

Spedizione/ricezione di messaggi

int msgsnd(int ds_coda, const void *buff, size_t nbyte, int flag)	
Descrizione	invoca la spedizione di un messaggio su una coda
Parametri	1) ds_coda: descrittore della coda su cui si vuole operare 2) buff: puntatore al buffer che contiene il messaggio 3) nbyte: taglia del messaggio, in byte 4) flag: opzione di spedizione (IPC_NOWAIT è non bloccante)
Descrizione	-1 in caso di fallimento

int msgrcv(int ds_coda, void *buff, size_t nbyte, long type, int flag)	
Descrizione	invoca la ricezione di un messaggio da una coda
Parametri	1) ds_coda: descrittore della coda su cui si vuole operare 2) buff: puntatore al buffer che dovrà contenere il messaggio 3) nbyte: numero massimo di byte del messaggio da ricevere 4) type: tipo del messaggio da ricevere 5) flag: opzione di spedizione (IPC_NOWAIT è non bloccante)
Descrizione	-1 in caso di fallimento

Durante la SPEDIZIONE del messaggio sicuramente come primo parametro dobbiamo passare il descrittore della coda di messaggio, quindi il codice operativo di quella coda di messaggio, e poi se vogliamo spedire il messaggio, questo messaggio deve esser stato preventivamente impacchettato in un'area di memoria, e come secondo parametro passiamo il puntatore a quell'area di memoria; Come terzo parametro specifichiamo la taglia in Byte del messaggio, e l'ultimo parametro è un flag che ci permette di cambiare la modalità di spedizione (quindi BLOCCANTE o NON BLOCCANTE). Il flag IPC_NOWAIT INDICA che noi vogliamo eseguire la spedizione in modalità NON BLOCCANTE: quindi se per esempio attualmente all'interno di quella message queue non c'è storage sufficiente per raccogliere il nostro messaggio, noi comunque riprendiamo il controllo e ci troveremo in uno scenario in cui ci viene comunicato un errore (-1).

La parte importante è il secondo parametro, ossia il puntatore all'area di memoria dove noi dove inseriamo il messaggio che vogliamo spedire, e il terzo parametro, ossia la quantità di byte. Se noi andiamo a vedere sul manuale (sulla pagina man) dell'API msgsnd(), E Lì ci viene indicato che il secondo parametro che noi passiamo è vero che è un puntatore ad un'area di memoria generica, ma in quell'area di memoria noi dovremmo aver formattato il messaggio secondo questa regola:

```
struct msgbuf {
    long mtype;
    char mtext[1];
};
```



```
NAME
    msgrcv, msgsnd - System V message queue operations

SYNOPSIS
    #include <sys/types.h>
    #include <sys/ipc.h>
    #include <sys/msg.h>

    int msgsnd(int msgid, const void *msgp, size_t msgsz, int msgflg);
    ssize_t msgrcv(int msgid, void *msgp, size_t msgsz, long msgtyp,
        int msgflg);

DESCRIPTION
    The msgsnd() and msgrcv() system calls are used, respectively, to send messages to, and receive messages
    from, a System V message queue. The calling process must have write permission on the message queue in
    order to send a message, and read permission to receive a message.

    The msgp argument is a pointer to a caller-defined structure of the following general form:

    struct msgbuf {
        long mtype;          /* message type, must be > 0 */
        char mtext[1];       /* message data */
    };


