

Sistemi operativi - laboratorio

Esercitazione 3.2: pipe

30/03/2020

Ese_1: (lab)

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.

Ese_2: (lab)

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 è un bytes stream channel. Se due o più 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 scrivere, 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 chunk e chunk sono copiati nella pipe in modo atomico tramite write()
```

```
write(pipe, &item, item.size + sizeof(item.size));
```

Il processo Consumator, leggere prima il campo size, e poi i dati affetti dalla pipe.
ssize_t size;
read(pipe, &size, sizeof(size)); // leggo dimensione chunk
read(pipe, &buffer, size); // leggo chunk

Ese_3: (home)

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) Il vincolo riportato nel testo potrebbe indurre a pensare ad un meccanismo di sincronizzazione (per esempio ai semafori). Tuttavia, niente di tutto cio' e' necessario. La system call read() e' di per se bloccante se nessun dato e' presente nella pipe.

Il 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