RT-POSIX: Scambio di Messaggi

Corso di
Progetto e Sviluppo di
Sistemi in Tempo Reale

Marcello Cinque
Daniele Ottaviano

RT-POSIX: Scambio di Messaggi

- Sommario della lezione:
  - Scambio di Messaggi: introduzione
  - Utilizzo delle code di messaggi in RT-POSIX

- Riferimenti
  - https://pubs.opengroup.org/onlinepubs/009695399/idx/realtime.html
  - https://www.rt-thread.io/document/site/programming-manual/posix/posix/#message-queue
  - https://www.softprayog.in/programming/interprocess-communication-using-posix-message-queues-in-linux

Scambio di messaggi

- Il modello a scambio di messaggi (o ad ambiente locale) offre una forma alternativa di comunicazione e sincronizzazione tra processi rispetto al modello a memoria comune e semafori
- Prevede che ciascun processo operi su un proprio spazio di memoria privato e che la comunicazione avvenga mediante uno scambio di messaggi su un canale di comunicazione, offerto dal sistema operativo
- Il modello può essere facilmente esteso ai sistemi distribuiti su rete

Primitive di scambio messaggi

- Due tipologie di primitive

```
send (destination, message)
receive (source, message)
```

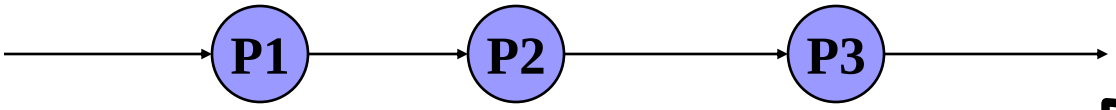
- Si distinguono per:
  - Modello di comunicazione
  - Tipo di sincronizzazione dei processi comunicanti

Modello di comunicazione

- **Diretta**:
  - I processi devono necessariamente indicare il nome del processo con cui vogliono corrispondere
  - **Diretta *simmetrica***: se mittente e ricevente devono entrambi specificare il nome del processo (comunicazione *uno-ad-uno*)
  - **Diretta *asimmetrica***: se solo il mittente indica il nome del destinatario (client-server, comunicazione *molti-ad-uno*)
- **Indiretta**:
  - I messaggi vengono depositati in un contenitore, denominato *mailbox* o *coda di messaggi* (*message queue*), che disaccoppia processi (comunicazione *molti-a- molti*)

Comunicazione Diretta

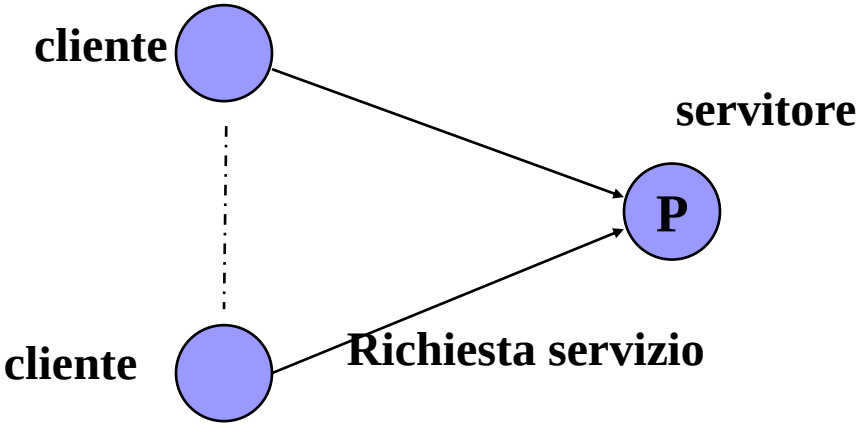
- Diretta e simmetrica: **schema a pipeline**



Send (msg, P2)

Receive (msg, P1)

- Diretta e Asimmetrica: **cliente-servitore**

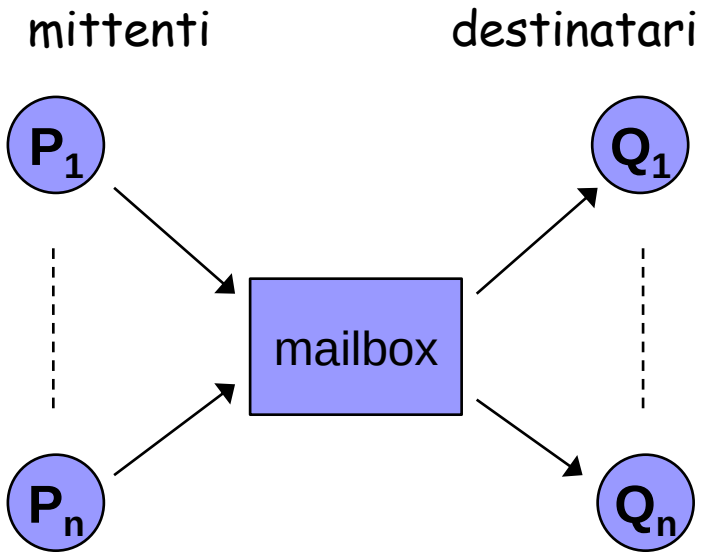


Send (msg, P)

