2024-09-11

Locking (Sincronizzazione)

Definiamo un *rwlock* come un oggetto su cui i processi possono acquisire e poi rilasciare un lock in scrittura (processi scrittori) o in lettura (processi lettori), rispettando le seguenti condizioni:

1. più processi possono avere contemporaneamente il lock in lettura, purché nessun processo abbia il lock in scrittura;
2. un solo processo alla volta può avere il lock in scrittura.

I processi che provano ad acquisire un lock si sospendono fino a quando le condizioni non lo consentono. Se più processi sono in attesa di acquisire un lock, si da la precedenza ai processi lettori; i processi scrittori sono ordinati tra loro in base alla precedenza.

Un processo che possiede un lock in lettura può anche eseguire un *upgrade* del lock per trasformarlo in un lock in scrittura, sempre rispettando le condizioni. Un lock in scrittura che era stato ottenuto tramite l'upgrade di un lock in lettura può poi essere ri-trasformato in un lock in lettura tramite una operazione di *downgrade*. Per comodità, la stessa operazione di *downgrade* può essere usata anche su un lock in lettura o su un lock in scrittura non ottenuto tramite upgrade, e in questi casi corrisponde a rilasciare il lock corrispodente.

Per realizzare i rw definiamo i seguenti tipi (file `sistema.cpp`):

```
enum rw_states {
    RW_NONE,    ///< nessun lock
    RW_WRITER,  ///< lock in scrittura
    RW_READER,  ///< lock in lettura
    RW_UPGRADED ///< lock in lettura trasfromato in scrittura
};
struct des_rw {
    /// id del processo che possiede il write lock (0 se nessuno)
    natl writer;
    /// numero dei processi che possiedono un read lock
    natl nreaders;
    /// processi in attesa di acquisire un read lock
    des_proc* w_readers;
    /// processi in attesa di acquisire il write lock
    des_proc* w_writers;
};
/// Array dei descrittori di rwlock
des_rw array_desrw[MAX_RW];
/// Numero di rwlock allocati
natl rw_allocati = 0;
struct des_proc_rw {
```

```
    des_rw *r;   ///< rwlock riferito (nullptr se il descrittore non è usato)
    rw_states state; ///< tipo di lock posseduto
};
```

La struttura `des_rw` descrive un rwlock. Il campo `writer` contiene l'id del processo che ha il lock in scrittura (0 se nessuno); il campo `nreaders` conta i processi che hanno il lock in lettura; il campo `w_readers` è una lista di processi in attesa di acquisire un lock in lettura; il campo `w_writers` è una lista di processi in attesa di acquisire il lock in scrittura.

La struttura `des_proc_rw` descrive invece un lock posseduto da un processo. Il campo `r` punta al descrittore del rwlock corrispondente, e il campo `state` descrive il tipo di lock (nessuno, scrittura, lettura, scrittura ottenuta tramite upgrade).

```
natl alloca_rw()
{
    natl i;

    if (rw_allocati >= MAX_RW)
        return 0xFFFFFFFF;

    i = rw_allocati;
    rw_allocati++;
    return i;
}

bool rw_valido(natl rw)
{
    return rw < rw_allocati;
}

extern "C" natl c_rw_init()
{
    natl rw = alloca_rw();

    if (rw == 0xFFFFFFFF)
        return 0xFFFFFFFF;

    des_rw *r = &array_desrw[rw];

    r->nreaders = 0;
    r->writer = 0;
    r->w_readers = nullptr;
    r->w_writers = nullptr;
    return rw;
}
```

```
static des_proc_rw *rw_proc_find(des_rw *r = nullptr, des_proc *p = esecuzione)
{
    for (int i = 0; i < MAX_PROC_RW; i++) {
        des_proc_rw *rp = &p->rws[i];
        if (rp->r == r)
            return rp;
    }
    return nullptr;
}
extern "C" void c_rw_writelock(natl rw)
{
    if (!rw_valido(rw)) {
        flog(LOG_WARN, "rw_writelock(%d): rwlock non valido", rw);
        c_abort_p();
        return;
    }

    des_rw *r = &array_desrw[rw];
    des_proc_rw *rp;

    if (rw_proc_find(r)) {
        flog(LOG_WARN, "rw_writelock(%d): rwlock gia' attivo", rw);
        c_abort_p();
        return;
    }

    rp = rw_proc_find();
    if (!rp) {
        esecuzione->contesto[I_RAX] = false;
        return;
    }
    rp->r = r;

    esecuzione->contesto[I_RAX] = true;
    if (r->nreaders > 0 || r->writer) {
        rp->state = RW_NONE;
        inserimento_lista(r->w_writers, esecuzione);
        schedulatore();
    } else {
        rp->state = RW_WRITER;
        r->writer = esecuzione->id;
    }
}
```

