

SYSTEM V IPC

È un acronimo che descrive le 3 forme di IPC:

• Code di messaggi:

Che hanno come caratteristiche:

- confini delimitati: I messaggi hanno lunghezza fissa
- selezione per tipo: Ogni messaggio ha un campo type che può essere usato per la selezione.

• Semafori:

Usati per la sincronizzazione con un intero mantenuto nel kernel.

• Memoria Condivisa

Condivisione dello stesso segmento di mem. Operazione veloce

OPERAZIONI SU IPC

Operazioni che si possono fare sono:

CREAZIONE / APERTURA

Tutte e 3 le IPC mettono una get per la loro creazione.

Ogni get ha bisogno di una chiave, che servirà a distinguere tra creazione e apertura.

msgget()

Messaggi

semget()

Semafori

shmget()

Shared memory

Ogni elemento può anche specificare se è il creatore, se 2 cercano di farlo va in errore

ELIMINAZIONE

L'eliminazione di un'istanza si fa con il comando `ctl` seguito dal flag `IPC_RMID`

L'eliminazione varia in base all'IPC:

- immediata: Per semafori e code di mess.
- non imm: Per file sharing, che avverrà quando tutti hanno chiuso il file.

Tutti gli oggetti rimangono attivi fin quando non sono

```
id = msgget(key, IPC_CREAT | S_IRUSR | S_IWUSR);
if (id == -1)
    errExit("msgget");

// come per tutte le altre
// systcall get, key è il primo
// argomento, e l'identificatore
// è restituito come risultato
// della funzione.

// Specifichiamo i permessi di
// accesso al nuovo oggetto
// come ultimo argomento
// (flags), utilizzando le costanti
// elencate qui a fianco

S_IROD 000 Group-read
S_IWGR 020 Group-write
S_IROWR 060 Group-read/write
S_IROTH 04 Other-read
S_IWOTH 02 Other-write
S_IROWXH 062 Other-read/write
```

Creating/opening a System V IPC object

```
id = msgget(key, IPC_CREAT | S_IRUSR | S_IWUSR);
if (id == -1)
    dderrExit("msgget");

// Un processo che desideri accedere allo stesso oggetto
// IPC esegue una chiamata get specificando la stessa key per
// ottenere lo stesso identificatore per quell'oggetto.

// Se non esiste un oggetto IPC corrispondente alla key, ed è
// stata specificata la costante IPC_CREAT come parte dei flags,
// la get crea un nuovo oggetto IPC.

// Un processo può garantire di essere il creatore di un
// oggetto IPC specificando il flag IPC_EXCL.

// Se il flag IPC_EXCL è specificato, l'oggetto IPC corrispondente
// alla key esiste già, la get fallisce con l'errore EXIST.
```

Cancellazione di oggetti IPC

```
if (shmctl(id, IPC_RMID, NULL) == -1)
    errExit("shmctl");

// La system call id (msgctl(), semctl(), shmctl()) per
// ciascun meccanismo di IPC esegue un gran numero
// di operazioni di controllo sull'oggetto.

// Mentre molte di queste operazioni sono specifiche
// dei vari meccanismi di IPC, alcune sono comuni a
// tutti.

// Per esempio, una generica operazione di controllo
// è IPC_RMID, utilizzata per cancellare un oggetto.
```

Daniele Radicioni - Laboratorio di Sistemi Operativi, corso A - turno T1

direttamente cancellati oppure il sistema si spegne.

GENERAZIONE CHIAVI >

Le chiavi sono di tipo key_t. Per poterne generare altre chiavi univoche si può:

- Generare un numero casuale → Condiviso in header
- Usare il flag IPC_PRIVATE nelle get();
- Usare la funzione ftok(); che probabilmente genererà una chiave unica.

Generazione di key con IPC_PRIVATE

```
id = msgget(IPC_PRIVATE, S_IRUSR | S_IWUSR);
```

- In questo caso non è necessario specificare i flag IPC_CREAT o IPC_EXCL.
- Questa tecnica è particolarmente utile in applicazioni con molti processi in cui il processo padre crea l'oggetto IPC prima di eseguire la fork(), con il risultato che il figlio eredita l'identificatore dell'oggetto IPC.
- È possibile utilizzare questa tecnica anche in applicazioni client-server (che coinvolgono processi non collegati), ma i client devono avere un mezzo per ottenere gli identificatori degli oggetti IPC creati dal server (e viceversa).
- Per esempio, dopo avere creato un oggetto IPC, il server potrebbe scrivere il proprio identificatore su un file, che potrebbe essere letto dai client.



