarsi all'interno della lista delle connessioni.

int cont_connect: contatore. Ce ne serviamo per capire se abbiamo esaurito il numero massimo di tentativi di riconnessione possibili.

struct sockaddr info_server: contiene l'indirizzo del server. **(*)**
socklen_t dim_info: contiene l'informazione sulla dimensione del suddetto indirizzo. **(*)**

Dati socket del chiamante **(*)**
int info_domain: specifica il protocollo usato (IPv4, IPv6, ecc...).
int info_type: specifica il tipo di socket creato (TCP, datagram, raw)
int info_protocol;

Dati bind del chiamante **(*)**
struct sockaddr addr_bind: contiene le informazioni su indirizzo ed eventuale numero di porta locali al quale il sistema operativo collega il socket.
socklen_t addrlen_bind: specifica la dimensione della suddetta struttura.

Dati accept del chiamante **(*)**
struct sockaddr cli_addr: specifica l'indirizzo IP e il numero di porta del client al ci si è connessi.
socklen_t cli_addrlen: specifica la grandezza della suddetto campo.

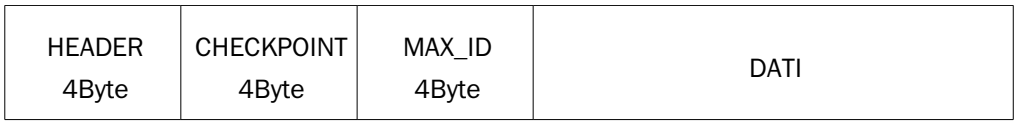
(*) Li salviamo nella struttura perchè lo strato di sessione non conosce questi dati a priori, ma sono necessari per il ripristino della connessione.

Campi per la gestione dei pacchetti
int quota: "segnalibro" per ricordare a che punto del pacchetto siamo arrivati nella sua lettura (utilizzato dalla Read_from_packet)
int checkpoint: identificatore del pacchetto (numero di sequenza) (utilizzato dalla Write)
*char * buffer:* puntatore al buffer privato della connessione del quale ci serviamo per salvare momentaneamente il pacchetto appena letto (utilizzato dalla Read_from_packet).
int sended_byte: contatore per ricordare complessivamente quanti byte sono stati spediti (usato dalla Write).
int readed_byte: contatore per ricordare complessivamente quanti byte sono stati letti (usato dalla Read).
int lastid: campo usato per memorizzare il checkpoint dell'ultimo pacchetto letto con successo (usato dalla Read_from_packet)
int max_id: specifica quanti pacchetti verranno spediti (e quindi dovranno complessivamente essere letti) (usato dalla Read_from_packet e dalla Write).

get_current_body_bytes	macro che calcola quanti byte di informazioni ci sono nel pacchetto in esame (e, di conseguenza, quanti di padding). Viene usata, quindi, per calcolare l'header del pacchetto;
get_pkt_number	macro che calcola quanti pacchetti devono essere inviati per poter consegnare all'altro end-system tutte le informazioni richieste. Viene usata per capire quando terminare le write dei vari pacchetti;
for_each	macro per scorrere la lista delle connessioni;
/ API per le liste */*	
socket_map_init	si occupa di azzerare ogni campo della struttura socket_map_list, per prepararla all'uso;
socket_map_list_add	inserisce un elemento in testa ad una lista;
socket_map_list_add_tail	inserisce un elemento in coda ad una lista;
_list_add	funzione che materialmente esegue l'inserimento per conto delle due funzioni di cui sopra, manipolando i puntatori adeguati nella struttura;
list_del	rimuove l'elemento in testa alla lista;
_list_del	funzione che materialmente esegue la cancellazione per conto della list_del, manipolando i puntatori adeguati nella struttura;
list_is_last	controlla se l'elemento in input è l'ultimo della lista;
list_empty	restituisce 1 se la lista è vuota, 0 altrimenti;
list_next	restituisce l'elemento successivo nella lista (l'elemento stesso in caso la lista sia vuota);
list_prev	restituisce l'elemento precendente nella lista (l'elemento stesso in caso la lista sia vuota);
*/*API per le connessioni */*	
is_ccs	cerca nella lista delle connessioni la connessione che ha come fd (pubblico o privato) quello passato in input e ne restituisce il tipo (0: normale, 1: CCS); -1 in caso di elemento non pervenuto;
get_real_fd	cerca nella lista delle connessioni la connessione con il fd in input (contemplando che possa essere passato un fd pubblico o privato normale o proveniente da una accept/ReAccept) e restituisce il privato. NB: si noti che in caso sia CCS il fd reale è il privato e nel caso, invece sia una connessione TCP normale, allora fd pubblico e privato sono uguali;
stampa_lista	funzione che stampa il set di fd di ogni connessione nella lista (principalmente per debug);
get_connection	ricerca nella lista il fd passato in input (qualsiasi esso sia) e restituisce

