Esempio Messaggi (manca)

lunedì 8 maggio 2023 12:25

ESEMPIO

```c
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <string.h>


#define TAGLIA 128

#define REGULAR_MESSAGE 1
#define SYNCH_MESSAGE   2

typedef struct{
        long mtype;
        char mtext[TAGLIA];
    } msg;

int ris;

void produttore(int ds_coda) {/* lettura delle stringhe e spedizione sulla coda */
        msg messaggio;
        printf("process %d - digitare le stringhe da trasferire (quit per terminare):\n",getpid());
    fflush(stdout);
        do {
            scanf("%s",messaggio.mtext);
            messaggio.mtype = REGULAR_MESSAGE;
            ris = msgsnd(ds_coda, &messaggio, TAGLIA, IPC_NOWAIT);
            if ( ris == -1 ){
            printf("process %d - errore nella chiamata msgsnd\n",getpid());
            fflush(stdout);
        }
        } while( (strcmp(messaggio.mtext,"quit") != 0));

#ifdef SYNCHRONIZED
        messaggio.mtype = SYNCH_MESSAGE;
        ris = msgsnd(ds_coda, &messaggio, 0 , IPC_NOWAIT);
#endif
    exit(0);
 }

void consumatore(int ds_coda) { /* ricezione delle stringhe e visualizzazione sullo standard output */
        msg messaggio;

#ifdef SYNCHRONIZED
        ris = msgrcv(ds_coda, &messaggio, 0, SYNCH_MESSAGE, 0);
        if ( ris == -1 ){
        printf("process %d - errore nella chiamata msgrcv\n",getpid());
        fflush(stdout);
        }
#endif

        do {
            ris = msgrcv(ds_coda, &messaggio, TAGLIA, REGULAR_MESSAGE, 0);
            if ( ris == -1 ){
            printf("process %d - errore nella chiamata msgrcv\n",getpid());
            fflush(stdout);
        }
            printf("process %d - %s\n", getpid(),messaggio.mtext);
        fflush(stdout);
        } while( (strcmp(messaggio.mtext,"quit") != 0));
        exit(0);
}
```

```c
int main(int argc, char *argv[]) {
        int  des_coda, status;
    long chiave = 40;//try different keys and then check with ipcs
    //long chiave = IPC_PRIVATE;

        des_coda = msgget(chiave, IPC_CREAT|IPC_EXCL|0666);
        if ( des_coda == -1 ){
            printf("process %d - errore nella chiamata msgget\n",getpid());
        fflush(stdout);
        exit(EXIT_FAILURE);
    }
        if ( fork()!=0 ) {
            if ( fork()!=0 ) {
                wait(&status);
                wait(&status);
            }
            else produttore(des_coda);
        }
        else consumatore(des_coda);

        ris = msgctl(des_coda, IPC_RMID, NULL);

        if ( ris == -1 ){
            printf("process %d - errore nella chiamata msgctl",getpid());
        fflush(stdout);
    }

    return 0;

}
```

L'applicazione .c eseguirà una fork e genererà un clone di sé stessa. Avremo un parent e un child, e questi due utilizzeranno, per scambiare informazioni, una message queues.
All'interno della struttura main() avremo al suo interno la creazione di una message queues con una certa chiave e quest'ultima è pari al codice numerico 40, ed è dichiarata long. Andiamo a creare (IPC_CREAT) ed in maniera esclusiva (IPC_EXCL), il che implica dire che se una message queue con questa stessa chiave già esiste, des_coda deve avere come risultato -1, perché ci deve fornire errore la msgget(). Se non c'è stato un errore, eseguiamo una prima fork(), eseguiamo una seconda fork() se siamo il parent, e poi attendiamo i due processi figli (81,82,rig). I due processi figli, nel loro address space, hanno la copia del codice di des_coda: siamo cloni e quindi all'interno degli address space dei figli le variabili del parent (almeno per quello che era il valore nel momento in cui il clonaggio è avvenuto) vengono copiate.
Quindi in uno chiamiamo la procedura produttore() e lavorerà con il codice della coda che passiamo in input, e poi consumatore() sarà una procedura che in input prenderà lo stesso codice della coda.
Quando chiamiamo il produttore o il consumatore, siamo in uno dei due child, mentre nel parent attendiamo entrambi, e chiaramente il parent dopo aver atteso entrambi esce da questo IF-ELSE e rimuove la coda che non serve più (riga 88). Chiaramente controlliamo il valore di ritorno nella riga 90 e se è -1 andiamo ad indicare un errore, altrimenti la rimozione è andata a buon fine.

