SISTEMI OPERATIVI IIN/IEL/IDT
INFORMATICA INDUSTRIALE E SISTEMI OPERATIVI IDI

SISTEMI DI ELABORAZIONE P.O.
prova scritta preliminare del 30.01.2004

Nome: _____ Cognome: _____

Un servizio accessibile da remoto tramite protocollo TCP (ad esempio un server web) è realizzato mediante una coda a dimensione limitata e due classi di threads (*ascoltatori* e *lavoratori*) che ripetono indefinitamente le seguenti operazioni:

- i threads *ascoltatori* accedono in mutua esclusione ad una porta di ascolto predefinita (ad esempio, per il server web, la porta 80) attraverso una chiamata alla funzione `accept()`; la funzione restituisce una struttura dati di tipo `Socket`, contenente i dati che identificano il cliente che ha fatto la richiesta. I thread ascoltatori depositano tale struttura nella coda;
- ciascun *lavoratore* preleva dalla coda una struttura dati di tipo `Socket`, legge la richiesta proveniente dal cliente e quindi invia la risposta.

Si consideri il caso in cui si abbiano una coda di dimensione K, M threads *ascoltatori* ed N threads *lavoratori* e si sviluppi una soluzione che faccia uso dei monitor e delle variabili di condizione per risolvere i problemi di sincronizzazione.
Si discuta infine quali modifiche debbano essere applicate alla soluzione proposta nel caso sussista il vincolo per cui ad ogni monitor può essere associata una sola variabile di condizione.

Soluzione.

Una istanza *p* del monitor *porta* è condivisa tra i processi *Ascoltatori*. I processi *Ascoltatori* e *Lavoratori* condividono un'istanza *c* del monitor *coda.*

Ascoltatore

```
Socket s;
while (true) {
    s = p.prendiPorta();

    c.deposita(s);
}
```

Lavoratore

```
Socket s;
while (true) {
    s = c.preleva();

    processaRichiesta(s);          // elabora la richiesta
}
```

Monitor per l'accesso alla porta

```
Monitor porta
{       Socket s;

        Socket prendiPorta()
        {       s = accept();
                return s;
        }
}
```

Monitor per l'accesso alla coda

```
Monitor coda
{       Socket buffer[K];
        Condition pieno, vuoto;
        int in = 0, out = 0, count = 0;

        void deposita(Socket s)
        {       if (count==K)          // il processo si sospende se la coda è piena
                        pieno.wait();
                count ++;
                buffer[in] = s;
                in = (in+1) % K;
                vuoto.signal();        // non ha effetto se la coda associata alla variabile di
                                       // condizione non contiene alcun processo
        }
```

```
        Socket prelevaCoda()
        {       if (count==0)           // il processo si sospende se la coda è vuota
                        vuoto.wait();
                s = buffer[out];
                out = (out+1) % K;
                count --;

                pieno.signal();         // non ha effetto se la coda associata alla variabile di
                                        // condizione non contiene alcun processo
                return s;
        }
}
```

Nel caso in cui ad ogni monitor possa essere associata una sola variabile di condizione, è possibile ipotizzare un meccanismo di funzionamento simile a Java. I processi produttori e consumatori si sospendono su una sola variabile (o in modo equivalente semplicemente richiamando implicitamente `wait()` e `signal()`) e sono risvegliati tutti da una primitiva di sincronizzazione `signalAll()`. I processi dovranno quindi testare nuovamente la condizione di buffer pieno/vuoto prima di poter procedere.

Monitor per l'accesso alla coda

```
Monitor coda
{       Socket buffer[k];
        int in = 0, out = 0, count = 0;

        void deposita(Socket s)
        {       while (count==K)            // il processo si sospende se la coda è piena
                        wait();
                count ++;
                buffer[in] = s;
                in = (in+1) % K;
                signalAll();    // risveglia tutti i processi sospesi con wait()
        }

        Socket prelevaCoda()
        {       while (count==0)            // il processo si sospende se la coda è vuota
                        wait();
                s = buffer[out];
                out = (out+1) % K;
                count --;
                signalAll();    // risveglia tutti i processi sospesi con wait()

                return s;
        }
}
```