	la struttura connessione ad esso associata;
set_new_buffer	crea un nuovo buffer delle dimensioni fornite in input e lo azzera per prepararlo all'uso;
reset_buffer	libera la memoria precedentemente allocata per il buffer privato della connessione in input;
set_socket_fd, set_accept_fd, set_connect_fd	assegnano alla connessione in input i corretti valori dei fd provenienti dalle socket, accept e connect di sistema fatte nello strato di sessione;
set_socket_info, set_bind_info, set_accept_info, set_connect_info	settano i campi inerenti informazioni riguardante la connessione provenienti da una socket, bind, accept o connect di sistema;
update_quota	aggiorna il valore del campo quota (gestione pacchetti) nella struttura della connessione;
reset_quota	azzera il campo quota;
get_quota	restituisce il valore di quota per la connessione in input;
update_sended_byte	aggiorna il numero di byte inviati per quella connessione;
reset_sended_byte	azzera il numero di pacchetti e byte inviati su quella connessione;
get_header_info	estrappola e copia nelle adeguate variabili le informazioni relative a numero di byte buoni per quel pacchetto (header), all'id del pacchetto (checkpoint) e al totale di pacchetti da inviare (max_id);

Struttura pacchetto:



update_lastid	aggiorna l'id dell'ultimo pacchetto inviato a seconda che sia l'ultimo pacchetto da inviare in assoluto oppure no;
free_all_connection	esegue una close di sistema su tutte le connessioni nella lista, nonché ne libera lo spazio in memoria.

TCP_Session.c (parte 1)

File che contiene tutte le funzioni che gli end-system sfrutteranno per instaurare e gestire la connessione.

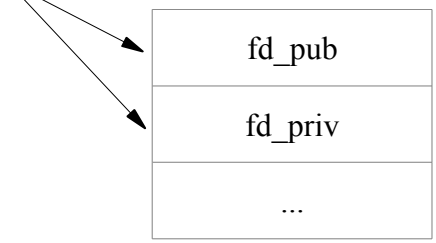
Init_TCP_Session_Module

azzera tutti i campi della connessione (sfruttando l'apposita funzione nel file socket_map_list.h);

Socket

funzione utilizzata esclusivamente dalle connessioni TCP normali; crea una struttura di tipo socket_map_list, che ospiterà tutti i dati della nuova connessione, ne azzera i campi, esegue la socket di sistema e, dopo averne controllato l'esito, setta i vari campi della struttura (fd e info) ed infine aggiunge la struttura in coda alla lista delle connessioni.
Ritorna -1 in caso di errore, int (fd pubblico creato) altrimenti;

Socket → fd in output

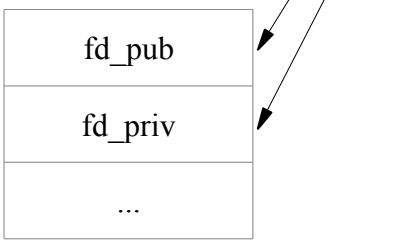


TCP_Session_IPv4_Socket

funzione utilizzata esclusivamente dalle CCS; crea una struttura di tipo socket_map_list, che ospiterà tutti i dati della nuova connessione, ne azzera i campi, esegue la socket di sistema (assegnando il fd che ne ritorna al campo fd privato) e riserva un fd pubblico ad esso associato. Dopo aver controllato l'esito della socket di sistema, setta i relativi campi nella struttura (fd e info) ed infine aggiunge la connessione in coda alla lista delle connessioni.
Ritorna -1 in caso di errore, int (fd pubblico creato) altrimenti;