Daniele Radicioni - Laboratorio di Sistemi Operativi, corso A - turno II

Generazione di key con ftok()

```
#include <sys/ipc.h>
key_t ftok(char *pathname, int proj);
```

Returns integer key on success, or -1 on error

- Questo valore di key è generato dal pathname fornito e dal valore proj utilizzando un algoritmo definito a livello di implementazione.
- Nella generazione della key, ftok() utilizza il numero inode piuttosto che il nome del file.
- Poiché l'algoritmo della ftok() dipende dal numero inode, il file in questione non dovrebbe essere rimosso e ricreato mentre l'applicazione è in esecuzione, poiché è probabile che il file sia ricreato con un diverso numero inode.



Daniele Radicioni - Laboratorio di Sistemi Operativi, corso A - turno II

Generazione di key con ftok()

- Il fine del valore proj è consentire di generare diverse key a partire dallo stesso file, è utile quando un'applicazione deve creare vari oggetti IPC dello stesso tipo.
- Storicamente, l'argomento proj era di tipo char, ed è spesso specificato come tale nelle chiamate a ftok().

```
key_t key;
int id;
key = ftok("./mydir/myfile", 'x');
...
id = msgget(key, IPC_CREAT | S_IRUSR | S_IWUSR);
...
```

Daniele Radicioni - Laboratorio di Sistemi Operativi, corso A - turno II

18

ASSOCIATED OBJECT >

Ogni IPC ha una struct che lo rappresenta. Al momento della get(); viene allocata un'istanza gestibile col comando

*ctl():

- Ottenerne Oggetto: (copy)

specificando come flag IPC_STAT

- Impostare Valori:

Specificando IPC_SET

PERMESSI

Nella struct è possibile specificare permessi nel campo "mode".

Sono sempre suddivisi in 3 gruppi (user, group, others) come i file:

Si possono inoltre usare ipcs e ipcrm come ls ed rm per gli IPC.

Associated Object Permissions

- Il campo mode della sottostruzione ipc_perm contiene i permessi per l'oggetto IPC. I permessi sono inizializzati utilizzando i 9 bit più bassi specificati nei flag della syscall get, ma possono essere modificati successivamente utilizzando l'operazione IPC_SET.
- Come i permessi sono divisi in tre categorie: owner (o user), group, e other, ed è possibile specificare diversi permessi per ogni categoria.



Daniele Radicioni - Laboratorio di Sistemi Operativi, corso A - turno II

21

comandi ipcs and ipcrm

- I comandi ipcs e ipcrm sono analoghi ai comandi ls e rm per i file.

```
$ ipcs
----- Shared Memory Segments -----
key      shmid  owner   perms        bytes  nattch  status
0x60731d8  262147
----- Semaphores Arrays -----
key      semid  owner   perms        nsems
0x5070c00  cecilia 660      6
0x5070c00  britta   660      1
----- Message Queues -----
key      msqid  owner   perms        used-bytes messages
0x70705958  229376  cecilia 620      12          2
```

Daniele Radicioni - Laboratorio di Sistemi Operativi, corso A - turno II

22

comandi ipcs and ipcrm

- Di default, per ciascun oggetto, ipcs visualizza la key, l'identificatore, l'owner e i permessi (espressi in notazione ottale), seguiti da informazioni specifiche per l'oggetto:

- per la memoria condivisa, ipcs visualizza la dimensione della regione di memoria condivisa, il numero di processi che attualmente hanno la regione attaccata ai propri spazi di indirizzi e dei flag di stato.
- per i semafori, ipcs visualizza la dimensione del set di semafori.
- per le code di messaggi, ipcs visualizza il numero totale di byte di dati e il numero di messaggi presenti nella coda.

