Record locking con la system call `fcntl`

Il **record locking** è un metodo per disciplinare la cooperazione tra processi.

Permette ad un processo di poter accedere ad un file in modo esclusivo.

È utile quindi per la gestione di database condivisi.

Tuttavia l'operazione di **lock** non è automatica, ma deve essere eseguita esplicitamente dal processo.

Esempio: prenotazione di voli aerei

La compagnia *ACME Airlines* usa un sistema di prenotazione dei voli basato su unix. Possiede due uffici per la prenotazione, A e B, ciascuno con un terminale connesso al computer della compagnia.

Da ciascuno dei terminali si può accedere al **database** delle prenotazioni, implementato come **file unix**, attraverso il programma `acmebook`.

Questo programma permette di leggere e aggiornare il database. In particolare, è possibile decrementare di uno il numero di posti liberi su un determinato volo.

Una situazione critica

Supponiamo che sul volo *ACM501* per Londra sia rimasto libero esattamente un posto. Il Sig.Jones e il Sig.Smith entrano nell'ufficio A e B, rispettivamente, chiedendo un biglietto per quel volo.

Consideriamo la seguente sequenza di eventi:

1. Dall'ufficio A viene lanciato il programma `acmebook`. Sia *PA* il processo risultante.

2. Dall'ufficio B viene lanciato il programma `acmebook`. Sia *PB* il processo risultante.

3. *PA* accede al database con una `read` e scopre che c'è un posto libero.

4. *PB* fa una `read` sul database e scopre che c'è un posto libero.

5. *PA* azzera il contatore di posti liberi con una `write` e consegna il biglietto al Sig.Jones.

6. *PB*, non sapendo che l'unico posto rimasto è stato già assegnato, azzera il contatore di posti liberi con una `write` e consegna il biglietto al Sig.Smith.

Come conseguenza è stato assegnato un posto in più rispetto a quelli disponibili.

Il **problema** nasce perchè più processi possono accedere contemporaneamente ad uno stesso file. Inoltre una singola operazione logica (la prenotazione), costituita da diverse chiamate alle system call `lseek`, `read`, `write`, può essere effettuata da più processi concorrenti.

Una **soluzione** consiste nel consentire ad un processo di fare il **locking** della parte di file sulla quale sta lavorando durante la prenotazione. La sistem call `fcntl` consente di effettuare il **locking** di (parte di) un file.

Record locking con `fcntl`

La system call `fcntl` offre due tipi di **locking**:

1. **Read locking**: Serve per evitare che i dati vengano modificati da un processo, compreso il processo che ha effettuato il **read locking**. Tuttavia tutti i processi possono accedere ai dati in lettura. In particolare, il **read locking** non consente a nessun processo di effettuare un **write locking**. Più processi possono effettuare un **read locking** contemporaneamente.

2. **Write locking**: Il processo che lo ha effettuato può modificare la parte di file su cui ha effettuato il **write locking**. Ma nessun altro processo può effettuare un **read/write locking** su quella parte di file.

Uso della system call `fcntl` per il locking

```
#include <fcntl.h>
int fcntl(int filedes, int cmd, struct flock *ldata);
```

- `filedes` è un descrittore di file aperto con il parametro `O_RDONLY` oppure `O_RDWR`, nel caso di un **read locking**, `O_WRONLY` oppure `O_RDWR`, nel caso di un **write locking**.

- `cmd` specifica l'azione da effettuare tramite valori definiti in `<fcntl.h>`:

`F_GETLK`	Fornisce la descrizione dei locking in base al contenuto di `ldata`. L'informazione restituita descrive il primo lock che blocca il lock descritto in `ldata`.
`F_SETLK`	Effettua un locking su un file; termina subito se non è possibile. Serve anche per rimuovere un lock.
`F_SETLKW`	Effettua un locking su un file, va in sleep se il lock è bloccato da un lock precedentemente effettuato da un altro processo.

Uso della system call `fcntl` per il locking (continua)

- La struttura `ldata` contiene la descrizione del locking. In particolare, comprende i seguenti campi:

```
short l_type;   /* descrive il tipo di lock */
short l_whence; /* offset */
off_t l_start ; /* offset in byte */
off_t l_len;    /* dimensione del segmento in byte */
pid_t l_pid;    /* definito dal comando F_GETLK */
```

  I campi `l_whence`, `l_start`, `l_len` specificano il segmento su cui effettuare, controllare o togliere il locking. `l_whence` determina da dove calcolare l'offset e può essere `SEEK_SET` (dall'inizio del file), `SEEK_CUR` (dalla posizione corrente), `SEEK_END` (dalla fine del file). `l_start` fornisce la posizione di partenza del segmento relativamente a `l_whence`. `l_len` è la lunghezza del segmento in byte. `l_type` fornisce il tipo di lock:

`F_RDLCK`	read locking
`F_WRLCK`	write locking
`F_UNLCK`	unlocking

  Il campo `l_pid` è rilevante solo solo se il parametro `cmd` vale `F_GETLK`. Se esiste un lock che blocca il lock descritto dagli altri membri della struttura, a `l_pid` viene assegnato l'identificatore del processo che ha effettuato il locking.

Esercizi

1. Scrivere un programma che simuli il sistema di prenotazione di voli *ACME*. Il programma acquisisce da std input il numero di posti da riservare rispettivamente dall'ufficio A e dall'ufficio B. Il `main` crea due processi figli, *PA* e *PB*, a cui passa il numero di posti da riservare rispettivamente dall'ufficio A e dall'ufficio B. I processi *PA* e *PB* chiamano ripetutamente la funzione `acmebook` per riservare i posti uno alla volta. La funzione `acmebook` utilizza il meccanismo del locking per effettuare le prenotazioni. Quando non ci sono più posti liberi, il programma termina, altrimenti aspetta da std input un'altra coppia di interi, rappresentanti i posti da riservare dagli uffici A e B.

2. Siano *P1* e *P2* due processi che lavorano sullo stesso file. Supponiamo che *P1* esegua un lock sulla sezione *SX* del file e *P2* esegua un lock sulla sezione *SY* dello stesso file. Che cosa succede se poi *P1* tenta di fare un lock su *SY* con `F_SETLKW` e *P2* tenta di fare un lock su *SX* con `F_SETLKW`?