Prova pratica di Calcolatori Elettronici

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

17 gennaio 2018

1. Siano date le seguenti dichiarazioni, contenute nel file `cc.h`:

```
struct st1 { char vc[4]; }; struct st2 { int vd[4]; };
class cl
{       st1 s; long v[4];
public:
        cl(char *c, st2 s2);
        void elab1(st1 s1, st2 s2);
        void stampa()
        {       int i;
                for (i=0;i<4;i++) cout << s.vc[i] << ' '; cout << endl;
                for (i=0;i<4;i++) cout << v[i] << ' '; cout << endl << endl;
        }
};
```

   Realizzare in Assembler GCC le funzioni membro seguenti.

```
cl::cl(char *c, st2 s2)
{
        for (int i = 0; i < 4; i++) {
                s.vc[i] = c[i];
                v[i] = s2.vd[i] + s.vc[i];
        }
}
void cl::elab1(st1 s1, st2 s2)
{
        cl cla(s1.vc, s2);
        for (int i = 0; i < 4; i++) {
                if (s.vc[i] < s1.vc[i])
                        s.vc[i] = cla.s.vc[i];
                if (v[i] <= cla.v[i])
                        v[i] -= cla.v[i];
        }
}
```

2. Vogliamo aggiungere al nucleo un meccanismo tramite il quale un processo può terminare forzatamente l'esecuzione di un altro processo. Aggiungiamo però la seguente limitazione: un processo può terminare solo sé stesso, oppure uno dei suoi *discendenti* (cioè i processi creati da esso stesso, oppure creati da questi, e così via). È un errore se un processo tenta di terminare un altro processo che non appartiene alla sua discendenza.

Per realizzare il meccanismo aggiungiamo la seguente primitiva:

```
bool kill(natl id);
```

La primitiva deve terminare forzatamente il processo di identificatore `id`, qualunque cosa esso stesse facendo (quindi anche se era bloccato su un semaforo o sospeso sulla coda del timer). Se non esiste alcun processo di identificatore `id`, la primitiva restituisce `false`, altrimenti `true`. Abortisce il processo in caso di errore.

Per tenere traccia della discendenza aggiungiamo il seguente campo al descrittore di processo:

```
struct des_proc *parent;
```

Il campo deve puntare al più giovane antenato vivente del processo. Alla creazione, punta al processo padre (quello che ha invocato la `activate_p()`); se il processo padre termina prima del figlio, il campo `parent` dovrà essere opportunamente aggiornato.

Non vogliamo modificare troppo il nucleo, quindi non aggiungiamo particolari strutture dati per trovare il `proc_elem` corrispondente ad un dato `id`, anche se questo comporta costose ricerche in tutti i posti in cui il `proc_elem` potrebbe trovarsi.

Modificare il file `sistema.cpp` per aggiungere le parti mancanti nella realizzazione di questo meccanismo.

ATTENZIONE: quando si fa terminare forzatamente un processo, tutte le sue risorse devono essere rilasciate come il processo se avesse chiamato `terminate_p()`. Fare anche attenzione a lasciare sempre semafori in uno stato consistente.