Daniele Radicioni - Laboratorio di Sistemi Operativi, corso A - turno II

comandi ipcs and ipcrm

- Il comando ipcrm cancella un oggetto IPC object. La forma generale di questo comando è la seguente:

```
$ ipcrm -X key
$ ipcrm -x id
```

- specifichiamo key (oppure l'identificatore id), e la lettera X (oppure x) è una q per le message queues, da una s per i semaphores, o da una m per la shared memory.

23

Daniele Radicioni - Laboratorio di Sistemi Operativi, corso A - turno II

24

MESSAGE QUEUE >

CREAZIONE / APERTURA

Creating/Opening a Message Queue

```
#include <sys/types.h> /* For portability */
#include <sys/msg.h>
int msgget(key_t key, int msgflg);

Returns message queue identifier on success, or
-1 on error
```

- l'argomento **key** è una chiave generata utilizzando un numero casuale, **IPC_PRIVATE** o **ftok()**;
- l'argomento **msgflg** è una maschera di bit che specifica i permessi da associare a una nuova coda di messaggi. Se la coda esiste già, permette di verificarne i permessi.

Daniele Radicioni - Laboratorio di Sistemi Operativi, corso A - turno T1

28

msgflg argument

- zero o più** fra i seguenti flag possono essere concatenati in OR () nel msgflg per controllare la msgget():
 - IPC_CREAT**: se non esiste una coda con la key specificata, crea una nuova coda;
 - IPC_EXCL**: se è presente anche **IPC_CREAT**, e una coda con la key specificata esiste già, restituisci un fallimento con errore **EXIST**.



Daniele Radicioni - Laboratorio di Sistemi Operativi, corso A - turno T1

29

esempio di invocazione msgget()

```
// creo una coda di messaggi tramite msgget
if((m_id = msgget(ftok("f_name.c",1), IPC_CREAT))<0)
    errExit("msgget error");

if((m_id = msgget(IPC_PRIVATE, 0644 ))<0)
    errExit("msgget error");
```



Daniele Radicioni - Laboratorio di Sistemi Operativi, corso A - turno T1

31

Segue il discorso generico sulla creazione

Condivisione key

La condivisione della chiave può avvenire come:

- File header condiviso

Con la chiave, che tutti i processi dovranno usare

- Condivisione padre-figli

Si può usare una **getppid()** nei processi figli, oppure usare i dati condivisi

INVIO DEI MESSAGGI

Si inviano con la sys call **msgsnd** che ha come parametri:

- id della message queue
- msgp: Un puntatore alla

struct rappresentante il messaggio. Può essere custom ma deve avere la forma:

- msgsiz: La dim. di mtext in byte
- msgflg: maschera di bit usata per altri comportamenti custom

per altri comportamenti custom

esempio di utilizzo della **msgsnd()**

```
struct queue q;
int m_id;

// ... inizializzazione di m_id e q ...

if(msqnd(m_id,
    &q, // la dimensione della struttura si sottra la dimensione del membro type
    &msg, // ... sizeof(m) - sizeof(type))
    &msg, // ... sizeoff(msg) - sizeoff(type), IPC_NOWAIT) <0){
    perror("Message send Error\n");
    exit(1);
} printf("Message send Error\n");
exit(1);
```

Daniele Radicioni - Laboratorio di Sistemi Operativi, corso A - turno T1

37



Daniele Radicioni - Laboratorio di Sistemi Operativi, corso A - turno T1

29

condivisione della key

- La condivisione della key può avvenire in diversi modi:

- In un file di definizioni **header.h**, incluso da tutti i processi che devono usare la stessa coda:

#define MYKEY 1234

- Il processo responsabile per l'allocatione della coda eseguirà:

int q_id = msgget(MYKEY, IPC_CREAT | 0644);

- Un processo che deve usare la coda associata a **MYKEY** eseguirà una chiamata come la seguente:

int q_id = msgget(MYKEY, 0);

- Se la coda associata a **MYKEY** esiste, viene restituito il suo identificatore, altrimenti viene restituito -1

Daniele Radicioni - Laboratorio di Sistemi Operativi, corso A - turno T1

32

condivisione della key

- Se la coda viene usata da un gruppo di processi fratelli, ossia creati tutti dallo stesso padre, è possibile sfruttare questa caratteristica così:

int q_id = msgget(getppid(), ...);