Aggiungiamo inoltre il seguente campo ai descrittori di processo:

```
struct des_proc {
```

```
    ...
    /// descrittori dei lock posseduti
    des_proc_rw rws[MAX_PROC_RW];
};
des_proc* crea_processo(void f(natq), natq a, int prio, char liv)
{
    ...
    for (int i = 0; i < MAX_PROC_RW; i++) {
        des_proc_rw *rp = &p->rws[i];
        rp->r = nullptr;
        rp->state = RW_NONE;
    }
}
```

L'array `rws` descrive tutti i lock posseduti dal processo e i loro stato (ogni processo può possedere al massimo `MAX_PROC_RW` lock). Le entrate libere dell'array `rws` hanno `r` impostato a `nullptr`.

Le seguenti primitive, accessibili dal livello utente, operano sui rwlock (nei casi di errore, abortiscono il processo chiamante):

- `natl rw_init()` (già realizzata): inizializza un nuovo rwlock e ne restituisce l'identificatore. Se non è possibile creare un nuovo rwlock restituisce `0xFFFFFFFF`.
- `bool rw_writlock(natl rw)` (già realizzata): acquisisce il lock in scrittura sul rwlock di identificatore `rw`. Se ci sono già processi che hanno un lock (di lettura o scrittura) e non lo hanno ancora rilasciato, sospende il processo in attesa che le condizioni permettano l'acquisizione del lock. È un errore se `rw` non è un id valido o se il processo possiede già un lock sullo stesso rwlock. Restituisce `false` se il processo possiede già `MAX_PROC_RW` lock.
- `bool rw_readlock(natl rw)` (già realizzata): acquisisce un lock in lettura sul rwlock di identificatore `rw`. Se c'è un processo che ha il lock in scrittura e non lo ha ancora rilasciato, sospende il processo in attesa che le condizioni permettano l'acquisizione del lock in lettura. È un errore se `rw` non è un id valido o se il processo possiede già un lock sullo stesso rwlock. Restituisce `false` se il processo possiede già `MAX_PROC_RW` lock.
- `void rw_upgrade(natl rw)`: Rilascia il lock in lettura e contestualmente ne acquisice uno in scrittura sul rwlock di identificatore `rw`. Se altri processi possedevano un lock in lettura, sospende il processo in attesa che le condizioni permettano l'acquisizione del lock in scrittura. Attenzione: se altri processi stavano erano in attesa di poter acquisire il lock in scrittura, il lock andrà al processo con priorità maggiore. È un errore se `rw` non è un id valido o se il processo non possiede un lock in lettura su `rw`.
- `void rw_downgrade(natl rw)`: Rilascia o esegue il downgrade del lock