Produttore() ha ricevuto in input il codice numerico associato a questa message queue, va a mandare una print sullo std:out in cui indica il pid del processo che sta eseguendo, e poi chiedo le stringhe da trasferire.
Quindi stiamo cercando di prendere una stringa, impacchettarla in un messaggio, ed inserirlo nella coda. Poi prenderà un'altra stringa, la impacchetterà e la inserirà ancora nella coda, e così via..!
Quindi per ogni stringa che lui riceve in input, la impacchetta nella stessa area di memoria d'appoggio, e poi spedisce il messaggio: tanto poi avviene la copia del messaggio e si può utilizzare la stessa area di memoria per andare ad acquisire un'altra stringa S' all'interno di quell'area di memoria, che poi verrà spedita all'interno di un messaggio.
Quindi stiamo inserendo all'interno della coda tutti i messaggi con stringhe differenti in modo tale che poi queste stringhe possano essere estratte dall'altro lato, ovvero dal consumatore P'.
Abbiamo quindi un DO-WHILE dove all'interno andiamo in scanf() a cercare una stringa sullo standard input. Questa stringa la scriviamo in messaggio.mtext (rig.29), dove messaggio è una struttura dati e mtext è un campo all'interno di questa struttura. Abbiamo definito un tipo, che è il tipo msg in cui al suo interno abbiamo la struttura con il campo long del tipo e poi il char mtext di una certa taglia. E quando noi abbiamo una variabile messaggio di tipo msg la variabile è strutturata in base alla struct.Noi stiamo prendendo la stringa e caricarla nell'array. Poi andiamo a popolare il campo mtype del messaggio con REGULAR MESSAGE, che ha codice numerico 1, e poi infine chiamiamo la msgsnd: abbiamo popolato tutto il messaggio, sia il codice numerico che rappresenta il tipo e sia la stringa che rappresenta l'effettivo contenuto informativo. Mandiamo il messaggio andando a specificare il puntatore a tutta la struttura che ospita il messaggio (secondo parametro), poi specifichiamo taglia - perché dobbiamo specificare solo la TAGLIA addizionale rispetto al TIPO - e infine andiamo a chiamare un'operazione NON BLOCCANTE, ossia IPC_NOWAIT. Tutta questa cosa va avanti finchè non prendiamo in input la stringa QUIT.
Consumatore() deve consumare questi messaggi, quindi deve anche lui eseguire un do-while in cui all'interno chiama proprio la ricezione utilizzando come primo parametro il descrittore di coda, indicando l'area di memoria in cui vuole ricevere il messaggio, al più TAGLIA byte, E POI ovviamente chiamiamo la ricezione del regular_message BLOCCANTE, perché 0 come ultimo parametro indica default e poi stampiamo il messaggio.
Nell'esecuzione il parent è in attesa dei due processi figli. I due processi figli sono tali per cui uno è in attesa che l'altro digiti stringhe, e l'altro in attesa di un messsaggio su quella message queue.

```
PES-MESSAGES/UNIX/baseline-msg-queue-example> ./a.out
process 11885 - digitare le stringhe da trasferire (quit per terminare):
ciao
process 11884 - ciao
mgrwkgm
process 11884 - mgrwkgm
;relmgelrkmg
process 11884 - ;relmgelrkmg
'ewl,mfew;lm
process 11884 - 'ewl,mfew;lm
quit
process 11884 - quit
```

Se ora listo tutte le code ottengo la message queues con codice numerico 28 in esadecimale, che sarebbe 40, che correntemente avevamo installato. Quindi se io faccio finire il programma con CTR + C ho terminato in maniera anormale e quindi se vado in IPCS a listare ottengo la message queue creata.

```
------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages
0x00000028 786432     francesco  666        0            0
```

```
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
0x00000028 983056     francesco  666        0            0

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0x00000000 19431431   francesco  600        4096       2          dest
0x00000000 12681233   francesco  600        142480     2          dest
0x00000000 20545557   francesco  600        6379080    2          dest
0x00000000 14811170   francesco  600        7795200    2          dest
--More--
```

RIMOZIONE DELLA CODA CON CODICE NUMERICO 40

```
0x00000028 983056     francesco  666        0            0

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0x00000000 19431431   francesco  600        4096       2          dest
0x00000000 12681233   francesco  600        142480     2          dest
0x00000000 20545557   francesco  600        6379080    2          dest
0x00000000 14811170   francesco  600        7795200    2          dest
francesco@linux-mxb5:~/git-web-site/FrancescoQuaglia.github.io/TEACHING/SIST
PES-MESSAGES/UNIX/baseline-msg-queue-example> ipcrm -Q 0x00000028
```

SINCRONIZZAZIONE. IL CONSUMER STA ASPETTANDO UN ALTRO TIPO DI MESSAGGIO E QUESTO TIPO DI MESSAGGIO ARRIVERÀ SOLTANTO AVREMO COMPLETATO TUTTE LE INFORMAZIONI IN INPUT, E QUINDI L'ALTRO PROCESSO RIUSCIRÀ AD OTTENERE LA POSSIBILITÀ DI ESTRARRE I MESSAGGI CHE RAPPRESENTANO EFFETTIVAMENTE I REGOLARI.
I MESSAGGI, ALL'INTERNO DI UNA CODA DI MESSAGGIO, POSSONO ESSERE USATI ANCHE PER MOTIVI DI COORDINAMENTO / SINCRONIZZAZIONE DELLE ATTIVITÀ ALL'INTERNO DELLE VARIE APPLICAZIONI.

```
PES-MESSAGES/UNIX/baseline-msg-queue-example> gcc example.c -DSYNCHRONIZED
francesco@linux-mxb5:~/git-web-site/FrancescoQuaglia.github.io/TEACHING/SISTEMI-OPERATIV
PES-MESSAGES/UNIX/baseline-msg-queue-example> ./a.out
process 12555 - digitare le stringhe da trasferire (quit per terminare):
ciao
a tutti
quit
process 12554 - ciao
process 12554 - a
process 12554 - tutti
process 12554 - quit
francesco@linux-mxb5:~/git-web-site/FrancescoQuaglia.github.io/TEACHING/SISTEMI-OPERATIV
PES-MESSAGES/UNIX/baseline-msg-queue-example>
```

Questo succede perché il consumatore potrebbe essere strutturato in modo tale che se syncronized è attivo, la prima cosa che cerca di attendere è (rig 49) un messaggio di tipo SYNC_MESSAGE, se questo non c'è e lui chiama la receive BLOCCANTE, chiaramente lui rimane in attesa. E quindi riceverà tutti i messaggi di tipo regolare solo dopo che sia arrivato questo sync_message.

Makefile

```
unsynchornized:
	gcc example.c -o example

synchronized:
	gcc example.c -o example -DSYNCHRONIZED
```

01:15:51 INIZIO WINDOWS MAIL SLOTS - 28_04_2021