

ESERCIZIO 3 (Testo)

Implementare un programma in cui un processo padre genera N processi figli.

Il processo padre assegna ad ogni figlio un indirizzo (intero > 1) e a se stesso 1.

Ciclicamente (per X volte) il processo padre interroga ogni figlio richiedendo un valore indicante il numero di richieste e il figlio risponde al padre che lo stampa a video.

Dopo X richieste il padre invia un comando di terminazione ai figli e ne attende la terminazione.

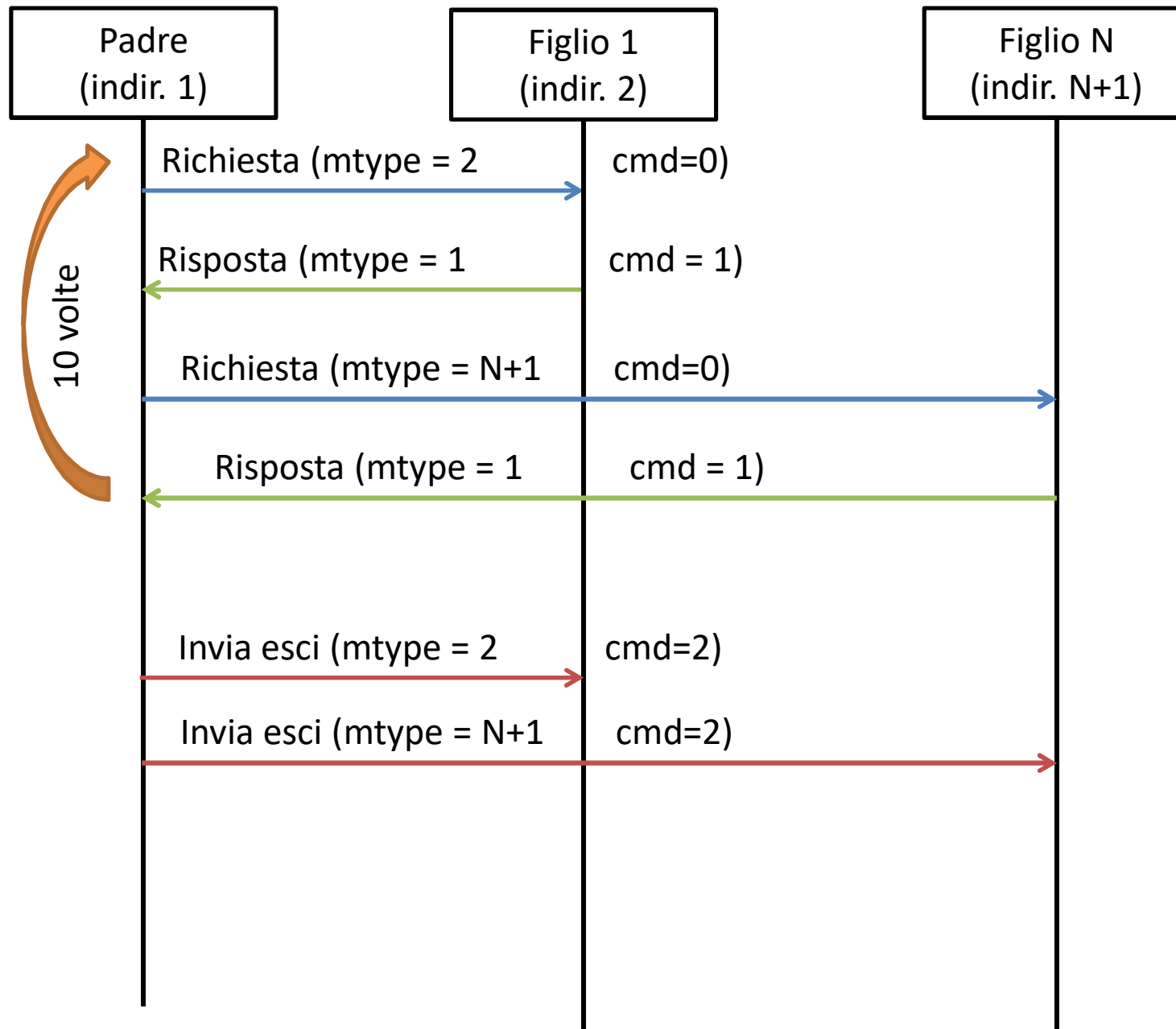
Utilizzare una coda di messaggi per la comunicazione tra padre e figli.

ESERCIZIO 3 (Suggerimenti)

Struttura del messaggio:

```
struct message {  
    long int message_type;  
    int command;  
    int value;  
}
```

- Utilizzare il campo *message_type* per indirizzare un messaggio ad uno specifico processo.
- Utilizzare il campo *command* per specificare il tipo di comando (es. 0=Richiesta, 1=Risposta, 2=Esci).
- Il padre deve avere indirizzo 1, quindi tutti i messaggi indirizzati al padre devono avere tipo 1.



Processo Padre

Genera N figli e passa ad ogni figlio l'indirizzo

```
for(i=0; i<N; i++) {  
    fork()  
    if (figlio)  
        funzione_figlio(i+2)  
}
```



Crea la coda di messaggi



Per 10 volte invia una richiesta ad ogni figlio e attende la risposta

```
for(j=0; j<10; j++) {  
    for(i=0; i<N; i++) {  
        invia_richiesta a i+2  
        riceve_risposta con mtype 1  
    }  
}
```



Invia una esci ad ogni figlio e ne attende la terminazione

```
for(i=0; i<N; i++) {  
    invia_esci a i+2;  
    wait(NULL)  
}
```

Generico Processo Figlio (indir. X)

Crea la coda di messaggi

Fin quando non riceve esci.
Riceve una richiesta ed invia una risposta.

```
while(rx_msg.cmd != 2) {  
    riceve_messaggio con mtype X  
    if (rx_msg.cmd == 0)  
        invia_risposta a 1 (mtype=1)  
}
```

Se si è usciti dal ciclo while vuol dire
che è arrivato il comando ESCI.

Quindi il processo termina!