Esercitazione [06]

Input/Output su Socket

Leonardo Aniello - aniello@dis.uniroma1.it

Daniele Cono D'Elia - delia@dis.uniroma1.it

Giuseppe Laurenza - laurenza@dis.uniroma1.it

Federico Lombardi - lombardi@dis.uniroma1.it

Sistemi di Calcolo - Secondo modulo (SC2)

Programmazione dei Sistemi di Calcolo Multi-Nodo

Corso di Laurea in Ingegneria Informatica e Automatica

A.A. 2016-2017

Sommario

- Soluzione esercizio con più produttori e più consumatori
- Obiettivi dell'esercitazione
- Descrittori in C
- Letture e Scritture su file
- Invio e Ricezione messaggi su socket

Esercizio

Many Producers/Many Consumers

- Estendere il sorgente `one_producer_one_consumer.c` per farlo funzionare con più produttori e/o più consumatori
- Soluzione
 - Più produttori (consumatori) possono trovarsi ad inserire nella (leggere dalla) stessa posizione, con la conseguente perdita (duplicazione*) di entry
 - Realizzare una mutua esclusione per l'accesso al buffer (e agli indici di lettura/scrittura) risolve il problema
 - Usare un semaforo binario per i consumatori ed uno per i produttori consente ai produttori e ai consumatori di lavorare in parallelo!
 - Le race condition tra produttori e tra consumatori sono invece già risolte dagli altri due semafori
 - Codice: `many_prod_many_cons.c`

* Oltre alla duplicazione, può verificarsi la perdita di un'entry se il `read_index` viene incrementato due volte: in tal modo un'entry verrebbe «saltata»

Obiettivi Esercitazione [6]

- Imparare ad effettuare operazioni di input e output usando i descrittori in UNIX
 - Lettura/scrittura su file
 - Invio/ricezioni messaggi su socket

Descrittori in UNIX

- I *file descriptor* (FD) sono un'astrazione per accedere a file o altre risorse di input/output come pipe e socket
- Ogni processo ha una *tabella dei descrittori* associata
 - Standard input 0
 - Standard output 1
 - Standard error 2
- Le operazioni di apertura (o creazione) di una risorsa di input/output sono legate al tipo della risorsa stessa
 - `open ()` per i file
 - `socket ()` o `accept ()` per le socket
 - `pipe ()` per le pipe (prossimamente...)

Letture con i descrittori

- La funzione `read()` è definita in `unistd.h`

```
ssize_t read(int fd, void *buf, size_t nbyte);
```

- `fd`: **descrittore della risorsa**
- `buf`: **puntatore al buffer dove scrivere il messaggio letto**
- `nbyte`: **numero massimo di byte da leggere**

Ritorna il numero di byte realmente letti, o `-1` in caso di errore

- Per i file, ritorna `0` in caso di end-of-file
- Per le socket, ritorna `0` in caso di connessione chiusa
- Il buffer deve essere dimensionato per contenere `nbyte` byte
 - Tale dimensione dipende dall'applicazione
 - Formato del file da leggere
 - Messaggi da scambiare in un protocollo di comunicazione

Lecture con i descrittori

Gestione degli interrupt

- L'esecuzione della funzione `read()` ha una certa durata
 - Per i file, richiede il tempo necessario per leggere fino a `nbytes` byte
 - Per le socket, il tempo necessario dipende dall'altro endpoint
- Nel tempo tra l'invocazione ed il termine della `read()`, la chiamata può essere interrotta da un segnale
 - Se l'interruzione avviene prima di riuscire a leggere qualsiasi dato (zero byte letti), la `read()` ritorna `-1` ed `errno` viene settato a `EINTR`
 - Se l'interruzione avviene dopo aver letto qualche dato (byte letti > zero), la `read()` ritorna il numero di byte letti fino a quel momento
- Una corretta implementazione deve riconoscere queste situazioni ed invocare di nuovo la `read()` per ritentare/completare la lettura

Lecture con i descrittori

Esempio in C

```
while(<not all bytes have been read>) {
    // read from fd up to n bytes and store into buf
    int ret=read(fd, buf, n);

    // no more bytes to read, quit
    if (ret==0) break;

    if (ret==-1) {
        if (errno==EINTR) continue; /* interrupted before
                                     reading any byte, retry */
        exit(EXIT_FAILURE); // an error occurred...
    }

    /* if interrupted when less than n bytes were read, pay
       attention to where to write on buf on resume! */
    <do something with read bytes>
}
```

Scritture con i descrittori

- La funzione `write()` è definita in `unistd.h`

```
ssize_t write(int fd, const void *buf, size_t nbyte);
```

- `fd`: descrittore della risorsa
- `buf`: puntatore al buffer contenente il messaggio da scrivere
- `nbyte`: numero massimo di byte da scrivere

