Sistemi operativi – laboratorio

Esercitazione 5: pipe

27 Aprile 2023 – 4 Maggio 2023

Ese_1:

Scrivere un programma che implementi il paradigma Produttore-Consumatore basato su PIPE.
Il processo Consumatore riceve da riga di comando il pathname di un file di testo. Successivamente, il processo Consumatore crea una PIPE, e un processo Produttore. Il processo Produttore legge il contenuto del file in chunk di massimo 100 caratteri. Ogni chunk letto viene inviato al processo Consumatore attraverso la PIPE. Il processo Consumatore stampa il contenuto del file ricevuto per mezzo della PIPE.

Ese_2:

Estendere Ese_1 affinché il processo Consumatore riceva da riga di comando una **lista** di pathname di file di testo. Per ogni file di testo, il processo Consumatore crea un processo Produttore, il quale invia il contenuto del file di testo al Consumatore attraverso un'unica PIPE.
La PIPE e' un bytes stream channel. Se due o piu' Produttori scrivono sulla stessa PIPE, come possiamo distinguere i dati scritti da un Produttore, dai dati scritti da un altro Produttore?

Soluzione:
Il Produttore antepone il numero di byte che intende scrive, prima dei dati effettivi.

```
                    ---------------------------------------------------------------------------
PIPE read end ← | 2 | byte-1 byte-2| 4| byte-1 byte-2 byte-3 byte-4| ----
                    ---------------------------------------------------------------------------
```

Il contenuto della PIPE sopra riportata mostra due chunk di dati.
Il primo chunk ha dimensione 2 bytes, mentre il secondo ha dimensione 4 bytes.

Consiglio:
definire la seguente struttura:

```
struct Item {
        ssize_t size;
        char value[MSG_BYTE];
};
```

Il campo *size* contiene il numero di byte che il processo Produttore intende scrivere sulla PIPE
Il campo *value* contiene i dati effettivi inviati tramite PIPE.

```
struct Item item;
// ... il Produttore inizializza item come sopra descritto
// dimensione chuck e chunk sono copiati nella PIPE in modo atomico tramite write()
write(PIPE, &item, item.size + sizeof(item.size));
```

Il processo Consumator, legge dalla PIPE prima il campo *size* e poi i dati.

```
ssize_t size;
read(PIPE, &size, sizeof(size)); // leggo dimensione chunk
read(PIPE, &buffer, size);
// leggo chunk
```

Ese_3:
Implementare tramite system call la seguente catena di comandi script:

```
ls -al | sort
```

Ese_3 b:
Implementare tramite system call la seguente catena di comandi script:

```
ls -al | cut -b27- | sort -n
```

Ese_4:
Scrivere un programma che simula una partita di ping-pong.

Il processo *Main* genera il sottoprocesso *Avversario*. I processi *Main* e *Avversario* comunicano attraverso PIPE. Per N volte (valore letto da riga di comando), il processo *Avversario* invia la stringa “pong” al processo *Main*, mentre *Main* invia la stringa “ping” al processo *Avversario*. Ad ogni stringa S ricevuta, i processi *Main* e *Avversario* stampano S a video. All’N-esima stringa ricevuta *Main* e *Avversario* terminano.

Vincolo: *Avversario* e *Main* sono sincronizzati in modo tale che ogni stringa “ping” sia sempre stampata a video prima della corrispondente stringa “pong”.

Consiglio:
1) La PIPE e’ un canale unidirezionale. Se due processi devono comunicare tramite PIPE, allora due PIPE sono necessarie.
2) Si ricorda che la system call *read()* offre implicitamente meccanismi di blocco per gestire la sincronizzazione della comunicazione, quindi non è necessario l’utilizzo di semafori.

Lo pseudo-codice della soluzione e’ quindi il seguente:

Main:
per N volte:
invia ‘ping’ tramite PIPE1
legge ‘pong’ tramite PIPE2
stampa ‘pong’

Avversario:
per N volte:
legge ‘ping’ tramite PIPE1
stampa ‘ping’

invia ‘pong’ tramite PIPE2