Prova pratica di Calcolatori Elettronici

C.d.L. in Ingegneria Informatica, Ordinamento DM 270

11 settembre 2024

1. Siano date le seguenti dichiarazioni, contenute nel file `cc.h`:

```
struct st {
        char vv1[4];
        long vv2[4];
};
class cl {
        st s;
public:
        cl(char v[]);
        void elab1(st& ss, int d);
        void stampa()
        {
                for (int i = 0; i < 4; i++)
                        cout << (int)s.vv1[i] << ' ';
                cout << '\t';
                for (int i = 0; i < 4; i++)
                        cout << s.vv2[i] << ' ';
                cout << endl;
                cout << endl;
        }
};
```

   Realizzare in Assembler GCC le funzioni membro seguenti.

```
void cl::elab1(st& ss, int d)
{
        for (int i = 0; i < 4; i++) {
                if (d < ss.vv2[i])
                        s.vv1[i] += ss.vv1[i];
                s.vv2[i] = d + i;
        }
}
```

2. 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`):

```
struct des_rw {
    natl writer;
    natl nreaders;
    des_proc* w_readers;
    des_proc* w_writers;
};
enum rw_states { RW_NONE, RW_WRITER, RW_READER, RW_UPGRADED };
struct des_proc_rw {
    des_rw *r;
    rw_states state;
};
```

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).

Aggiungiamo inoltre il seguente campo ai descrittori di processo:

```
des_proc_rw rws[MAX_PROC_RW];
```

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`.

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