Receive (msg, idCliente)

Comunicazione Indiretta

- I messaggi vengono inviati ad una struttura dati condivisa (una coda, mailbox)
- Un vantaggio di questo approccio è quello che la comunicazione tra sender e receiver è completamente disaccoppiata dalla presenza della mailbox



Send (msg, mailbox)
Receive (msg, mailbox)

Sincronizzazione

- **Send**:
  - **Sincrona**: Il mittente si blocca in attesa che il destinatario riceva
  - **Asincrona**: il mittente invia il messaggio e prosegue nell'esecuzione, senza attendere il destinatario
- **Receive**:
  - **Bloccante**: il destinatario si blocca in attesa del messaggio
  - **Non bloccante**: se il messaggio non è presente, il destinatario prosegue l'esecuzione

Scambio di messaggi in RT-POSIX

- POSIX inter-process comunication (IPC) è stata introdotta nello standard POSIX.1b (IEEE Std 1003.1b-1993) per le estensioni real-time.
- Le code di messaggi POSIX sono state rese disponibili in Linux dalla versione 2.6.6 (maggio 2004).
- Le chiamate POSIX IPC sono conformi allo standard ma potrebbero non essere disponibili su sistemi Unix-like più vecchi. Rispetto alle chiamate IPC System V, le chiamate POSIX IPC hanno un'interfaccia più pulita e sono più facili da usare.
- *N.B. per utilizzare le code di messaggi POSIX bisognerà linkare la libreria real-time librt tramite l'opzione di compilazione –lrt.*

RT-POSIX: Message Queue Control Block

- Ogni coda è identificata da una struttura dati di controllo **mqd_t** definita in questo modo:

```
struct mqdes
{
        rt_uiny16_t refcount;  /* Reference count */
        rt_uint16_t unlinked;  /* Separation status of the
                                  message queue, a value of 1
                                  indicates that the message
                                  queue has been separated */
        rt_mq_t mq;            /*RT-Thread message queue
                                  control block*/
        struct mqdes* next;
};
typedef struct mqdes* mqd_t;  /*Message queue control block
                                  pointer type redefinition*/
```

RT-POSIX, creazione/apertura code: Open

```
mqd_t mq_open(const char * name, int oflag,…);
```

- Crea una nuova coda o ne apre una esistente con un nome univoco.
  - **`name:`** nome della coda
  - **`oflag`**: modalità di apertura.
    - O_RDONLY: coda in sola lettura
    - O_WRONLY: coda in sola scrittura
    - O_RDWR: coda in lettura/scrittura
    - O_CREAT: crea una nuova coda
    - O_CREAT | O_EXCL: ritorna NULL se la coda esiste, altrimenti la crea
    - O_NONBLOCK: Nelle circostanze in cui una mq_send o una mq_receive dovessero bloccarsi, esse falliranno e restituiranno il codice di errore EAGAIN.
    - 0: ritorna NULL se la coda non esiste
  - Restituisce NULL se fallisce, altrimenti restituisce il descrittore della coda.

RT-POSIX, creazione/apertura code(2): Open

```
mqd_t mq_open(const char * name, int oflag,
              mode_t mode, struct mq_attr *attr);
```

- È possibile specificare i permessi e gli attributi della coda.
  - **mode**: specifica i permessi della coda.
  - **attr**: un puntatore agli attributi assegnati alla coda:

```
struct mq_attr
{
        long mq_flags;   /* Flags (ignored for mq_open()) */
        long mq_maxmsg;  /* Max. n° of messages on queue */
        long mq_msgsize; /* Max. message size (bytes) */
        long mq_curmsgs; /* n° of messages currently in queue
                            (ignored for mq_open()) */
};
```

RT-POSIX gestione attributi

```
int mq_getattr(mqd_t mqdes, struct mq_attr *attr);

int mq_setattr(mqd_t mqdes, const struct mq_attr
                    *newattr, struct mq_attr *oldattr);
```

- La funzione `mq_getattr` restituisce la strttura `mq_attr` per la coda dei messaggi con descrittore `mqdes`. Allo stesso modo, la funzione `mq_setattr` permette di impostare gli attributi di una coda. In realtà l'unica modifica attualmente valida è quella del flag `O_NONBLOCK` in `mq_flags`, altre modifiche in `newattr` sono ignorate.
  - **`mqdes:`** descrittore della coda
  - **`attr:`** attributi recuperati dalla coda
  - **`newattr:`** nuovi attributi
  - **`oldattr:`** vecchi attributi
  - Restituisce 0 in caso di successo e -1 in caso di fallimento.

