



Progetto in Team

– Simulazione dello stack ISO/OSI in C



Obiettivo

In gruppi di 5 persone, implementare uno stack ISO/OSI GIOCATTOLO composto da **7 librerie C (una per livello)**, le cui funzioni send/receive incapsulano/decapsulano un messaggio di chat.

Ogni studente gestisce **un solo livello** e deve:

1. aggiungere un **header** nella funzione `*_send()`;
2. rimuovere e validare l'header nella funzione `*_receive()`;
3. chiamare il livello sottostante in `send` e quello sovrastante in `receive`.

Il progetto si presta alla predisposizione di strumenti di condivisione per lo sviluppo in team. In prima battuta potreste usare drive condividendo la cartella di progetto, ma suggerisco di cimentarsi con `git` (si integra facilmente in VSC) mettendo il progetto in repository come **github.com**



Struttura del progetto

```
progetto/
├─ mezzo_trasmissivo / .h
├─ livello1_fisico.c / .h
├─ livello2_collegamento.c / .h
├─ livello3_rete.c / .h
├─ livello4_trasporto.c / .h
├─ livello5_sessione.c / .h
├─ livello6_presentazione.c / .h
├─ livello7_applicazione.c / .h
└─ main.c                (fornito)
```

Firma standard

```
/* nel file livelloX_nome.h */

char* livelloX_send(const char* dati /* + eventuali param */);

char* livelloX_receive(const char* pdu);
```

Header dei PDU per ogni livello

Liv.	Nome	Header da pre-pendere	Note funzionali sintetiche
7	Applicazione	<i>(nessun header – solo testo utente)</i>	Legge/scrive il messaggio
6	Presentazione	[PRES][ENC=ROT13,UPPER,BASE64,LE,BE,ecc]	Codifica/decodifica
5	Sessione	[SESS][INIT CLOSE ID=12345]	Gestisce apertura/chiusura
4	Trasporto	[TRANS][FRAG=n/N][ID=XX]	Frammenta oltre MTU (20 B)
3	Rete	[NET][SRC=ip_src][DST=ip_dst][TTL=n]	Inserisce IP + TTL (solo decremento facoltativo, no routing)
2	Collegamento dati	[MAC][SRC=AA:BB][DST=CC:DD]	Aggiunge MAC + checksum
1	Fisico	[BITSTREAM]	Simula sequenza di bit (come stringa)

Indicazioni per livello 2 – Data Link

Livello 2 semplificato: questo livello appone un codice di controllo in coda al payload. Quando il ricevente (destinatario) riceve il frame, verifica il checksum (CRC) e in caso positivo inoltra il payload al livello superiore. In caso di problemi si limita a scartare il frame. Nasce il problema di come fa il livello 3 a capire che che il pacchetto è andato perso, ma lascio a voi di intuire la soluzione.

Indicazioni per livello 3 – Routing

Livello 3 semplificato: nessuna logica di routing. Limitati ad aggiungere i campi SRC, DST e diminuire TTL di 1 nel receive. Se TTL diventa 0, stampa un messaggio di errore.

Indicazioni per livello 4 – Frammentazione

- Definire `#define MTU 20` (byte)
 - Se il payload ricevuto supera `MTU`, suddividerlo in più frammenti con header `[TRANS][FRAG=k/N][ID=nn]`.
 - In ricezione, ricomporre i frammenti quando sono tutti presenti. Nota: se mancassero dei frammenti (persi o corrotti) il Livello 4 dovrebbe gestirne la ritrasmissione ed eventualmente avvisare il livello superiore.
-

Indicazioni per livello 5 – Sessione

```
char* livello5_send(const char* dati, const char* action);  
// action="INIT" | "NORMAL" | "CLOSE"
```

- Quando `action=="INIT"` aggiungere `[SESS][INIT][ID=12345]`.
 - Quando `action=="CLOSE"` aggiungere `[SESS][CLOSE][ID=12345]`.
 - Il resto dei messaggi usa `[SESS][NORMAL][ID=12345]`.
 - In `receive` verificare/coerenza dell'ID (con quelli in cache).
-

Indicazioni per livello 6 – Presentazione


Il livello 6 (Presentazione) si occupa principalmente di:

- Traduzione dei dati tra diversi formati: codifiche di caratteri (ASCII, Unicode, EBCDIC), formati numerici (big-endian vs little-endian).
- Cifratura e decifratura (Encryption/Decryption): crittografia per proteggere i dati durante la trasmissione (es. SSL/TLS), sebbene sia spesso associata al livello 7.
- Compressione dei dati: per ottimizzare la velocità di trasmissione (ad es. ZIP, ecc).
- Serializzazione/Deserializzazione: Trasforma strutture dati complesse in stream di byte (es. JSON, XML, protobuf).


Livello 6 semplificato: nel progetto ci limitiamo ad implementare una struttura di selezione che a seconda della codifica indicata codifica/decodifica opportunamente il testo.

