Prova pratica di Calcolatori Elettronici (nucleo v6.*)

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

23 luglio 2014

1. Una *barriera* è un meccanismo di sincronizzazione tra processi che funziona nel modo seguente: la barriera è normalmente chiusa; se un processo arriva alla barriera e la trova chiusa, si blocca; la barriera si apre solo quando sono arrivati tutti i processi, che a quel punto si sbloccano; una volta aperta e sbloccati tutti i processi, la barriera si richiude e il meccanismo si ripete.

   Vogliamo realizzare il meccanismo della barriera nel nucleo, prevedendo la possibilità che esistano più barriere distinte, ciascuna con un proprio identificatore. Le barriere sono create dinamicamente, specificando il numero di processi che vi si devono sincronizzare. L'operazione di creazione restituisce l'identificatore della nuova barriera. Un processo si sincronizza su una data barriera specificandone l'identificatore.

   Prevediamo anche la possibilità di distruggere una barriera esistente. Tutti i processi eventualmente bloccati sulla barriera distrutta devono essere risvegliati.

   Per rappresentare una barriera introduciamo la seguente struttura dati:

```
struct barrier {
    natl nproc;
    natl narrived;
    des_proc *waiting;
    barrier *next;
    barrier *prec;
    natl id;
};
```

   Dove: `nproc` è il numero di processi che devono sincronizzarsi sulla barriera; `narrived` conta i processi arrivati alla barriera dall'ultima apertura (o dalla creazione, quando la barriera non è ancora mai stata aperta); `waiting` è la coda dei processi che attendono l'apertura della barriera; `next` e `prec` sono usati per realizzare la lista di tutte le barriere esistenti; `id` è l'identificatore della barriera.

   Aggiungiamo inoltre le seguenti primitive:

   - `natl barrier_create(natl nproc)` (già realizzata): crea una nuova barriera che sincronizza `nproc` processi e ne restituisce l'identificatore (`0xFFFFFFFF` se non è stato possibile completare l'operazione).
   - `bool barrier(natl id)` (da realizzare): fa giungere il processo corrente alla barriera di identificatore `id`. È un errore se tale barriera non esiste. Restituisce `true` quando termina normalmente, e `false` quando termina perchè la barriera è stata distrutta mentre il processo era in attesa dell'apertura.
   - `void barrier_destroy(natl id)` (da realizzare): distrugge la barriera di identificatore `id`. È un errore se tale barriera non esiste.

Le primitive abortiscono il processo chiamante in caso di errore e tengono conto della priorità tra i processi.

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