Interprocess Comunication in sistemi a scambio di messaggi

Leonardo Bizzoni

November 4, 2023

Un meccanismo per lo scambio di messaggi deve prevedere almeno 2 operazioni: *send()* e *receive()*. I messaggi possono avere lunghezza fissa o variabile.

Se 2 processi vogliono scambiarsi messaggi è necessario un **canale di comunicazione** che può essere realizzato in molti modi:

- Comunicazione diretta/indiretta
- Comunicazione sincrona/asincrona
- Gestione automatica o esplicita del buffer di comunicazione

1 Comunicazione diretta

Ogni processo che intende comunicare deve **nominare** esplicitamente il mittente/destinatario del messaggio.

send(P, msg) dove P è il processo destinatario
receive(Q, msg) dove Q è il processo mittente

In questo schema il canale di comunicazione viene creato automaticamente ed è associato esattamente a 2 processi. Inoltre questo schema ha una **simmetria** nell'indirizzamento (*mittente e destinatario devono nominarsi a vicenda*).

1.1 Comunicazione asimmetrica

La comunicazione asimmetrica è una variante dello schema precendete dove il mittente deve nomiare il destinatario ma il destinatario non nomina il mittente.

send(P, msg) dove P è il processo destinatario
receive(ID, msg) dove ID conterrà il nome del processo mittente

2 Comunicazione indiretta

In questo schema un processo può comunicare con altri tramite un certo numero di **mailbox** e 2 processi possono comunicare sse condividono una mailbox.

send(A, msg)
receive(A, msg) dove A è la mailbox condivisa tra i 2 processi.

Inoltre un canale di comunicazione in questo schema può essere condiviso tra più di 2 processi, basta che tutti condividano la stessa mailbox.

Sorge però il problema di sapere quale processo riceverà il messaggio. Ci sono diverse soluzioni in base allo schema di comunicazione scelto:

- Si fa in modo che un canale sia associato al più a 2 processi
- Si consente l'esecuzione di *receive()* ad un solo processo alla volta
- Si lascia decidere all'OS (*solo un processo riceverà il messaggio*)

Una mailbox può appartenere ad un processo o all'OS:

- Se appartiene al processo allora essa è locata nello spazio di memoria associato al processo e quindi non si hanno dubbi su chi è il proprietario della mailbox il quale potrà solo inviare messaggi. Inoltre al termine del processo, la mailbox verrà eliminata e sarà necessario comunicarlo a tutti i processi che tentano di accedervi.
- Se appartiene all'OS allora ha vita autonoma e deve permettere ad un processo di **creare** la mailbox(*il che lo rende il proprietario*) e **rimuoverla**.

3 Sincronizzazione della comunicazione

L'effettiva comunicazione tra processi avviene tramite l'utilizzo delle primitive *send(), receive()* esse posso venire effettuate in maniera **sincrona** (*bloccante*) o **asincrona**.

3.1 Comunicazione sincrona

Il processo mittente attende che il processo ricevente o la mailbox riceva il messaggio.

Il processo ricevente attende di ricevere un messaggio (*direttamente da un processo o nella mailbox*).

3.2 Comunicazione asincrona

Il processo mittente invia il messaggio e continua la sua esecuzione.

Il processo ricevente riceve un messaggio valide (*se c'era un messaggio*) o un valore nullo.

4 Esercitazione

```
#include <sys/types.h>
#include <sys/wait.h>
#include <signal.h>
#include <stdio.h>

#include <unistd.h>

void signal_handler(int sig) {
  switch (sig) {
  case SIGINT:
    printf("Hai premuto C-c!\n");
    break;
  case SIGUSR1:
    printf("Segnale SIGUSR1 intercettato!\n");
    break;
  case SIGALRM:
    printf("Tempo scaduto.\n");
    break;
  }
}

int main(void) {
  pid_t pid = fork();

  if (pid > 0) {
```

```
    printf("Child PID: %d\n", pid);

    /* sleep(3); */
    /* kill(pid, SIGUSR1); */

    wait(NULL);
  } else if (pid == 0) {
    signal(SIGINT, signal_handler);
    signal(SIGUSR1, signal_handler);
    signal(SIGALRM, signal_handler);

    alarm(5);

    printf("Prima del segnale.\n");
    pause();
    printf("Dopo il segnale.\n");
  } else {
    perror("Fork failed");
  }
}
```