TCP_Session_IPv4_Socket → fd in output

ReserveFileDescriptor → fd in output



SetsockoptReuseAddr

funzone che permette di configurare il socket in modo che la porta locale alla quale è collegato possa essere riutilizzata;

GetsockoptReuseAddr

funzione in grado di capire se un socket è configurato in modo

tale per cui la porta alla quale è collegato possa essere riutilizzata;

Bind

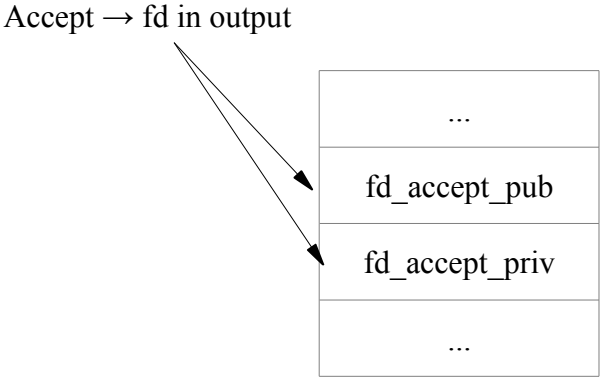
ricava il fd reale della connessione a partire da quello fornito in input, ricerca la relativa struttura della connessione nella lista delle connessioni e ne setta le informazioni inerenti l'indirizzo fornito in input (IP locale e porta associati al socket) e la sua dimensione.
Ritorna il risultato della bind di sistema (0 se è andata a buon fine, -1 altrimenti) oppure -1 in caso il fd in input non compaia in nessuna delle connessioni presenti nella lista;

Listen

(solo il server). Ricava il fd reale a partire da quello in input ed esegue la listen di sistema ("dichiara" il socket pronto per ricevere connessioni).
Ritorna il risultato della listen di sistema (0 se è andata a buon fine, -1 altrimenti) oppure -1, nel caso in cui il fd in input non sia stato reperito in nessuna delle connessioni presenti nella lista;

Accept

(solo il server). Viene eseguita solamente la prima volta che si tenta di accettare una connessione (per ogni connessione). Ricava il fd reale a partire da quello in input, ne estrappola la struttura connessione e setta i giusti campi secondo le informazioni inerenti l'indirizzo server fornite in input. Quindi esegue la accept di sistema, assegnando il valore ritornato al campo fd_accept privato e riserva un fd_accept pubblico per la connessione (mette il server in ascolto di eventuali connessioni richieste di connessione).
Ritorna il fd_accept pubblico oppure -1, nel caso in cui il fd in input non sia stato reperito in nessuna delle connessioni presenti nella lista;



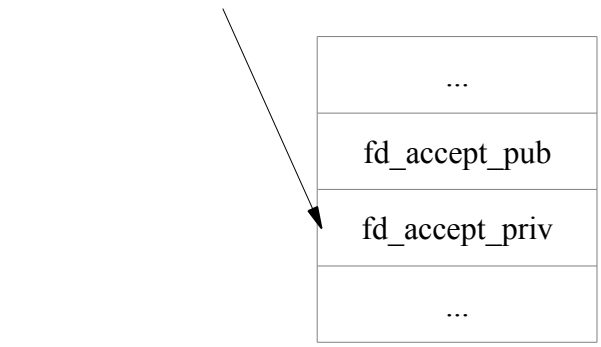
Connect

(solo il client). Viene eseguita solamente la prima volta che si tenta di completare la connessione al server (per ogni connessone) ed ha il compito di permettere al client di stabilire una connessione con il server.
Vengono trattati distintamente in casi in cui questa funzione viene chiamata da una connessione TCP normale oppure da

una CCS.
Se si tratta di una connessione normale, allora tutto ciò che facciamo è eseguire la connect di sistema e restituire il risultato, qualsiasi esso sia.
Se, invece, abbiamo a che fare con una CCS, innanzitutto ricaviamo il fd reale a partire da quello fornito in input, gestendo un possibile errore nella sua ricerca, quindi estrappoliamo la relativa connessione associata. Solo ora eseguiamo la connect di sistema. Se non dovessero sussistere errori, allora non ci rimane che settare corretamente i campi della struttura connessione inerenti le informazioni del server al quale ci siamo connessi e ritornare il risultato della connect di sistema. Essendo CCS, nella gestione di un errore della connect di sistema ci dobbiamo occupare di ritentare di instaurare la connessione, compito affidato alla funzione Ripristino;