- Se la coda viene usata da processi in relazione **padre-figlio**, si può sfruttare il fatto che un figlio eredita copia delle variabili da padre:

```
qid = msgget ( IPC_PRIVATE, ... );
p = fork();
if (p) { ... usa qid ... }
else { ... usa qid ... } // —figlio —
```

Daniele Radicioni - Laboratorio di Sistemi Operativi, corso A - turno T1

33

```
#include <sys/types.h> /* For portability */
#include <sys/msg.h>
int msgsnd( int msgid, const void *msgp,
            size_t msgsiz, int msgflg );
```

Returns 0 on success, or -1 on error

- La system call **msgsnd()** scrive un messaggio su una coda di messaggi;
- Per inviare un messaggio con la **msgsnd()**, è necessario assegnare il membro **mtype** della struttura a un valore maggiore di 0 e copiare i dati da trasmettere nei membri della struttura.
- L'argomento **msgsz** specifica il numero di bytes contenuti nel membro **mtext** della struttura.



Daniele Radicioni - Laboratorio di Sistemi Operativi, corso A - turno T1

35

```
struct mymsg {
    long mtype; /* Message type */
    char mtext[]; /* Message body */
}
```

```
#include <sys/types.h> /* For portability */
#include <sys/msg.h>
int msgsnd( int msgid, const void *msgp,
            size_t msgsiz, int msgflg );
```

Returns 0 on success, or -1 on error

- L'ultimo argomento, **msgflg**, è una maschera di bit dei flag che controllano l'operazione di **msgsnd()**. È definito solo un flag:
- IPC_NOWAIT**: Consente di eseguire una "nonblocking send". Di norma, se una coda è piena, **msgsnd()** si blocca finché non si libera abbastanza spazio per il messaggio che si desidera aggiungere. Se è specificato questo flag, la **msgsnd()** restituisce immediatamente con l'errore **EAGAIN**.

Daniele Radicioni - Laboratorio di Sistemi Operativi, corso A - turno T1

36

RICEZIONE DI MESSAGGI

La ricezione si specifica con msgrcv

- id: della message queue
- msgp: Il puntatore in cui salvare la struct del messaggio in send
- maxmsgsz: La dimensione massima del messaggio
- msgtyp : Definisce il tipo di ricerca del messaggio
 - =0: Prende il 1° della coda
 - >0: Prende il 1° con lo stesso mtype
(utile per evitare race conditions)
 - <0: Si forma una coda con priorità in cui si prende il 1° con val. min del valore assoluto
- msgflg: Contiene flag aggiuntivi specifici in or

```
#include <sys/types.h> /* For portability */
#include <sys/msg.h>
ssize_t msgrcv(int msgid, void *msgp,
               size_t maxmsgsz, long msgtyp, int msgflg );
```

Returns number of bytes copied into mtext field, or -1 on error

- La capienza del membro mtext del buffer msgp è espresso dall'argomento maxmsgsz.
- Se il corpo del messaggio da rimuovere dalla coda supera maxmsgsz bytes, nessun messaggio viene rimosso dalla coda, e la msgrcv() fallisce con errore E2BIG.

Daniele Radicioni - Laboratorio di Sistemi Operativi, corso A - turno T1

38

Utilizzo di msgtyp

- Non necessariamente i messaggi vengono letti nell'ordine con cui sono stati scritti e inviati alla coda. È possibile selezionare utilizzando il valore contenuto nel membro mtype. Questa selezione è controllata dall'argomento msgtyp:
- Se msgtyp è uguale a 0, viene prelevato il primo messaggio dalla coda e restituito al chiamante;
- se msgtyp è maggiore di 0 viene prelevato il primo messaggio il cui mtype è uguale a msgtyp e restituito al chiamante;
- specificando diversi valori per msgtyp, vari processi possono leggere da una coda di messaggi senza competere (racing) per leggere gli stessi messaggi. Una tecnica utile è quella in cui un processo seleziona messaggi contenenti il proprio process ID.

39

Utilizzo di msgflg

- Se msgtyp è minore di 0, la coda è trattata come una coda con priorità. Viene prelevato e restituito per primo il messaggio con il minimo mtype minore o uguale al valore assoluto di msgtyp.

msgrcv(id, msgp, maxmsgsz, -300, 0);

queue position	Message type (mtype)	Message body (mtext)
1	300	...
2	100	...
3	200	...
4	400	...
5	100	...