```

Su unix, ci deve essere ALMENO inizialmente un long (quindi un campo) che specifica il TIPO del messaggio. Per questo deve essere un tipo maggiore di 0, e poi c'è il contenuto effettivo del messaggio, nell'esempio sulla man page il contenuto riporta un unico byte, ma qui noi possiamo mettere quello che vogliamo. Quindi abbiamo un'area in memoria in cui all'inizio possiamo scrivere un long e sotto possiamo scrivere la quantità di byte che vogliamo, anche strutturarli secondo la regola che vogliamo, potremmo avere un messaggio che inizialmente è un long e poi sotto è a sua volta una struct, in cui noi andiamo ad inserire informazioni in campi multipli. Nella pagina MAN abbiamo un esempio semplice in cui il contenuto è un array di byte di un solo byte. Ma il terzo parametro che noi andiamo a specificare in questa API, ovvero la taglia del messaggio, questa taglia fa riferimento soltanto per quanto concerne alla seconda parte di questa struct, quindi sotto al long. La receive ci permette di passare un descrittore di una message queue / codice operativo come primo parametro, il messaggio verrà consegnato al chiamante al pointer dell'area di memoria che passiamo come secondo parametro, e come terzo parametro specifichiamo quanti byte vogliamo, e come quarto parametro possiamo specificare qual è il TIPO del messaggio a cui siamo interessati: quindi se ora nel deposito ci sono messaggi di tipo 1 e 2, magari noi vogliamo soltanto quelli di tipo 2, oppure vogliamo un ulteriore tipo che attualmente non c'è e tipicamente rimaniamo bloccati se passiamo un flag (quinto parametro) di default che tipicamente è zero, oppure se passiamo IPC_NOWAIT, riprendiamo immediatamente il controllo e andiamo in errore (-1). Quindi l'ultimo parametro ci permette di specificare se l'operazione di ricezione debba essere eseguita in maniera BLOCCANTE o NON BLOCCANTE. È chiaro che se noi specifichiamo l'operazione in maniera NON BLOCCANTE e il messaggio È PRESENTE, ci viene consegnato: quindi non andremo in blocco - quindi il sistema ci da immediatamente quel messaggio e poi riconsegna il controllo user space al chiamante di questa API - e non ci viene fornito nessun errore, perché il messaggio è stato consegnato al chiamante. Il valore di ritorno è la quantità di byte che ci sono stati dati, perché noi chiediamo un certo messaggio e al più vogliamo ricevere una certa quantità di byte, ma magari il messaggio è più breve della quantità di byte che noi stiamo chiedendo e quindi dobbiamo sapere quanti byte REALMENTE ci sono consegnati. Questa receive è tale per cui che, il tipo del messaggio che noi possiamo voler ricevere, può essere specificato come un tipo 0, questo vuol dire che non siamo interessati a nessun tipo e quindi ci verrà fornito il messaggio più vecchio all'interno della coda.

```
* If msgtyp is 0, then the first message in the queue is read.

* If msgtyp is greater than 0, then the first message in the queue of type msgtyp is read, unless MSG_EXCEPT
was specified in msgflg, in which case the first message in the queue of type not equal to msgtyp will be
read.

* If msgtyp is less than 0, then the first message in the queue with the lowest type less than or equal to
the absolute value of msgtyp will be read.
```

Quindi stiamo facendo una ricezione FIFO su tutti i messaggi all'interno della coda. Se il message type è maggiore di 0, significa che siamo interessati davvero a quel message type e quindi ci verrà dato il primo messaggio che è nella coda di quel TIPO specifico, quindi stiamo facendo FIFO solo su quel tipo di messaggio. Se il message type è minore di 0 allora significa che non siamo interessati ad un unico message type, ma a tutti i message type che sono identificabili considerando codici numerici che sono minori uguali al valore assoluto della quantità che noi stiamo passando. Se noi passiamo un valore negativo, vi è l'equivalente valore assoluto associato, quindi se noi passiamo -3, il suo valore assoluto è 3, in questa ricezione indica che siamo interessati a tutti i messaggi che eventualmente hanno codice numerico fino al valore 3, e ovviamente questi ci vengono dati in ordine dei loro valori numerici.

Con il comando IPCS su LINUX sto chiamando il listing di quante message queues esistono e che sono utilizzate da tante applicazioni

che usiamo e hanno ovviamente anche la loro chiave

```
PES-MESSAGES/UNIX> lpcs

----- Message Queues -----
key      msqid      owner      perms      used-bytes  messages
0x00000032 786432      francesco  666        0           0
0x00003105 131073      francesco  666        0           0
0x0000311c 163842      francesco  666        0           0
0x00003122 196611      francesco  666        0           0
0x00003127 229380      francesco  666        0           0
0x00003129 262149      francesco  666        0           0
0x0000313f 360454      francesco  666        0           0
0x00003140 393223      francesco  666        0           0
0x0000256c 557064      francesco  666        0           0
0x0000256f 589833      francesco  666        0           0
0x00002572 622602      francesco  666        0           0
0x00002575 655371      francesco  666        0           0
0x000025c8 819212      francesco  666        0           0
0x000025ca 851981      francesco  666        0           0
0x000025cc 884750      francesco  666        0           0
0x000025d2 917519      francesco  666        0           0

----- Shared Memory Segments -----
key      shmid      owner      perms      bytes      nattch     status
0x00000000 19431431    francesco  600        4096       2         dest
```