ReAccept

(solo il server CCS). Viene eseguita ogni volta che si presume sia caduta la connessione (chiamata dalla Ripristino).
Esegue una Close sul fd_accept privato della connessione. In seguito ricava il fd reale a partire dal fd fornito in input e ne estrappola la relativa struttura connessione. Quindi esegue una nuova accept di sistema, assegnando il risultato al fd_accept privato nella connessione.
Ritorna il risultato della accept di sistema (un intero che rappresenta il nuovo fd_accept privato, in caso di buona riuscita, -1 altrimenti);



ReConnect

(solo il client CCS). Viene eseguita ogni volta che si presume sia caduta la connessione (chiamata dalla Ripristino).
Come prima cosa esegue una Close sul fd privato della connessione. Quindi esegue una socket di sistema per poter riservare un altro fd privato nel tentativo di reinstaurare la connessione perduta. Infine esegue la connect di sistema sul nuovo fd privato (qualsiasi sia il risultato della socket).
Ritorna il risultato della connect di sistema (0, se è andata a buon fine oppure -1).
NB: sarà la funzione chiamante Ripristino che si occuperà di ritentare la ReConnect in caso di errore di quest'ultima;

Ripristino	In breve: viene chiamata ogniqualvolta si presume che la connessione sia caduta. Si occupa di smistare i chiamanti in base alla loro tipologia (client o server), per poter richiamare le corrette funzioni che tenteranno di reinstaurare la connessione (rispettivamente ReConnect o ReAccept). Più nel dettaglio: estrae la connessione associata al fd che le viene passato in input, quindi ne ricava il pubblico. Dal momento che viene chiamata sia in caso si tratti di connessione TCP normale, sia nel caso sia CCS, dobbiamo distinguere i casi. Se è connessione normale, allora non dobbiamo tentare di reinstaurare la connessione, quindi basta fornire un errore in output. Se è CCS, a questo punto dobbiamo distinguere se il chiamante è un server oppure un client. Per fare ciò sfruttiamo il campo fd_accept_pub (se è 0, allora il chiamante è client, server altrimenti). Caso client: possiamo tentare la ReConnect fino a MAX_TENTATIVI volte. Per ogni volta eseguo la ReConnect e controllo il risultato: se è 0 allora la connessione è stata ristabilita correttamente, quindi posso interrompere i tentativi ritornare al chiamante. Se non riesco a ristabilire la connessione entro MAX_TENTATIVI tentativi, allora non mi resta che ritornare al chiamante un errore. Caso server: sfrutto la select per verificare (fino ad un massimo di MAX_TENTATIVI tentativi) se il socket è pronto per tentare una ReAccept. Nel caso lo sia, eseguo la ReAccept e ritorno il nuovo fd_accept privato; altrimenti, se esaurisco tutti i tentativi senza ottenere una risposta positiva dalla select, ritorno errore al chiamante. **NB:** si assume che se avviene un crash di tipo d') allora questo coinvolge tutte le connessioni. Per questo motivo vengono ripristinate tutte quante contemporaneamente.
Read	permette di leggere i dati che pervengono dall'altro end-system. Ricava il fd reale a partire da quello fornito in input (gestendo il caso di errore nella sua ricerca) e ne estrappola la relativa struttura connessione. Quindi richiama la Read_from_packet. Infine controlliamo il risultato che ci perviene da quest'ultima funzione, la quale restituisce il numero di byte letti dal pacchetto in esame: se è uguale o minore di 0, allora si suppone che la connessione sia caduta, quindi affidiamo alla Ripristino il compito di ristabilirla; altrimenti non resta che aggiornare il campo readed_byte (che tiene il conto di quanti byte complessivamente abbiamo letto) ed infine ritorniamo il risultato della Read_from_packet al chiamante (la Readn);
Read_from_packet	ha la funzione di leggere a mano a mano i pacchetti che pervengono e di inviare il corrispondente ack. Si compone di 4 blocchi principali: il caricamento del

pacchetto, il controllo di eventuali pacchetti doppi, la gestione del pacchetto in caso la quantità di dati da leggere richiesti dal chiamante sia inferiore a quella dei complessivi dati “buoni” all'interno del pacchetto (caso 1) e la gestione del caso in cui la quantità da leggere sia maggiore o uguale alla grandezza dei dati “buoni” all'interno del pacchetto (caso 2).