Indicazioni per Mezzo Trasmissivo (il cavo)

Per simulare correttamente il **livello 1 - Fisico**, occorre un mezzo fisico su cui inviare i frame di bit. Possiamo farlo mediante due primitive, che si appoggiano ad un buffer **FIFO (coda circolare o lineare)**: rappresenta un *mezzo trasmissivo* dove i frame di bit (o stringhe simulate) vengono trasmessi (appesi in coda) e poi letti dalla funzione che riceve (in testa).

 mezzo_fisico.h

```
//trasmette il frame di bit
void mezzo_fisico_send(const char* bitstream)
//legge dal mezzo trasmissivo il frame di bit
char* mezzo_fisico_receive()
```

 mezzo_fisico.c

```
#define FIFO_SIZE 1024

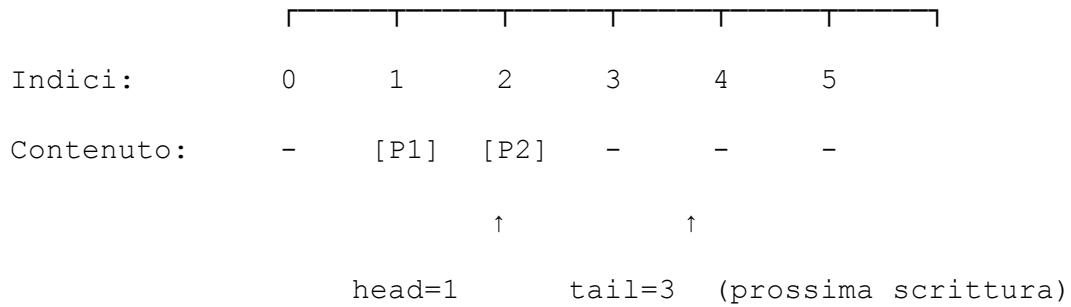
char* fifo[FIFO_SIZE];
int fifo_head = 0;
int fifo_tail = 0;

void mezzo_fisico_send(const char* bitstream) {
    fifo[fifo_tail] = strdup(bitstream); // Copia stringa
    fifo_tail = (fifo_tail + 1) % FIFO_SIZE;
}

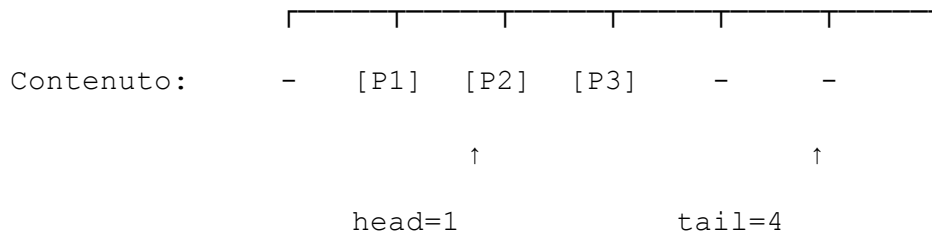
char* mezzo_fisico_receive() {
    if (fifo_head == fifo_tail) return NULL; // FIFO vuota
    char* data = fifo[fifo_head];
    fifo_head = (fifo_head + 1) % FIFO_SIZE;
    return data;
}
```

Di seguito uno schema di come funziona.

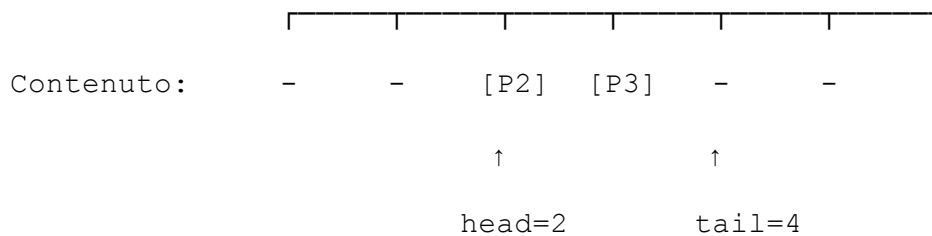
FIFO buffer (dimensione: 6 per esempio)



Invio [P3] **+**



Ricevo **-**



Test (main.c)

```
/* flusso di esempio */
```

```
char* p1 = livello7_send("Ciao!\n");    // incapsulamento ↓
```

```
char* m1 = livello7_receive(p1);        // decapsulamento ↑
```

```
printf("Ricevuto: %s", m1);
```

Dovrà visualizzare "**Ricevuto: Ciao!**" e stampare gli header in ogni fase/livello per visualizzare l'imbustamento.

Livello	Operazione	Dati risultanti
7	Applicazione	"Ciao!\n"
6	Presentazione	[PRES][ENC=ROT13]Cn vb! \a
5	Sessione	[SESS][ID=12345][PRES][ENC=ROT13]Cn vb! \a
4	Trasporto	[TRANS][FRAG=1/1][ID=01][SESS][ID=12345][PRES][ENC=ROT13]Cn vb! \a
3	Rete	[NET][SRC=192.168.1.1][DST=192.168.1.2][TTL=5][TRANS]...
2	Collegamento	[MAC][SRC=AA:BB][DST=CC:DD][NET]...
1	Fisico	[BITSTREAM][MAC][SRC=AA:BB][DST=CC:DD][NET]...

Nota: per il parsing dei vari campi di ciascun header consiglio il ricorso a funzioni standard come strstr(), strtok(), sscanf(). In alternativa scrivere una piccola funzione parser personalizzata per ogni livello.