Prova pratica di Calcolatori Elettronici (nucleo v6.*)

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

20 luglio 2011

1. Vogliamo aggiungere al nucleo il meccanismo dei *segnali*. Un qualunque processo può inviare un segnale ad un altro processo (di livello utente) di cui conosce l'identificatore. Ogni processo di livello utente può associare un *gestore* al segnale, dove per gestore si intende una qualunque funzione di un tipo opportuno. Il tipo dei gestori è definito come segue:

   ```
   typedef void (*gestore)();
   ```

   (Puntatore a una funzione senza argomenti e che non restituisce niente). Il gestore verrà eseguito dal processo ogni volta che il segnale viene ricevuto. Se il processo non aveva associato alcun gestore, il segnale viene ignorato.

   Per realizzare tale meccanismo procediamo come segue. Supponiamo che il processo P invii un segnale al processo utente Q, che aveva associato il gestore g. In fondo alla pila sistema di Q troveremo le 5 parole quadruple che permettono a Q di tornare al punto del codice di livello utente che stava eseguendo prima di interrompersi. Per fare in modo che Q esegua il gestore g, modifichiamo queste parole quadruple (salvandone il vecchio valore) in modo che Q salti alla prima istruzione di g la prossima volta che tornerà in esecuzione. Il gestore g, prima di terminare, dovrà chiamare una particolare primitiva che ripristini il valore precedentemente salvato per le 5 parole quadruple, in modo che al ritorno dalla primitiva stessa Q ritorni al punto in cui si trovava prima di ricevere il segnale. (Per permettere un corretto ritorno, salviamo e ripristiniamo l'intero stato). Per svolgere le operazioni appena descritte aggiungiamo le seguenti funzioni di utilità:

   - `void salva_ritorno(natl id)`: salva lo stato corrente del processo `id`.
   - `void forza_ritorno(natl id)`: cambia lo stato del processo `id` in modo da fargli eseguire il gestore.
   - `void ripristina_ritorno(natl id)`: ripristina lo stato salvato da `salva_ritorno(id)`.

   Si noti che Q potrebbe ricevere più di un segnale prima di avere l'opportunità di eseguire il gestore, quindi dobbiamo tener conto del numero di segnali pendenti: all'invio di un segnale eseguiamo le operazioni di salvataggio e cambiamento dello stato solo se non c'erano già altri segnali pendenti; al termine di un gestore bisogna controllare se ci sono altri segnali pendenti (nel qual caso il gestore va rieseguito) oppure no (nel qual caso bisogna ripristinare lo stato salvato).

   Aggiungiamo i seguenti campi al descrittore di ogni processo:

   ```
   gestore gest;
   natl    pendenti;
   natl    salva_contesto[N_REG];
   ```

   Il campo `gest` è il gestore associato al segnale (0 se nessun gestore è stato associato). Il campo `pendenti` conta il numero di segnali ricevuti e non ancora gestiti. Il campo `salva_contesto` contiene lo stato salvato `salva_ritorno()`.

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

- `void gestisci(gestore gest)` (tipo `0x3a`, già realizzata): associa il gestore `gest` al segnale. Azzera eventuali segnali pendenti. (`gest` può essere 0).
- `bool segnala(natl id)` (tipo `0x3b`): Invia un nuovo segnale al processo di identificatore `id`. Se il processo `id` non esiste restituisce `false`, altrimenti restituisce `true`. Se il processo `id` non ha associato un gestore, non fa altro. È un errore se processo di identificatore `id` non è un processo di livello utente.
- `void termina_gestore()` (tipo `0x3c`): Primitiva che deve essere chiamata dai gestori prima di terminare. È un errore se la primitiva viene chiamata senza che ci sia almeno un segnale pendente.

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