del processo sul rwlock di identificatore `rw`. Nel caso di downgrade, rilascia il lock in scrittura e contestualmente riacquisice un lock in lettura. Negli altri casi rilascia semplicemente il lock (in lettura o scrittura). È un errore se `rw` non è valido o se il processo non possiede lock su `rw`.

```
extern "C" void c_rw_readlock(natl rw)
{
    if (!rw_valido(rw)) {
        flog(LOG_WARN, "rw_readlock(%d): rwlock non valido", rw);
        c_abort_p();
        return;
    }

    des_rw *r = &array_desrw[rw];
    des_proc_rw *rp;

    if (rw_proc_find(r)) {
        flog(LOG_WARN, "rw_readlock(%d): rwlock gia' attivo", rw);
        c_abort_p();
        return;
    }

    rp = rw_proc_find();
    if (!rp) {
        esecuzione->contesto[I_RAX] = false;
        return;
    }
    rp->r = r;

    esecuzione->contesto[I_RAX] = true;
    if (r->writer) {
        rp->state = RW_NONE;
        inserimento_lista(r->w_readers, esecuzione);
        schedulatore();
    } else {
        rp->state = RW_READER;
        r->nreaders++;
    }
}
```

Modificare il file `sistema.cpp` in modo da realizzare le primitive mancanti.

```
void rw_wakeup_writer(des_rw *r)
{
    des_proc *p = rimozione_lista(r->w_writers);
    if (!p)
        return;
```

```
    des_proc_rw *rp = rw_proc_find(r, p);
    rp->state = (rp->state == RW_READER ? RW_UPGRADED : RW_WRITER);
    r->writer = p->id;
    inserimento_lista(pronti, p);
}

bool rw_wakeup_readers(des_rw *r)
{
    natl n = r->nreaders;

    while (des_proc *p = rimozione_lista(r->w_readers)) {
        des_proc_rw *rp = rw_proc_find(r, p);
        rp->state = RW_READER;
        r->nreaders++;
        inserimento_lista(pronti, p);
    }
    return r->nreaders > n;
}

/// Parte C++ della primitiva rw_upgrade()
extern "C" void c_rw_upgrade(natl rw)
{
    if (!rw_valido(rw)) {
        flog(LOG_WARN, "rw_writelock(%d): rwlock non valido", rw);
        c_abort_p();
        return;
    }

    des_rw *r = &array_desrw[rw];
    des_proc_rw *rp;

    rp = rw_proc_find(r);
    if (!rp || rp->state != RW_READER) {
        flog(LOG_WARN, "rw_upgrade(%d): stato non valido", rw);
        c_abort_p();
        return;
    }

    r->nreaders--;
    if (r->nreaders > 0) {
        inserimento_lista(r->w_writers, esecuzione);
        schedulatore();
    } else if (r->w_writers && r->w_writers->precedenza > esecuzione->precedenza) {
        inserimento_lista(r->w_writers, esecuzione);
        rw_wakeup_writer(r);
        schedulatore();
```

```
    } else {
        rp->state = RW_UPGRADED;
        r->writer = esecuzione->id;
    }
}

extern "C" void c_rw_downgrade(natl rw)
{
    if (!rw_valido(rw)) {
        flog(LOG_WARN, "rw_downgrade(%d): rwlock non valido", rw);
        c_abort_p();
        return;
    }

    des_rw *r = &array_desrw[rw];
    des_proc_rw *rp = rw_proc_find(r);
    if (!rp) {
        flog(LOG_WARN, "rw_downgrade(%d): rwlock non attivo", rw);
        c_abort_p();
        return;
    }
    inspronti();
    switch (rp->state) {
    case RW_READER:
        rp->r = nullptr;
        rp->state = RW_NONE;
        r->nreaders--;
        if (!r->nreaders)
            rw_wakeup_writer(r);
        break;
    case RW_UPGRADED:
        rp->state = RW_READER;
        r->nreaders = 1;
        r->writer = 0;
        rw_wakeup_readers(r);
        break;
    case RW_WRITER:
        rp->r = nullptr;
        rp->state = RW_NONE;
        r->writer = 0;
        if (!rw_wakeup_readers(r))
            rw_wakeup_writer(r);
        break;
    default:
        fpanic("stato %d non attesso in rwlock %d", rp->state, rw);
        break;
```

```
    }
    schedulatore();
}
```