Pipe

Per comunicare tra processi che si trovano sulla stessa macchina, precisamente sullo stesso SO, (comunicazione interprocesso intramacchina) non vengono usati i socket, bensì i PIPES. Essi possono essere paragonati a dei file con istruzioni sequenziali.

I socket, come sappiamo, creano un canale bidirezionale, basato sul protocollo TCP. Invece le pipes stabiliscono un canale monodirezionale (un tubo, da qui il nome) formato da due capi in cui uno è scrittore ed uno è lettore.

Le informazioni, una volta lette, spariscono, quindi non disponibili per altri processi. Ciò non accade se lo scrittore riproduce il messaggio per tutti i lettori mancanti. Esistono anche meccanismi di ritrasmissione, ad esempio a seguito di un crash.

Le pipes, a livello implementativo, sono dei buffer di dimensione variabile, solitamente 4 kB. I dati da comunicare vengono scritti e letti da questa zona di memoria.
Si possono notare delle analogie con il problema PRODUTTORE – CONSUMATORE, dove i primi sono gli scrittori, i secondi sono i lettori e la pipe è il buffer. Il produttore si blocca quando il buffer è pieno, invece il consumatore entra in attesa con buffer vuoto.

La comunicazione tra i thread di uno stesso processo (interthread interprocesso) non necessitano di pipes; infatti possono usare lo spazio che condividono, eventualmente protetto da semafori.

Le pipe sfruttano i descrittori e quindi le operazioni read() e write(). Ne esisterà uno per la scrittura e uno per la lettura.

Esistono due tipi di legami tra processi:

1. Relazionati: processi legati da una fork() (Padre e figli) -> i descrittori vengono passati in maniera automatica, essi saranno infatti ereditati in modo che il figlio possa comunicare.
2. Non relazionati: processi creati da due chiamate diverse in uno stesso terminale o da terminali diversi. In questi casi vengono usate le Named Pipe (FIFO). Esse sono pipe con un nome univoco; sono molto simili ai named semafori in cui processi diversi vengono sincronizzati.

int pipe(int fd[2]); --> creazione di una pipe

L'argomento passato è un puntatore ad un array (buffer) di dimensione minima 2. Scrivo minima perché l'array potrebbe essere più grande, ma verranno sempre considerate solo le prime due posizioni. Infatti fd[0] rappresenta il descrittore di lettura della Pipe, invece fd[1] è quello di scrittura.

Essi possono essere usati con le funzioni read() e write().
Esistono varie interpretazioni del ritorno di una read(), esse dipendono dalle risorse usate:
- file: se essa ritorna con 0 allora significa che è stato letto tutto il contenuto
- socket: sostituita dalla recv(), se ha come return 0 allora l'altro end-point ha chiuso la connessione

Per le pipe si usa una interpretazione simile a quella usata nei socket.
Infatti per convenzione quando gli scrittori terminano il loro lavoro, e quindi concludono il "file" da inviare, chiudono il descrittore fd[1]; la read(), semplicemente, notifica l'evento di disconnessione.

Nel caso in cui uno scrittore continui a scrivere anche quando tutti i lettori hanno chiuso il proprio descrittore fd[0], riceve un segnale SIGPIPE, chiamato anche Broken-pipe.

Per evitare situazioni di deadlock, è necessario che i processi chiudano i descrittori che non usano più tramite una semplice close(). Ad esempio quando un figlio lettore eredita l'array, la prima cosa che deve fare è chiudere la sua copia fd[1] e poi iniziare a leggere da fd[0]. Se non accadesse ciò, l'evento che notifica la conclusione del messaggio, cioè quando tutti gli scrittori hanno chiuso il proprio descrittore, non potrebbe mai avvenire (il suo fd[1] rimarrebbe aperto), creando una lettura infinita e quindi una deadlock.

ESEMPIO: Trasferimento di stringhe tramite PIPE

```
#include <stdio.h>

#define Errore_(x) { puts(x); exit(1); }

int main(int argc, char *argv[]) {
    char messaggio[30];   int pid, status, fd[2];
    ret = pipe(fd);                                  /* crea una PIPE */
    if ( ret == -1 )                                 /* gestione di errore */
       Errore_("Errore nella chiamata pipe");
    pid = fork();                                    /* crea un processo figlio */
    if ( pid == -1 )
       Errore_("Errore nella fork");
    if ( pid == 0 ) {                                /* processo figlio: lettore */
       close(fd[1]);                                 /* il lettore chiude fd[1] */
       while( read(fd[0], messaggio, 30) > 0 )
/* la read prende come parametri: il descrittore,il buffer dove riversare il contenuto e il numero byte da
leggere al massimo -> si esce da questa chiamata quando il padre ha chiuso il suo descrittore */
             printf("letto messaggio: %s", messaggio);
       close(fd[0]);
}


/* processo padre: scrittore */

else {
    close(fd[0]);                                              /* chiude il descrittore di lettura */
    puts("digitare testo da trasferire (quit per terminare):");
    do {
         fgets(messaggio,30,stdin);
         write(fd[1], messaggio, 30);
/* la write prende come parametri il descrittore, il buffer di dove si trova effettivamente il messaggio e la
grandezza di questo (in byte) */
         printf("scritto messaggio: %s", messaggio);
    } while( strcmp(messaggio,"quit\n") != 0 );

    close(fd[1]);
    wait(&status);  attesa proceso figlio termini
}
}
```

Named Pipe (FIFO)

Come detto prima, è usata per i processi non relazionati. Funziona come i named semafori. Per FIFO si intende la modalità di estrazione, first in first out, la prima cosa che inserisco è la prima che faccio uscire. Ciò è l'essenza del canale, quello che viene inserito sequenzialmente dal primo capo, viene letto nello stesso ordine dal secondo.