CARICAMENTO PKT: prepara un nuovo buffer, legge un pacchetto dal buffer di sistema e gestisce gli eventuali errori. Infine estrae le informazioni di header, checkpoint e massimo id pacchetto possibile.

CONTROLLO PKTs DOPPI: manda all'altro end-system un ack per quel pacchetto, che in realtà viene scartato, per passare alla lettura (comprensiva di gestione errori) del pacchetto successivo.
NB: il controllo viene fatto grazie all'aggiornamento e alla consultazione di un particolare campo della struttura (lastid) che tiene in memoria l'id dell'ultimo pacchetto la cui lettura si è conclusa correttamente (e quindi anche il relativo invio dell'ack). Viene resettato una volta che è stato inviato l'ultimo pacchetto della serie.

CASO 1 ... dati da leggere < dati nel pkt:
ritorna "count" al chiamante e aggiorna la quota (il “segnalibro”).

CASO 2 ... dati da leggere >= dati nel pkt:
invia un ack per quel pkt all'altro end-system, gestisce le variabili per il controllo di eventuali pacchetti doppi, azzera la quota ed infine ritorna al chiamante il totale byte letti.

Valori di ritorno: -1 in caso di errore di lettura pkt o scrittura ack; int (byte letti) altrimenti;

Write

ha la funzione di scrivere sul buffer di sistema i dati che poi verranno inviati dal sistema operativo all'altro end-system. Ricava il fd reale a partire da quello fornito in input, gestendo l'eventuale errore nella sua ricerca, quindi estrappola la relativa

