Prova pratica di Calcolatori Elettronici

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

23 febbraio 2018

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

```
struct st1 { int vi[4]; };
struct st2 { char vd[4]; };
class cl {
        char v1[4]; char v3[4]; long v2[4];
public:
        cl(st1& ss);
        cl(st1 s1, int ar2[]);
        cl elab1(char ar1[], st2 s2);
        void stampa() {
                for (int i = 0; i < 4; i++) cout << (int)v1[i] << ' '; cout << endl;
                for (int i = 0; i < 4; i++) cout << (int)v2[i] << ' '; cout << endl;
                for (int i = 0; i < 4; i++) cout << (int)v3[i] << ' '; cout << endl << endl;
        }
};
```

Realizzare in Assembler GCC le funzioni membro seguenti.

```
cl::cl(st1& ss)
{
        for (int i = 0; i < 4; i++) {
                v1[i] = ss.vi[i]; v2[i] = v1[i] / 2;
                v3[i] = 2 * v1[i];
        }
}
cl::cl(st1 s1, int ar2[])
{
        for (int i = 0; i < 4; i++) {
                v1[i] = s1.vi[i]; v2[i] = v1[i] / 4;
                v3[i] = 2 * ar2[i];
        }
}
cl cl::elab1(char ar1[], st2 s2)
{
        st1 s1;
        for (int i = 0; i < 4; i++) s1.vi[i] = ar1[i] + i;
        cl cla(s1);
        for (int i = 0; i < 4; i++) cla.v3[i] = s2.vd[i];
        return cla;
}
```

2. Vogliamo aggiungere al sistema una primitiva

```
natl newproc(void f(int), int a, natl prio, natl secs)
```

La primitiva è simile alla `activate_p()` e permette di creare un nuovo processo con corpo `f` e parametro `a`, eseguito a priorità `prio`. A differenza della `activate_p()`, però, il nuovo processo avrà sempre privilegio utente e dovrà terminare (cioè, invocare la primitiva `terminate_p()`) entro `secs` secondi, altrimenti verrà terminato forzatamente. Come la `activate_p()`, la primitiva `newproc()` restituisce l'identificatore del processo creato o `0xffffffff` se non è stato possibile portare a termine la creazione.

Per motivi di convenienza realizziamo la nuova primitiva nel modulo I/O e sfruttiamo i seguenti meccanismi già disponibili nel modulo I/O e nel modulo sistema:

- modulo I/O: possiamo usare le funzioni `natl startwatchdog(natl secs)` e `stopwatchdog(natl wd)`;
- modulo sistema: possiamo usare la primitiva `bool call_user(void f(int), int a)`.

La funzione `startwatchdog(secs)` avvia un timer di `secs` secondi e ne restituisce l'identificatore (`0xffffffff` se non vi sono più timer disponibili). La funzione `stopwatchdog(wd)` ferma il timer di identificatore `wd`. Se il timer non viene fermato in tempo (cioè, prima che siano passati i `secs` secondi con cui era stato inizializzato), il processo che aveva invocato `startwatchdog()` viene terminato forzatamente.

La primitiva `call_user(f, a)`, invocabile solo dal modulo I/O, trasforma temporaneamente il processo corrente in un processo utente, passando ad eseguire il corpo `f` con parametro `a`. Se e quando il corpo `f` chiama `terminate_p()`, la primitiva `call_user()` torna al chiamante e l'esecuzione prosegue a livello sistema. Se invece il processo utente viene terminato forzatamente, l'intero processo termina la propria esecuzione.

Modificare i file `io.s` e `io.cpp` in modo da realizzare la primitiva `newproc()`.