Prova pratica di Calcolatori Elettronici (nucleo v6.*)

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

12 gennaio 2012

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. I segnali sono caratterizzati da un *tipo*, dove il tipo è un numero che va da 0 a $MAX\_SEGNALI$. Ogni processo di livello utente può associare un *gestore* diverso per ciascun tipo di segnale diverso da 0, dove per gestore si intende una qualunque funzione `void` e senza argomenti. Definiamo per comodità il seguente tipo per i puntatori a gestore:

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

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

   I segnali possono essere temporaneamente e selettivamente mascherati. Se un processo riceve un segnale mascherato, questo resta pendente e verrà gestito non appena sarà stato smascherato.

   Il segnale numero 0 è speciale: non è possibile associarvi un gestore e non può essere mascherato. Se un processo richiede tali operazioni il nucleo le ignora.

   Per realizzare tale meccanismo disponiamo delle seguenti funzioni di utilità:

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

   Si noti che un processo potrebbe ricevere più di un segnale prima di avere l'opportunità di eseguire un gestore, quindi dobbiamo tener conto dei segnali pendenti. Per semplificare l'implementazione stabiliamo che ci possa essere al più un segnale pendente per ogni tipo di segnale (se viene ricevuto un segnale di un tipo che era già pendente il nuovo segnale viene ignorato). Stabiliamo inoltre una priorità tra i segnali, data dall'ordine numerico del loro tipo: 0 è la priorità massima e $MAX\_SEGNALI - 1$ la minima. All'invio di un segnale eseguiamo le operazioni di salvataggio solo se non c'erano già altri segnali pendenti, e forziamo il nostro gestore solo se il nostro segnale è quello a priorità massima; al termine di un gestore bisogna controllare se ci sono altri segnali pendenti (nel qual caso andrà eseguito il gestore a priorità massima tra questi) oppure no (nel qual caso bisogna ripristinare lo stato salvato).

   Aggiungiamo i seguenti campi al descrittore di ogni processo:

   ```
   gestore gest[MAX_SEGNALI];
   bool    pendenti[MAX_SEGNALI];
   bool    mascherati[MAX_SEGNALI];
   //    salva_contesto
   ```

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 `mascherati` tiene conto di quali segnali sono attualmente mascherati. Il campo `salva_contesto` contiene lo stato salvato da `salva_ritorno()`.

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

- `void gestisci(natl signo, gestore gest)` (da realizzare): associa il gestore `gest` al segnale di tipo `signo`, se permesso. Azzera eventuali segnali pendenti di quel tipo. (`gest` può essere 0).
- `bool segnala(natl signo, natl id)` (già realizzata): Invia un nuovo segnale di tipo `signo` al processo di identificatore `id`. Se tale processo non esiste, non ha associato un gestore o `signo` è 0, non fa altro. Se il processo `id` non esiste restituisce `false`, altrimenti restituisce `true`. È un errore se il processo di identificatore `id` non è un processo di livello utente. È un errore se `signo` non è minore di *MAX_SEGNALI*.
- `void termina_gestore(natl signo)` (già realizzata): Primitiva che deve essere chiamata dai gestori di segnali di tipo `signo` prima di terminare. È un errore se la primitiva viene chiamata senza che ci siano segnali di tipo `signo` pendenti.
- `void maschera(natl signo)` (da realizzare): Maschera il segnale `signo` per il processo che la invoca, se permesso.
- `void smaschera(natl signo)` (da realizzare): Smaschera il segnale `signo` per il processo che la invoca, se permesso. (Attenzione: se il segnale appena smascherato era pendente deve essere gestito).

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