

Sviluppo di software di rete con l'utilizzo di sistemi UNIX

Locking di file e di record

Claudio Vicari

Definizioni

- ➔ **IPC: Inter Process Communication.** Con questo termine ci riferiamo ai vari metodi disponibili in Unix per la comunicazione fra processi
- ➔ ci sono quattro principali forme di IPC:
 - 1 **passaggio di messaggi**
 - pipe, FIFO, code di messaggi
 - 2 **sincronizzazione**
 - mutex, lock, semafori
 - 3 **memoria condivisa**
 - 4 **remote procedure call**
 - SUN RPC

Lock di file e di record

- ➡ i file possono essere condivisi fra processi
- ➡ se due processi scrivono contemporaneamente nello stesso file, il risultato è imprevedibile, perché dipende fortemente dallo scheduling determinato dal processore

Visualizza esempio

- ➡ `gcc -DNOLOCK -o /tmp/prova lockmain.c`
- ➡ `echo 1 > seqno ; ./prova & ./prova`
- ➡ come vediamo, le cose cambiano da esecuzione ad esecuzione

Lock di file e di record: funzione fcntl

```
int fcntl(int fd, int cmd, struct flock *lock);

struct flock {
    short l_type;    /* F_RDLCK, F_WRLCK, F_UNLCK */
    short l_whence; /* SEEK_SET, SEEK_CUR, SEEK_END */
    off_t l_start;   /* Starting offset for lock */
    off_t l_len;     /* Number of bytes to lock */
    pid_t l_pid;     /* PID blocking our lock
                      (F_GETLK only) */
};
```

fcntl: operazioni possibili

- ➡ i lock possono essere in lettura o scrittura. Molti processi possono leggere dallo stesso file contemporaneamente. Un lock in scrittura, però, blocca tutti gli altri processi!
- ➡ i comandi che possiamo specificare sono:
 - F_SETLK per richiedere un lock (non blocca, restituisce un errore se non è possibile ottenerlo)
 - F_SETLKW per richiedere un lock in modo bloccante
 - F_GETLCK per sapere se c'è già un lock (le informazioni finiscono nella struttura flock)

fcntl: operazioni possibili

- ➔ mediante la struttura flock, possiamo specificare:
 - il tipo di lock: F_RDLCK, F_WRLCK, F_UNLCK
 - quali byte bloccare: whence (SEEK_SET, SEEK_CUR, SEEK_END), start, len
- ➔ in Unix parliamo di locking di file o di record, ma in realtà non esiste un concetto di record! Possiamo però specificare a piacere quali byte bloccare nel file specificato
- ➔ possiamo bloccare dalla posizione specificata fino alla fine del file, specificando 0 per l_len
- ➔ possiamo bloccare l'intero file in due modi:
 - l_whence = SEEK_SET, l_start=0, l_len=0
 - stando all'inizio del file (lseek), l_whence = SEEK_CUR, l_start=0, l_len=0

Esempio: funzione di lock

```
void
my_lock(int fd)
{
    struct flock lock;

    lock.l_type = F_WRLCK;
    lock.l_whence = SEEK_SET;
    lock.l_start = 0;
    lock.l_len = 0; /* write lock entire file */

    Fcntl(fd, F_SETLKW, &lock);
}
```

Esempio: funzione di unlock

```
void
my_unlock(int fd)
{
    struct flock lock;

    lock.l_type = F_UNLCK;
    lock.l_whence = SEEK_SET;
    lock.l_start = 0;
    lock.l_len = 0;          /* unlock entire file */

    Fcntl(fd, F_SETLK, &lock);
}
```

Esempio

- ➔ `gcc -DLOCK_SIMPLE -o /tmp/prova lockmain.c`
- ➔ se adesso rilanciamo i due processi, l'output del programma è corretto
- ➔ se però ne compiliamo uno con le nuove funzioni di lock, l'altro senza lock, l'output torna non predicibile
- ➔ questo tipo di locking è detto *advisory*. Un processo può comunque scavalcarlo! Dunque, è necessario che tutti i processi cooperino
- ➔ l'altro tipo di locking è detto *mandatory*, e previene l'uso del file da parte di altri processi

Mandatory locking

- ⇒ non tutti i sistemi hanno il mandatory locking
 - può essere necessario montare il filesystem attivando l'opzione `-o mand`
- ⇒ per attivare il mandatory locking, è necessario che il file abbia:
 - il bit set-group-id impostato a 0
 - il bit set-uid impostato a 1
 - su linux è differente!!

Mandatory locking: esempio

Visualizza esempio

- ➔ attivare l'opzione per avere il filesystem con mandatory locking
- ➔ compilare l'esempio:
 - `gcc -DLOCK_SIMPLE -o /tmp/prova mandatory_prova.c`
- ➔ creare il file ed impostare i diritti:
 - `touch mandatory_lock; chmod 2700 mandatory_lock`
- ➔ eseguire: `cd /tmp; ./prova`
- ➔ provare ad accedere il file con `vi`: si blocca!

Locking: problemi

- ⇒ anche con il mandatory locking, un processo che non cooperi può creare problemi
 - potrebbe darsi che un tal processo prenda il controllo, legga dal file, quindi un altro processo blocca il file, legge e quindi scrive, e infine il processo non cooperante scriverebbe il numero di sequenza errato
- ⇒ non bisogna usare la libreria stdio con il locking, a causa della bufferizzazione
- ⇒ attenzione alle priorità fra lock!
 - se un processo blocca il file in lettura, altri processi possono bloccarlo in lettura...
 - i processi che scrivono, potrebbero non ottenere mai il lock!
 - dipende dall'implementazione...

Esempi d'uso

- ➔ il locking di file può essere usato per evitare che ci siano istanze multiple di uno stesso programma (in particolare, si usa per i processi demone)
- ➔ esempio: se lanciamo due volte dhcpcd (demone per ottenere un indirizzo con dhcp), otteniamo un errore
- ➔ molti demoni fanno ciò, scrivendo anche il proprio PID nel file, in modo che un'altra istanza possa anche mandargli segnali

Esempi d'uso

Visualizza esempio

- ➡ nel codice di esempio, il processo apre e blocca il file
- ➡ il locking è usato in modo non bloccante, cioè anche se non è possibile ottenere il lock, la chiamata a `fcntl` ritorna immediatamente
- ➡ quindi, eseguiamo una richiesta di lock che è anche un test

Alternativa: utilizzo di file come lock

- ➔ se un file viene creato chiamando open, con i flag O_CREAT e O_EXCL, la funzione restituisce un errore in caso che il file esista, altrimenti lo crea
- ➔ quindi, questo file può essere utilizzato come se fosse un lock
- ➔ problemi:
 - se il processo termina accidentalmente, il file non viene rimosso
 - non c'è la possibilità di fare una wait, nell'attesa di ottenere il lock
 - è più lento dei lock su file e record
 - tuttavia, esistono ancora dei programmi che usano questa tecnica

Esercizi

- ➡ scrivere un programma che scriva (ogni secondo) in un file il proprio pid ed il tempo passato dal momento in cui è stato eseguito, per 10 secondi. Quindi, rilasci il lock per 10 secondi, e tenti di riottenerlo per proseguire
- ➡ inserirvi un processo figlio che attenda 5 secondi, quindi cerchi di ottenere il controllo del file e continui a scrivervi il proprio pid, ed il tempo passato dall'esecuzione (altri 5 secondi)
- ➡ fare in modo che i ruoli dei due processi nello scrivere il file si alterni ciclicamente...