

Sviluppo di software di rete con l'utilizzo di sistemi UNIX

IPC: code di messaggi

Claudio Vicari

Definizioni

- ➔ una coda di messaggi è una lista di messaggi
 - si possono mettere messaggi nella lista
 - si possono leggere e rimuovere messaggi
- ➔ i messaggi sono detti record
- ➔ ogni messaggio ha una sua priorità, decisa da chi lo invia
- ➔ non c'è bisogno che qualcuno stia attendendo un messaggio, per poterlo scrivere (diversamente da pipe e FIFO)
- ➔ un processo può scrivere qualcosa e quindi terminare: un secondo processo può leggere anche dopo la terminazione del primo (ancora in contrasto con pipe e FIFO)

Messaggi

- ➔ le code di messaggi Posix forniscono sempre il messaggio più vecchio, di priorità più alta
- ➔ possono essere impostate per generare segnali quando un messaggio è posto in una coda vuota
- ➔ i messaggi hanno degli attributi:
 - la priorità
 - la lunghezza dei dati contenuti nel messaggio
 - i dati
- ➔ ricordiamo che pipe e FIFO invece, sono canali in cui non c'è distinzione alcuna fra i byte che vengono scritti

Funzione mq_open

```
#include <mqueue.h>
mqd_t mq_open(const char *name, int oflag,
    ...);
```

- ⇒ mq_open crea una coda, o ne apre una già esistente
- ⇒ in oflag possiamo specificare i soliti flag O_RDONLY, O_WRONLY, O_RDWR nonché O_CREAT, O_EXCL
- ⇒ per creare una nuova coda, è necessario specificare anche un modo
- ⇒ si possono specificare anche altri attributi come quarto parametro...

Funzioni mq_close, mq_unlink

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

- ➔ chiude la coda, similmente a quello che avviene con le altre *close*
- ➔ tutte le code aperte da un processo vengono chiuse alla terminazione

```
int mq_unlink( mqd_t mqdes );
```

- ➔ chiede la cancellazione della coda. Questa avverrà, comunque, solo dopo che sarà stata chiusa da tutti i processi

Esempi

- ➔ poiché le code sopravvivono alla chiusura dei programmi, possiamo usare piccoli programmi indipendenti per testarle
- ➔ possiamo creare una coda, inviare un messaggio, riceverlo
 - `./mqcreate /coda`
 - `./mqsend /c 20 1 //manda 20 byte`
 - `./mqreceive /c //attende la ricezione di un msg`
 - `./mqunlink /c //rimuove dal sistema la coda`

Funzioni per gli attributi

```
int mq_getattr( mqd_t mqdes,  
               struct mqattr *attr );  
int mq_setattr( mqd_t mqdes,  
               const struct mqattr *attr,  
               struct mqattr *attr );
```

```
struct mq_attr {  
    long mq_flags;    // 0, O_NONBLOCK  
    long mq_maxmsg; /* massimo n di msg  
                    consentito */  
    long mq_msgsize; /* dimensione max di un  
                    messaggio */  
    long mq_curmsgs; // n di messaggi in coda  
}
```

Attributi delle code

- ➔ nella chiamata ad `open`, possiamo specificare gli attributi come quarto argomento – la `open` considera solamente `mq_maxmsg` e `mq_msgsize`
- ➔ `mq_setattr` può impostare solamente i flag, che consentono solamente di scegliere se le chiamate devono essere bloccanti o meno
- ➔ con `mq_setattr`, possiamo anche avere nel terzo argomento gli attributi precedenti
- ➔ esempio: programmi `mqgetattr`, `mqcreate`

Uso dei messaggi: mq_receive

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

- ➔ riceve un messaggio dalla coda mqdes
- ➔ mq_receive restituisce sempre il messaggio più vecchio, fra quelli di maggiore priorità
- ➔ in msg_prio otteniamo anche la priorità del messaggio ricevuto
- ➔ se non c'è un messaggio disponibile, la funzione si blocca ad attenderne uno
- ➔ se la msg_len è minore di mq_msgsize, la funzione fallisce immediatamente restituendo un errore E_MSGSIZE

Uso dei messaggi: mq_send

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

- ➔ invia un messaggio alla coda mqdes, dandogli una priorità msg_prio
- ➔ il messaggio può anche avere lunghezza 0
- ➔ esempi: mqsend, mqreceive
 - provare mqreceive con l'opzione -n, per renderlo non bloccante

Notifiche ai processi: mq_notify

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

- ➔ questa funzione imposta (o rimuove) la modalità di notifica di eventi per una coda, verso il processo corrente
- ➔ se notification non è NULL, il processo verrà registrato per la notifica
- ➔ se notification è NULL, la registrazione sarà rimossa
- ➔ un solo processo per volta può essere registrato
- ➔ i processi bloccati in mq_receive hanno priorità sui processi in attesa di notifica
- ➔ la registrazione viene rimossa ad ogni notifica

Dettagli per struct sigevent

```
union sigval {
    int sival_int; // valore intero
    void * sival_ptr; // puntatore
};

struct sigevent {
    int sigev_notify; /*SIGEV_{NONE,SIGNAL,
                        THREAD} */
    int sigev_signo; // signo da ricevere
    union sigval sigev_value; /* argomento
                               ulteriore da passare all'handler */
    void(*sigev_notify_function)(union sigval);
    pthread_attr_t *sigev_notify_attributes;
}
```

Dettagli per struct sigevent

```
union sigval {
    int sival_int; // valore intero
    void * sival_ptr; // puntatore
};

struct sigevent {
    int sigev_notify; /*SIGEV_{NONE,SIGNAL,
                        THREAD} */
    int sigev_signo; // signo da ricevere
    union sigval sigev_value; /* argomento
                               ulteriore da passare all'handler */
    void(*sigev_notify_function)(union sigval);
    pthread_attr_t *sigev_notify_attributes;
}
```

Esempio: mqnotifysig1.c

- ➞ proviamo:
 - a> ./mqnotifysig1 /c
 - b> ./mqsend /c 20 3
- ➞ notare però che anche le funzioni sulle code non potrebbero essere chiamate dall'interno di un signal handler!
- ➞ la soluzione potrebbe essere quella dell'impostazione di una variabile globale all'interno dell'handler

Esercizi

- ➔ scrivere un programma che generi un processo padre ed un figlio
- ➔ scrivere in una coda un messaggio di bassa priorità
- ➔ il padre scriva ogni secondo un messaggio ad alta priorità nella coda
- ➔ il figlio, invece, legga dalla coda di messaggi e stampi quello che riceve
- ➔ usare una sincronizzazione con semafori per far sì che il figlio non legga mai il messaggio di alta priorità