RT-POSIX, detach: Unlink

```
int mq_unlink(const char * name);
```

- Il processo che invoca la funzione chiede la distruzione della coda. La coda viene effettivamente deallocata e le risorse occupate vengono liberate quando non ci sono più processi con un descrittore aperto verso la coda (in altri termini viene distrutta quando tutti i processi fanno la close e il `refcount` diventa 0)
  - **`name:`** nome della coda
  - Restituisce 0 in caso di successo e -1 in caso di fallimento.

RT-POSIX, terminazione: close

```
int mq_close(mqd_t mqdes);
```

- Quando un thread/processo finisce la sua esecuzione, esso chiude le code di messaggi che occupa. Questo succede sia se la chiusura è volontaria sia se è involontaria. Se tutte le funzioni che avevano aperto la coda la chiudono e inoltre la coda è nello stato separato (unlinked), la coda viene eliminata e le risorse occupate vengono liberate.
  - **`mqdes:`** Descrittore della coda
  - Restituisce 0 in caso di successo e -1 in caso di fallimento.

RT-POSIX invio: send

```
int mq_send (mqd_t mqdes, const char *msg_ptr,
             size_t msg_len, unsigned msg_prio);
```

- La funzione invia un messaggio su una coda di tipo **`mqdes`**. Se la coda è piena il thread che prova ad effettuare la send riceverà un codice di errore (`-RT_EFULL`)
  - **`mqdes:`** descrittore della coda
  - **`msg_ptr`**: puntatore al messaggio da inviare
  - **`msg_len`**: lunghezza del messaggio da inviare
  - **`msg_prio`**: priorità del messaggio
  - Restituisce 0 in caso di successo e -1 in caso di fallimento.

RT-POSIX, invio temporizzato: timedsend

```
int mq_timedsend (mqd_t mqdes, const char *msg_ptr,
                  size_t msg_len, unsigned msg_prio,
                  const struct timespec *abs_timeout);
```

- Implementa una send su una coda di tipo **`mqdes`**. Se la coda è piena il thread che prova ad effettuare la send aspetta per un tempo pari ad `abs_timeout` prima di restituire un messaggio di errore.
  - **`mqdes:`** descrittore della coda
  - **`msg_ptr`**: puntatore al messaggio da inviare
  - **`msg_len`**: lunghezza del messaggio da inviare
  - **`msg_prio`**: priorità del messaggio
  - **`abs_timeout`**: tempo di attesa in caso di coda piena
  - Restituisce 0 in caso di successo e -1 in caso di fallimento.

RT-POSIX, ricezione bloccante: receive

```
int mq_receive (mqd_t mqdes, char *msg_ptr,
                size_t msg_len, unsigned *msg_prio);
```

- La funzione rimuove il messaggio più vecchio a priorità più alta da una coda **mqdes**. Se la coda è vuota il thread che ha chiamato la funzione si bloccherà fino a quando non riuscirà a prelevare un messaggio dalla coda.
  - **mqdes:** descrittore della coda
  - **msg_ptr**: puntatore al messaggio ricevuto
  - **msg_len**: lunghezza del messaggio ricevuto
  - **msg_prio**: recupera la priorità del messaggio ricevuto
  - Restituisce La lunghezza del messaggio in caso di successo e -1 in caso di fallimento.

RT-POSIX, ricezione temporizzata: timedreceive

```
int mq_timedreceive (mqd_t mqdes, char *msg_ptr,
                     size_t msg_len, unsigned *msg_prio,
                     const struct timespec *abs_timeout);
```

- La funzione rimuove il messaggio più vecchio a priorità più alta da una coda **mqdes**. Se la coda è vuota il thread che ha chiamato la funzione si bloccherà per un tempo pari ad `abs_timeout` per poi restituire -1 se la coda continua ad essere vuota.
  - **mqdes:** descrittore della coda
  - **msg_ptr**: puntatore al messaggio ricevuto
  - **msg_len**: lunghezza del messaggio ricevuto
  - **msg_prio**: recupera la priorità del messaggio ricevuto
  - **abs_timeout**: tempo di attesa in caso di coda vuota
  - Restituisce La lunghezza del messaggio in caso di successo e -1 in caso di fallimento.

RT-POSIX notifica asincrona: notify

```
int mq_notify(mqd_t mqdes, const struct sigevent *sevp);
```

- La funzione permette di registrarsi o deregistrarsi per la ricezione di una notifica asincrona quando arriva un nuovo messaggio nella coda vuota riferita dal descrittore **`mqdes`**.
  - **`mqdes:`** descrittore della coda
  - **`sevp:`** è un puntatore ad una struttura sigevent.
  - Restituisce 0 in caso di successo e -1 in caso di fallimento.

