Prova pratica di Calcolatori Elettronici

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

20 settembre 2016

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

```
struct st1 { char vc[4]; }; struct st2 { int vd[4]; };
class cl
{       st1 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.vc[i] << ' '; cout << endl;
                for (i=0;i<4;i++) cout << v[i] << ' '; cout << endl << endl;
        }
};
```

   Realizzare in Assembler GCC le funzioni membro seguenti.

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

2. Introduciamo un meccanismo di *broadcast* tramite il quale un processo può inviare un messagio ad un insieme di processi. Per ricevere un broadcast i processi si devono preventivamente registrare. Un processo può inviare un brodcast tramite la primitiva `broadcast(msg)`, la quale attende anche che tutti i processi che risultano registrati ricevano il messaggio. Un processo registrato può ricevere un broadcast invocando la primitiva `listen()`, che attende che sia disponibile il prossimo messaggio.

   Per realizzare i broadcast introduciamo la seguente struttura dati:

```
struct broadcast {
    int registered;
    int nlisten;
    natl msg;
    proc_elem *listeners;
    proc_elem *broadcaster;
};
```

Dove: `registered` è il numero di processi registrati; `nlisten` conta i processi che hanno invocato `listen()` dall'ultimo completamento di una operazione di broadcast; `msg` contiene l'ultimo messaggio di broadcast; `listeners` è la coda dei processi in attesa del prossimo messaggio; `broadcaster` è la coda in cui attende il processo che vuole inviare il broadcast, in attesa che tutti i processi registrati invochino `listen()`.

Aggiungiamo inoltre le seguenti primitive (abortiscono il processo in caso di errore):

- `void reg()` (tipo 0x3a, già realizzata): registra il processo per la ricezione dei broadcast; non fa niente se il processo è già registrato;
- `natl listen()` (tipo 0x3b, da realizzare): riceve il prossimo messagio di broadcast; è un errore se il processo non è registrato;
- `void broadcast(natl msg)` (tipo 0x3c, da realizzare): invia in broadcast il messaggio `msg`; è un errore se il processo è registrato.

Le primitive abortiscono il processo chiamante in caso di errore e tengono conto della priorità tra i processi.

Per semplicità si assuma che, durante tutta la sua esecuzione, ogni processo provi al più una sola volta a inviare un broadcast o ad ascoltare un messaggio. Inoltre, al più un processo alla volta tenta di eseguire un broadcast.

Modificare i file `sistema.cpp` e `sistema.S` in modo da realizzare le primitive mancanti.