Prova pratica di Calcolatori Elettronici

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

28 gennaio 2026

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

```
struct st1 { char vc[4]; }; struct st2 { int vd[4]; };
class cl
{       st2 s; long v[4];
public:
        cl(char *c, st2 s2);
        void elab1(st1& s1, st2 s2);
        void stampa()
        {       int i;
                for (i=0;i<4;i++) cout << s.vd[i] << ' '; cout << endl;
                for (i=0;i<4;i++) cout << v[i] << ' '; cout << endl << endl;
        }
};
```

   Realizzare in Assembler GCC le funzioni membro seguenti.

```
void cl::elab1(st1& s1, st2 s2)
{
        cl cla(s1.vc, s2);
        for (int i = 0; i < 4; i++) {
                if (s.vd[i] < s1.vc[i]) {
                        s.vd[i] = cla.s.vd[i];
                        v[i] += cla.v[i];
                }
        }
}
```

2. Definiamo un *rw* come un oggetto su cui i processi possono leggere o scrivere rispettando le seguenti condizioni:

   1. più processi possono leggere contemporaneamente, purchè nessun processo stia scrivendo;
   2. un solo processo alla volta può scrivere.

   Supporremo anche che ci sia un massimo (`MAX_RW_READERS`) al numero di processi che possono leggere contemporaneamente.

   Per leggere o scrivere su un rw, i processi devono prima acquisire il diritto di lettura o scrittura, quindi devono rilasciare tale diritto quando hanno terminato.

   Per evitare che i processi attendano indefinitamente, introduciamo le seguenti regole di *fairness*

- nuovi processi non possono acquisire il diritto di lettura se ci sono processi in attesa di acquisire il diritto di scrittura;
- al momento del rilascio di un diritto, i processi scrittori danno la precedenza ai processi lettori, e viceversa.

Per realizzare i rw definiamo la seguente struttura (file `sistema.cpp`):

```
struct des_rw {
    natl readers[MAX_RW_READERS];
    natl writer;
    natl nreaders;
    des_proc* w_readers;
    des_proc* w_writers;
};
```

Il campo `readers` memorizza gli *id* degli eventuali processi che hanno acquisito il diritto di lettura e non lo hanno ancora rilasciato (i campi non utilizzati contengono zero). Il campo `nreaders` memorizza il numero di tali processi. Il campo `writer` memorizza l'*id* dell'eventuale processo che ha acquisito il diritto di scrituttura e non lo ha ancora rilasciato (zero se non c'è). La lista `w_readers` contiene i processi in attesa di acquisire il diritto di lettura. La lista `w_writers` contiene i processi in attesa di acquisire il diritto di scrittura.

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

- `natl rw_init()` (già realizzata): inizializza un nuovo rw e ne restituisce l'identificatore. Se non è possibile creare un nuovo rw restituisce `0xFFFFFFFF`.
- `void rw_acq_write(natl rw)` (realizzata in parte): tenta di acquisire il diritto di scrittura sul rw di identificatore `rw`. Se necessario, sospende il processo in attesa che le condizioni permettano l'acquisizione del diritto di scrittura.
- `void rw_acq_read(natl rw)` (realizzata in parte): tenta di acquisire il diritto di lettura sul rw di identificatore `rw`. Se necessario, sospende il processo in attesa che le condizioni permettano l'acquisizione del diritto di lettura.
- `void rw_rel_write(natl rw)` (realizzata in parte): Rilascia il diritto di scrittura sul rw di identificatore `rw`.
- `void rw_rel_read(natl rw)` (realizzata in parte): Rilascia il diritto di lettura sul rw di identificatore `rw`.

È sempre un errore tentare di acquisire un diritto se se ne possiede già uno, o tentare di rilasciare un diritto che non si possiede.

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