Prova pratica di Calcolatori Elettronici (nucleo v6.*)

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

18 febbraio 2015

1. Introduciamo nel nucleo una versione semplificata del meccanismo dei socket per la comunicazione interprocesso, relativamente alle sole fasi di connessione e chiusura.

   Un socket è un oggetto identificato tramite un numero naturale. È possibile creare una connessione tra due socket tramite le operazioni di `accept(natl id)` e `connect(natl src, natl dest)`.

   Un processo che esegue `accept()` su un socket si pone in attesa di richieste di connessione verso verso quel socket. Su uno stesso socket ci può essere più di un processo in attesa di connessioni (più processi possono invocare `accept()` su uno stesso socket).

   La primitiva `connect()` permette ad un processo di richiedere una connessione da un certo socket sorgente verso un socket destinatario. Se sul socket di destinazione vi dei processi in attesa, quello a più alta priorità viene risvegliato e la connessione viene completata. I socket connessi saranno: il socket sorgente della `connect()`; un nuovo socket, il cui identificatore sarà restituito dalla `accept()`. Se non vi sono processi in attesa, la `connect()` fallisce.

   In qualunque momento può essere invocata una `close()` su un socket, che rende il socket nuovamente disponibile per qualunque uso. Se vi sono processi in attesa di connessioni, questi devono essere tutti risvegliati facendo fallire la primitiva che avevano invocato. Se il socket è connesso con un altro socket, il processo che invoca la `close()` deve attendere che anche l'altro socket venga chiuso. Se il socket era già inutilizzato, la primitiva fallisce.

   Per realizzare questo meccanismo definiamo le seguenti strutture dati:

```
enum sock_state {
    SOCK_AVAIL,
    SOCK_ACCEPTING,
    SOCK_CONNECTED,
    SOCK_CLOSING
};
struct des_sock {
    sock_state state;
    des_proc *waiting;
    des_sock *peer;
};
```

   I possibili stati di un socket sono i seguenti:

   - `SOCK_AVAIL`: il socket non è al momento utilizzato;
   - `SOCK_ACCEPTING`: c'è almeno un processo che sta accettando connessioni su questo socket;
   - `SOCK_CONNECTED`: il socket è connesso;
   - `SOCK_CLOSING`: il socket è in fase di chiusura.

La lista `waiting` contiene i processi che sono in attesa di qualche evento sul socket (connessioni o completamento della chisura); il campo `peer` punta all'altro socket nel caso di socket connessi.

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

- `natl socket()` (già realizzata): crea un socket e ne restituisce l'identificatore (`0xFFFFFFFF` se non è stato possibile crearlo);
- `natl accept(natl id)` (già realizzata): pone il processo in attesa di connessioni sul socket `id`; restituisce 0xFFFFFFFF in caso di fallimento; è un errore se il socket non esiste.
- `bool connect(natl src, natl dest)` (già realizzata): tenta di connettere il socket `src` con il socket `dest`; restituisce `false` in caso di fallimento; è un errore se uno dei due socket non esiste.
- `bool close(natl id)` (da realizzare): chiude il socket; restituisce `false` in caso di fallimento.

Tenere conto di eventuali preemption.