Daniele Radicioni - Laboratorio di Sistemi Operativi, corso A - turno T1

40

```
ssize_t msgrcv(int msgid, void *msgp,
                size_t maxmsgsz, long msgtyp, int msgflg);
```

- L'argomento msgflg è una maschera di bit formata mettendo in OR zero o più flag:

- IPC_NOWAIT. Esegue una ricezione 'nonblocking'. Normalmente, se sulla coda non sono presenti messaggi con il msgtyp specificato, msgrcv() si blocca fino a quando tale messaggio non diviene disponibile. Specificando il flag IPC_NOWAIT comporta che in questo caso la msgrcv() ritorni immediatamente con errore ENOMSG.

- MSG_NOERROR. Di default, se la dimensione dell'mtext eccede lo spazio disponibile (definito dall'argomento maxmsgsz), msgrcv() fallisce. Se viene specificato il flag MSG_NOERROR, la msgrcv() rimuove il messaggio dalla coda, ne tronca l'intervallo a maxmsgsz bytes, e lo restituisce al chiamante. I dati troncati sono persi.

Daniele Radicioni - Laboratorio di Sistemi Operativi, corso A - turno T1

41

CONTROLLO DELLA CODA

Si gestisce tramite msgctl che accetta:

- id
- cmd: Definisce il comando di gestione
- buf: Il puntatore alla eventuale struct di gestione

Message Queue Control Operations

```
#include <sys/types.h> /* For portability */
#include <sys/msg.h>

int msgctl(int msgid, int cmd, struct msqid_ds *buf);
```

Returns 0 on success, or -1 on error

- L'argomento cmd specifica l'operazione da eseguire sulla coda.

- IPC_STAT. Copia la struttura msqid_ds nel buffer puntato da buf.

- IPC_SET. Aggiorna i membri della struttura msqid_ds associata con questa coda di messaggi, utilizzando i valori presenti nel buffer puntato da buf.

Daniele Radicioni - Laboratorio di Sistemi Operativi, corso A - turno T1

Associated Data Structure

- Ogni coda di messaggi ha associata una struttura msqid_ds:

```
struct msqid_ds {
    struct ipc_perm ipc_perm; /* Ownership & permissions */
    time_t msg_stime; /* Time of last message */
    time_t msg_rtime; /* Time of last receive */
    time_t msg_ctime; /* Time of last change */
    unsigned long __msg_cbytes; /* Number of bytes in queue */
    __msgnum_t msg_qnum; /* Number of messages in queue */
    __msglen_t msg_qbytes; /* Maximum bytes in queue */
    pid_t msg_lpid; /* PID of last msgsnd() */
    pid_t msg_rpid; /* PID of last msgrcv() */
};
```

Daniele Radicioni - Laboratorio di Sistemi Operativi, corso A - turno T1

```
struct msqid_ds {
    struct ipc_perm ipc_perm;
    uid_t uid; /* ushort: Owner's user ID */
    gid_t gid; /* ushort: Owner's group ID */
    uid_t cuid; /* Creator's user ID */
    gid_t cgid; /* Creator's group ID */
    unsigned short mode; /* permissions */
    unsigned short __seq; /* Sequence number */
};
```

Daniele Radicioni - Laboratorio di Sistemi Operativi, corso A - turno T1

Message Queue Control Operations

```
#include <sys/types.h> /* For portability */
#include <sys/msg.h>

int msgctl(int msgid, int cmd, struct msqid_ds *buf);
```

Returns 0 on success, or -1 on error

- L'argomento cmd specifica l'operazione da eseguire sulla coda.

- IPC_RMID. Rimuove immediatamente la coda di messaggi e la sua associata struttura dati msqid_ds.

- Tutti i messaggi presenti sulla coda vanno persi e qualsiasi processo lettore o scrittore in attesa sulla coda è immediatamente svegliato, con la msgsnd() o la msgrcv() che falliscono con errore EIDRM. Il terzo argomento msgctl() è ignorato per questa operazione.

Daniele Radicioni - Laboratorio di Sistemi Operativi, corso A - turno T1

43