

# Prova pratica di Calcolatori Elettronici (nucleo v6.\*)

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

2 luglio 2013

1. Un mutex con *priority inheritance* (*pim*) è un semaforo di mutua esclusione che tiene conto delle priorità dei processi per evitare il fenomeno dell'*inversione di priorità*: se un processo ad alta priorità  $P_1$  è bloccato in attesa di acquisire la mutua esclusione su una risorsa posseduta da un processo a bassa priorità  $P_2$ , non vogliamo che processi a priorità intermedia tra  $P_1$  e  $P_2$  interrompano  $P_2$ , perché questo allungherebbe ingiustamente il tempo di attesa di  $P_1$ . I *pim* risolvono questo problema facendo in modo che il processo che possiede la mutua esclusione innalzi la propria priorità, ponendola uguale alla maggiore tra le priorità dei processi in attesa sulla stessa risorsa e la sua. L'innalzamento di priorità è temporaneo: quando il processo libera la risorsa, ritorna alla sua priorità originaria.

Se un processo può acquisire più di una risorsa per volta, dobbiamo tenere conto di alcune complicazioni:

- Un processo  $P_1$  può essere in possesso di una risorsa  $R_1$  e contemporaneamente essere in attesa di acquisire una risorsa  $R_2$ . Se innalziamo la priorità di  $P_1$  dobbiamo anche innalzare la priorità del processo  $P_2$  che possiede  $R_2$ . A sua volta  $P_2$  potrebbe essere in attesa di una risorsa  $R_3$ , quindi dobbiamo anche innalzare la priorità del processo  $P_3$  che possiede  $R_3$ , e così via.
- Se un processo che possiede più risorse ne rilascia una, non deve ritornare alla sua priorità originaria, ma alla massima priorità richiesta dalle risorse che ancora possiede.

Per realizzare i *pim* definiamo la seguente struttura (file `sistema.cpp`):

```
struct des_pim {
    natl curr_prio;
    des_proc *owner;
    des_proc *waiting;
    des_pim *prec;
};
```

Il campo `curr_prio` punta alla priorità massima dei processi in attesa di acquisire la risorsa (0 se non ve ne sono). Il campo `owner` punta al processo che possiede la risorsa (0 se la risorsa è libera). Il campo `waiting` è una lista di processi in attesa di acquisire la risorsa. Il campo `prec` serve a costruire una lista delle risorse possedute dall'attuale owner del *pim* (0 se la risorsa è libera). Le risorse sono in lista in ordine di acquisizione, con la più recente in testa.

Aggiungiamo inoltre i seguenti campi al descrittore di processo:

```
natl orig_prio;
des_pim *owner;
des_pim *waiting;
```

Il campo `orig_prio` contiene la priorità originaria del processo. Il campo `owner` punta all'ultimo *pim* acquisito dal processo (0 se il processo non possiede nessun *pim*). Il campo `waiting` punta al *pim* su cui il processo è in attesa (0 se non è in attesa su alcun *pim*).

Le seguenti primitive, accessibili dal livello utente, operano sui *pim*. In caso di errore abortiscono il processo.

- `natl pim_init()` (inizializza un nuovo pim e ne restituisce l'identificatore. Se non è possibile creare un nuovo pim restituisce `0xFFFFFFFF`).
- `void pim_wait(natl pim)` (da realizzare): tenta di acquisire la mutua esclusione sul pim di identificatore `pim`. È un errore tentare di acquisire un pim che si possiede già.
- `void pim_signal(natl pim)` (già realizzata): rilascia la mutua esclusione sul pim di identificatore `pim`. È un errore tentare di rilasciare un pim diverso dall'ultimo acquisito.

Attenzione: poichè i pim cambiano dinamicamente la priorità di processi mentre questi possono trovarsi già in qualche coda, succede che le code dei processi non sono più ordinate per priorità. È dunque necessario modificare la funzione `rimozione_lista()` in modo che non si limiti ad estrarre dalla testa, ma cerchi il processo a maggiore priorità tra tutti quelli in lista.

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