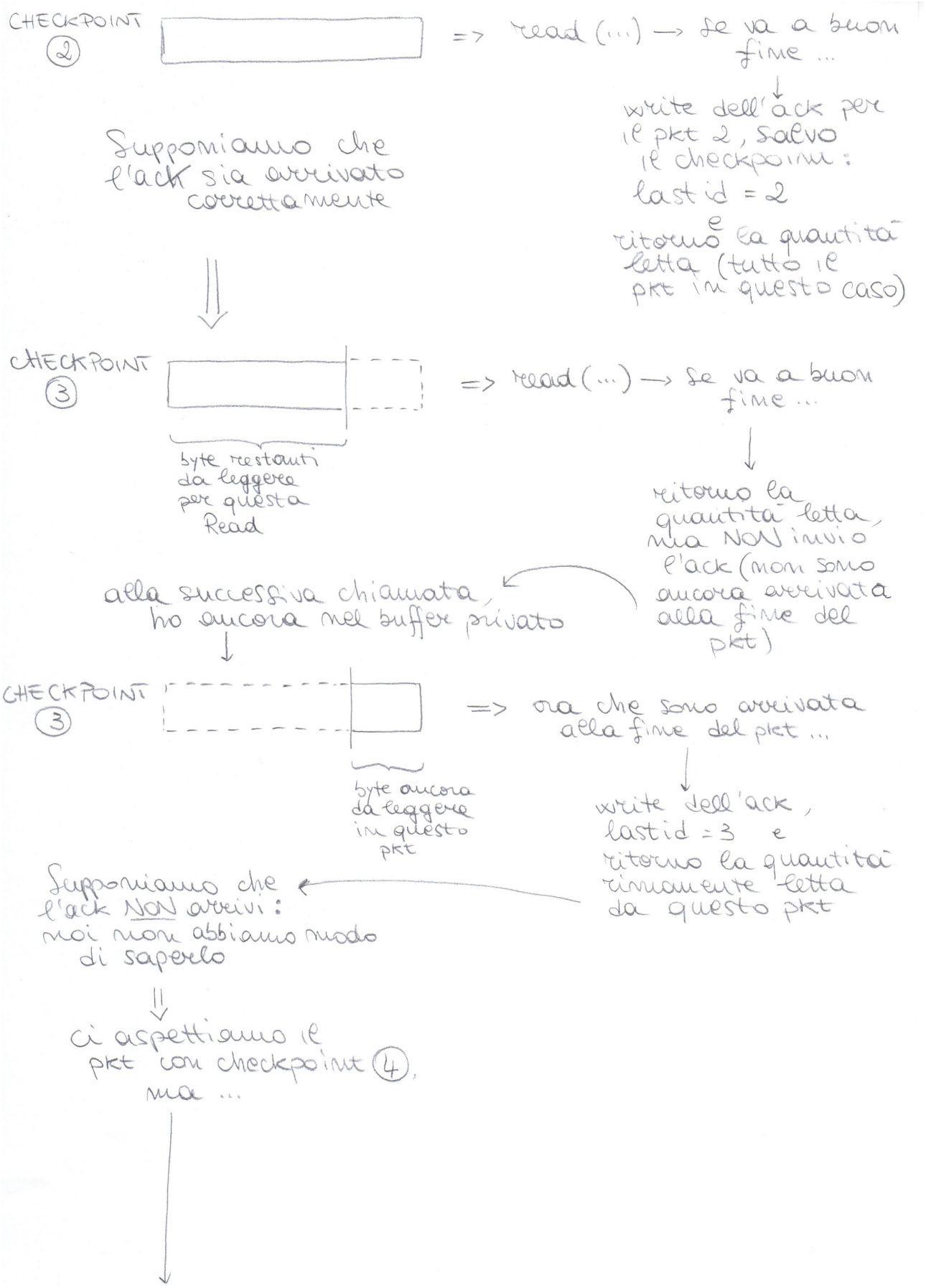
	struttura connessione. A questo punto calcola quanti pacchetti dovrà complessivamente inviare per poter far pervenire all'altro end-system count byte totali e lo memorizza nell'apposito campo della struttura. Quindi, fino a che non ha inviato il numero di pacchetti calcolato, compone mano a mano ogni pacchetto, inserendo: header (contenente l'informazione sulla quantità di dati "buoni" presenti nel pacchetto), chechpoint (id pacchetto), max_id (numero pacchetti da inviare complessivamente) e i veri e propri dati, presi dal buffer passato in input alla Write [NB: se i dati rimasti da scrivere nel pacchetto sono inferiori alla dimensione del corpo del pacchetto stesso, il restante spazio è azzerato.]. A questo punto eseguo una send di sistema e controllo che vada a buon fine; se così non è, cedo il controllo alla Ripristino per ristabilire la connessione. Se, al contrario, la send mi comunica che ho scritto esattamente tutti i byte che mi aspettavo di scrivere, allora mi metto in lettura dell'ack. Se questo non arriva, cedo nuovamente il controllo alla Ripristino, altrimenti non mi resta che aggiornare il numero di byte che complessivamente ho scritto, gestire i checkpoint e ritornare al chiamante la quantità complessiva di byte scritti;
Close	chiude un socket e termina la connessione (non prima che sia terminato l'invio o la ricezione di dati ancora in transizione sulla rete). Effettua la close di sistema sul fd reale (ricavato a partire da quello fornito in input) e ne ritorna il risultato, che è 0 se è andata a buon fine oppure -1 in caso di errore. Il suo scopo è chiudere la connessione per poi liberare il socket;
Close_TCP_Session_Module	esegue una close di sistema su tutte le connessioni nella lista, nonché ne libera lo spazio in memoria.
ReserveFileDescriptor	apre un fd in sola lettura per garantire che l'intero che lo rappresenta non venga utilizzato da un'altra system call.

