Prova pratica di Calcolatori Elettronici (nucleo v6.*)

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

13 settembre 2012

1. Vogliamo aggiungere al nucleo un meccanismo di scambio di messaggi tramite canali. In questo meccanismo un processo può inviare un messaggio `msg` su un canale da cui un altro processo può poi prelevarlo. Assumiamo che i messaggio siano di tipo `natl`. Ogni canale ha un identificatore, di tipo `natl`. Il canale accoda i messaggi inviati e non ancora ricevuti, e ogni canale ha una coda di lunghezza massima finita. Un processo che vuole inviare un nuovo messaggio su un canale e trova la coda piena si blocca fino a quando almeno un messaggio non è stato ricevuto. Un processo che vuole ricevere un messaggio da un canale, ma trova la coda vuota, si blocca fino a quando almeno un messaggio non è stato inviato.

   Inoltre, un processo può porsi in attesa da due canali contemporaneamente. In questo caso il processo si blocca solo se non ci sono messaggi su entrambi i canali e si risveglia non appena c'è un messaggio su almeno uno dei canali. Se c'è un messaggio su entrambi i canali, il processo legge dal canale specificato per primo.

   Per realizzare il precedente meccanismo introduciamo il seguente descrittore di canale:

```
struct des_channel {
        des_proc *wait_w;
        des_proc *wait_r;
        natl *msg_buf;
        natl size;
        natl n_free;
        natl first_free;
        natl first_unread;
};
```

   Il campo `wait_w` è una coda su cui si sospendono i processi in attesa di inviare un messaggio. Il campo `wait_r` è una coda su cui si sospendono i processi in attesa di ricevere un messaggio. Il campo `msg_buf` punta ad un buffer che può contiene al massimo `size` elementi di tipo `natl`. I campi `first_free`, `first_unread` e `n_free` servono a realizzare, nel buffer puntato da `msg_buf`, una coda circolare di messaggi in attesa di essere ricevuti.

   Inoltre aggiungiamo un campo `natl msg` ai descrittori di processo. Un processo che si deve bloccare a causa di una coda piena può salvare qui il messaggio che voleva inviare.

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

   - `natl channel_init(natl size)` (già realizzata): alloca un nuovo canale che pò contenere al massimo `size` messaggi e ne restituisce l'identificatore. Se l'allocazione di un nuovo canale non è possibile, non alloca alcuna risorsa e restituisce `0xFFFFFFFF`.
   - `void channel_send(natl chan_id, natl msg)` (da realizzare): invia il messaggio `msg` sul canale `chan_id`. **Attenzione**: il ricevitore potrebbere essere in attesa anche su un altro canale e quindi può essere necessario eliminarlo da ogni coda. È un errore se `chan_id` non corrisponde ad un canale precedentemente allocato.

- `natl channel_receive(natl chan_id)` (già realizzata): riceve un messaggio dal canale `chan_id`. È un errore se `chan_id` non corrisponde ad un canale precedentemente allocato.
- `natl channel_receive2(natl chan_id1, natl chan_id2)` (da realizzare): riceve un messaggio dai canali `chan_id1` e `chan_id2`. È un errore se `chan_id1` o `chan_id2` non corrispondono ad un canale precedentemente allocato.

Si assuma che su ogni canale vi possa essere un solo processo alla volta in attesa di ricevere.

Modificare i file `sistema.cpp` e `sistema.s` in modo da realizzare le primitive e il codice mancante.

N.B. Gestire correttamente eventuali *preemption*.