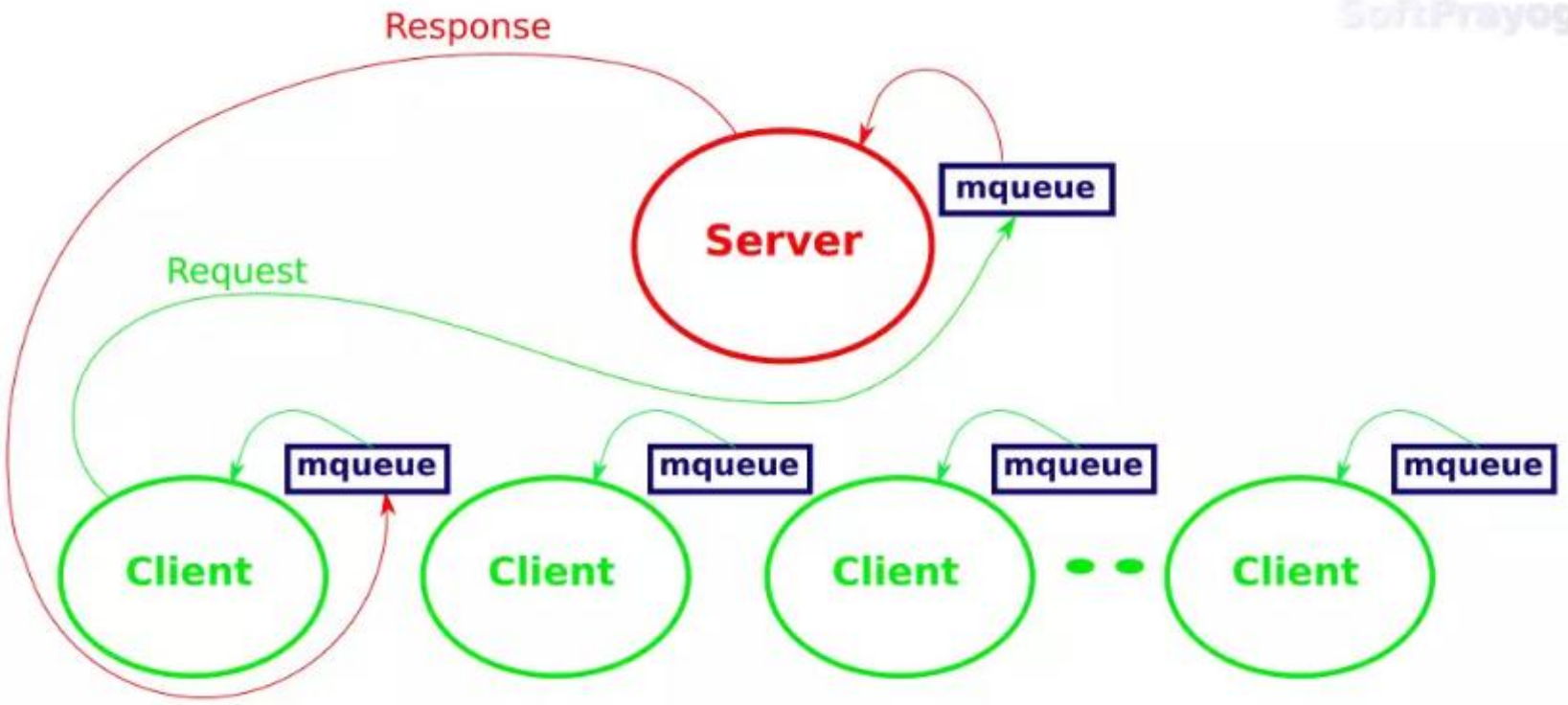
ESEMPIO: Client-Server

- L’esempio mostra l’utilizzo delle code di messaggi POSIX come mezzo per la comunicazione inter-processo tra un Server e uno o piu Client.
- Il Server ha il compito di inviare un **token** ai Client quando richiesto.
- Il token contiene un valore intero che viene incrementato ad ogni richiesta.

ESEMPIO: Client-Server (2)

- Il Server mette a disposizione una coda per ricevere le richieste dei Client.
- I Client inviano le richieste sulla coda creata dal Server.
- Inoltre, ogni Client mette a disposizione una propria coda per ricevere la risposta (il token) dal server.
- I Client leggono il token ricevuto sulla propria coda e lo stampano a video.

ESEMPIO: Client-Server (3)



Interprocess Communication between client and server using Message Queues

https://www.softprayog.in/programming/interprocess-communication-using-posix-message-queues-in-linux