Ritorna il numero di byte realmente scritti, o `-1` in caso di errore

- Gestione degli interrupt analoga alla `read()`
 - In caso di interrupt prima di aver scritto il primo byte, viene ritornato `-1` e settato `errno` a `EINTR`
 - In caso di interrupt dopo aver scritto almeno un byte, viene ritornato il numero di byte realmente scritti

Scritture con i descrittori

Esempio in C

```
while(<not all bytes have been written>) {
    // write to fd up to n bytes from buf
    int ret=write(fd, buf, n);

    if (ret==-1) {
        // interrupted before writing any byte, retry
        if (errno==EINTR) continue;

        // an error occurred...
        exit(EXIT_FAILURE);
    }

    /* if interrupted when less than n bytes were
       written, pay attention to where you start
       reading from in the buffer on resume */
    <do something>
}
```

Esercizio: copiare un file in C

- Sorgente da completare: `copy.c`
- Argomenti
 - File sorgente S
 - File destinazione D
 - Dimensione B del batch di lettura/scrittura (opzionale, default 128 byte)
- Semantica
 - Effettuare una copia di S in D tramite una sequenza di letture da S e scritture in D a blocchi di B byte per volta
- Esercizio: completare il codice dove indicato
 - Per testare la propria soluzione è disponibile lo script `test.sh`

Invio e Ricezione messaggi su Socket

- Scenario: architettura client-server su protocollo TCP
 - Il server è in ascolto su una certa porta nota
 - Il client effettua una connessione verso il server su quella porta
- Una volta aperta una connessione TCP tra due processi, ogni processo può accedervi tramite un descrittore
- L'invio e la ricezione di messaggi tramite socket vengono gestiti in maniera analoga a `write()` e `read()` su file
 - È necessario disporre di un descrittore della socket
 - Lettura e scrittura avvengono a blocchi
 - Ci sono alcune differenze....

Invio messaggi su Socket

- La funzione `send()` è definita in `sys/socket.h`

```
ssize_t send(int fd, const void *buf, size_t n, int flags);
```

- `fd`: **descrittore della socket**
- `buf`: **puntatore al buffer contenente il messaggio da inviare**
- `n`: **numero massimo di byte da scrivere**
- `flags`: **fissato a 0, rende la `send()` equivalente alla `write()`**

Ritorna il numero di byte realmente scritti, o `-1` in caso di errore

- **Default: semantica **bloccante****
 - Se buffer di invio nel kernel non contiene spazio sufficiente per il messaggio da inviare, rimane bloccata in attesa...

Ricezione messaggi su Socket

- La funzione `recv()` è definita in `sys/socket.h`

```
ssize_t recv(int fd, void *buf, size_t n, int flags);
```

- `fd`: descrittore della socket
- `buf`: puntatore al buffer dove scrivere il messaggio ricevuto
- `n`: numero massimo di byte da leggere
- `flags`: se fissato a 0, la `recv()` è equivalente alla `read()`

Ritorna il numero di byte realmente letti, o `-1` in caso di errore

- Ritorna 0 in caso di connessione chiusa
- Default: semantica **bloccante**
 - Se l'altro endpoint non invia nulla, rimane bloccata in attesa
 - Trasferisce i dati disponibili fino a quel momento nel buffer del kernel, entro il limite di `n` bytes, piuttosto che restare in attesa di ricevere l'intera quantità specificata...

Letture e scritture su Socket

Valori di ritorno ed interrupt

- L'analisi e gestione dei valori di ritorno per letture e scritture su socket è più complessa rispetto a quanto visto per i file
- La `send()` è analoga alla `read()`: un segnale può potenzialmente causare un invio parziale di dati, o interrompere la chiamata prima che il primo byte venga trasmesso (setando `errno` ad `EINTR`)
- Per la `recv()`, oltre agli stessi effetti derivanti dalla ricezione di segnali visti per `send()`, si pone il problema che al momento della chiamata possono essere disponibili meno dati di quelli attesi!
 - Come distinguere questo caso da quello dell'interruzione dovuta alla ricezione di un segnale?
 - Come fare quando la dimensione dei dati da ricevere non è nota a priori?
 - In questa esercitazione ci limiteremo a gestire soltanto il caso in cui le chiamate vengono interrotte prima che un byte sia stato letto o scritto

Esercizio proposto: TimeServer

- Scenario
 - Due processi: un client ed un server
 - Il server è in ascolto in attesa di connessioni TCP
 - Il client si connette al server ed invia un comando
 - Messaggio «TIME»
 - Se il server riceve il messaggio atteso, la risposta conterrà ora e data correnti, altrimenti manderà un messaggio di errore
- Sorgenti: `server.c` e `client.c`
- Esercizio
 - Completare le parti mancanti, relative all'invio/ricezione di messaggi via socket
 - Per l'esecuzione, è necessario lanciare client e server su terminali diversi