Per prima cosa una named pipe deve essere creata tramite la seguente segnatura:
int mkfifo(char*name, int mode);

I parametri passati sono:

- name: puntatore ad una stringa che rappresenta il nome univoco
- mode: intero che specifica le modalità di creazione (bisogna crearla o no) e una maschera di bit che costituiscono i permessi di accesso

Per eliminare una FIFO bisogna chiamare la funzione unlink().

Solitamente, l'apertura di una FIFO è bloccante, nel senso che il processo che tenti di aprirla in lettura (scrittura) viene bloccato fino a quando un altro processo non la apre in scrittura (lettura). È possibile inibire questo comportamento aggiungendo il flag O_NONBLOCK al parametro mode passato alla system call open().
Come nel caso pipe, se un processo tenta di scrivere su una FIFO che non ha un lettore esso riceve il "segnale SIGPIPE"(Broken Pipe) da parte del sistema operativo.

Esempio: client/server tramite FIFO in maniera bidirezionale → gestione di due named pipe

```
/* Processo Server */

#include <stdio.h>

#include <fcntl.h>

typedef struct {
   long type;
   char fifo_response[20];
} request;
/* struttura REQUEST composta da un long che rappresenta il tipo di richiesta e il messaggio di massimo 20 Byte */

int main(int argc, char *argv[]){
   char *response = "fatto";
   int pid, fd, fdc, ret;
   request r;
   ret = mkfifo("serv", O_CREAT|0666);
/* crea una named pipe PERENNE (serv), verrà eliminata definitivamente solo quando il server smetterà di esistere, dove i client possono diventare scrittori e mandare le proprie richieste*/
   if ( ret == -1 ) {
      printf("Errore nella chiamata mkfifo\n");
      exit(1);

    }
/* gestione di errore */
   fd = open("serv",O_RDONLY);
/*il server apre la fifo serv in modalità lettura */
   while(1) {
/* inizia un ciclo while in cui il server prova a leggere la serv fino a che non vi è scritta una richiesta da parte del client*/
      ret = read(fd, &r, sizeof(request));
/* letto il contenuto, esso viene riversato in una variabile di tipo request (i messaggi scambiati sono strutturati) */
      if (ret != 0) {
         pid = fork();
/* dopo aver ottenuto il messaggio, il server crea un figlio che ha il compito di gestire la richiesta*/
         if (pid == 0) {
            printf("Richiesto un servizio (fifo di restituzione = %s)\n", r.fifo_response);
/* switch sul tipo di servizio */
            sleep(10);          /* emulazione di ritardo per il servizio */
```

```
            fdc = open(r.fifo_response,O_WRONLY);
/* il figlio apre la FIFO creata dal client in modalità scrittore, lì riporterà la risposta*/
            write(fdc, response, 20);
/*la risposta avviene tramite un messaggio statico (come richiesta)*/
            close(fdc);
/*viene chiuso il descrittore di scrittura così da far capire che ha terminato l'operazione */
            exit(0);

          }/* end if */

        } /* end if */

   }

}
```

/* Processo Client */ il suo obiettivo è ricevere messaggio di risposta dal server tramite una pipe TEMPORANEA

```
#include <stdio.h>

#include <fcntl.h>

typedef struct {
    long type;
    char fifo_response[20];
} request;
/*la struttura REQUEST è sempre uguale, ciò permette una comunicazione chiara, la conversazione avviene nella stessa lingua */

int main(int argc, char *argv[]) {
    int pid, fd, fdc, ret; request r; char response[20];
    printf("Selezionare un carattere alfabetico minuscolo: ");
/*il client chiede all'utente il nome da dare alla FIFO dove scriverà la risposta il server*/
    scanf("%s",r.fifo_response);
    if (r.fifo_response[0] > 'z' || r.fifo_response[0] < 'a' ) {
        printf("carattere selezionato non valido, ricominciare operazione\n");
        exit(1);
    }
    r.fifo_response[1] = '\0';
    ret = mkfifo(r.fifo_response, O_CREAT|0666);

/*creazione della FIFO di risposta */
    if ( ret == -1 ) {
        printf("\n servente sovraccarico - riprovare \n");
        exit(1);

    }
    fd = open("serv",O_WRONLY);
/*il client apre da scrittore la FIFO serv dove mette la richiesta ed il nome della FIFO creata per la rispsta */
    if ( fd == -1 ) {
        printf("\n servizio non disponibile \n");
        ret = unlink(r.fifo_response); exit(1);
    }
```

```
    write(fd, &r, sizeof(request));
/* il client scrive il messaggio (nome della FIFO + richiesta) N.B client e server devono essere d'accordo sulla struttura da utilizzare → request */
    close(fd);
/* appena finisce di scrivere chiude il descrittore di scrittura, il client è sicuro che il server riceverà la richiesta */
    fdc = open(r.fifo_response,O_RDONLY);
/* apre la pipe di risposta come lettore */
    read(fdc, response, 20);
/* legge e memorizza il messaggio nella zona di memoria response */
    printf("risposta = %s\n", response);
    close(fdc);
    unlink(r.fifo_response);
/* chiude il descrittore ed elimina in maniera definitiva la pipe creata dal client */
}
```