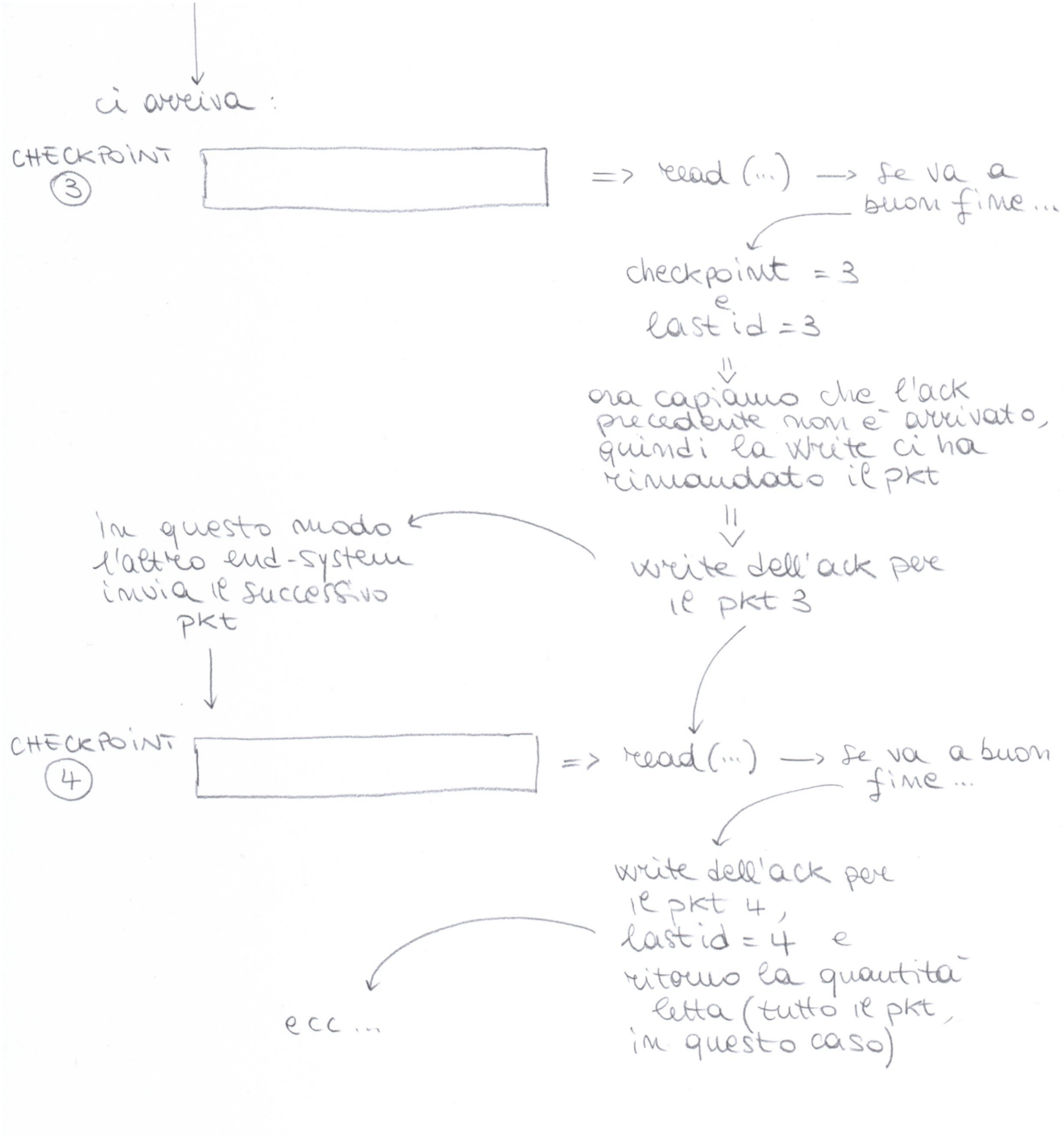
TCP_Session.c (parte 2): scelte progettuali

- I pacchetti sono di lunghezza fissa. La loro dimensione prima della compilazione può essere modificata a piacimento (modificando il valore della costante PKT_DIM e nei limiti delle dimensioni del buffer di sistema) senza che ciò comprometta il funzionamento del progetto. NB: aumentando la dimensione del pkt, aumenta l'efficienza della trasmissione.

- Write bloccante: prima di proseguire nell'esecuzione attende l'ack per il pacchetto appena inviato.

- Algoritmo per il controllo dei pacchetti doppi:





TCP_Session.h

Il file contiene la dichiarazione delle principali costanti di gestione dei timeout e dei prototipi di tutte le funzioni contenute nel file TCP_Session.c .

Makefile

Contiene i comandi per ripulire i file oggetto e compilare il progetto.