Prova pratica di Calcolatori Elettronici

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

26 febbraio 2020

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

```
struct st1 { char vi[4]; };
struct st2 { char vd[4]; };
class cl {
        char v1[4]; int v3[4]; long v2[4];
public:
        cl(st1& ss);
        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::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 fornire ai processi la possibilità di bloccarsi in attesa che un altro processo riceva una eccezione, quindi decidere se tale processo deve proseguire nonostante l'eccezioni o essere distrutto. Un processo P deve prima registrarsi, tramite la primitiva `proc_attach(natl id)`, con il processo di identificatore `id`, chiamiamolo Q, di cui vuole controllare la ricezione delle eccezioni. Da questo momento in poi, se Q riceve una eccezione deve essere messo in *pausa*. Diremo che P è il *master* di Q e che Q è lo *slave* di P. Un processo master può registrarsi con un numero qualunque di slave. Una volta registratosi, il processo master può invocare la primitiva `proc_wait()` per bloccarsi in attesa che almeno uno dei suoi slave vada

in pausa (l'attesa può essere nulla se qualche slave era già andato in pausa nel frattempo). La primitiva `proc_wait()` restituisce al processo P l'identificatore di uno dei suoi slave in pausa. In caso di più slave in pausa, la primitiva restituisce l'identificatore dello slave con priorità maggiore. A questo punto il master può terminare la pausa dello slave invocando la primitiva `proc_cont(natl id, bool terminate)`. Il parametro `terminate` permette di decidere se lo slave deve proseguire dal punto in cui aveva ricevuto l'eccezione, o essere distrutto. Nota: gli slave che terminano prima di ricevere una eccezione si scollegano dal master; se il master è bloccato nella `proc_wait()` e tutti gli slave si scollegano, la `proc_wait()` termina restituendo 0xFFFFFFFF; quando un master termina, tutti gli slave vengono scollegati e quelli in pausa vengono distrutti.

Per realizzare questo meccanismo aggiungiamo i seguenti campi al descrittore di processo:

```
des_proc *slaves;
bool is_waiting;
struct proc_elem *paused_slaves;

des_proc *master;
des_proc *next_slave;
natl last_exception;

struct proc_elem *my_proc_elem;
```

I primi tre campi sono relativi ai master, con il seguente significato: `slaves` è una lista di tutti gli slave del master; `is_waiting` vale `true` se il master è in attesa nella `proc_wait()`; `paused_slaves` è una coda che contiene tutti gli slave attualmente in pausa. I secondi tre campi sono realtivi agli slave, con il seguente significato: `master` punta al master dello slave; `next_slave` è usato per creare la lista di tutti gli slave dello stesso master (lista la cui testa è il puntatore `slaves` nel master); `last_exception` contiene il numero dell'ultima eccezione ricevuta dallo slave (o 32 se lo slave aveva invocato `terminate_p()`). Infine, il campo `my_proc_elem`, valido per ogni processo, contiene un puntatore al `proc_elem` associato al processo stesso.

Si modifichino i file `sistema/sistema.s` e `sistema/sistema.cpp` per implementare il meccanismo e le seguenti primitive (abortiscono il processo in caso di errore):

- `bool proc_attach(natl id)`: (tipo 0x59, già realizzata) La primitiva restituisce `false` se il processo che la invoca è uno slave, oppure se il processo `id` non esiste oppure è già un master. È un errore se il processo P è già master o cerca di diventare master di se stesso. Altrimenti fa in modo che P diventi il master di `id` e restituisce `true`.
- `natl proc_wait()`: (tipo 0x5a, da realizzare): attende che almeno un processo slave vada in pausa per la ricezione di una eccezione (nota: si trascurino i page faulti, tipo 14, e le interruzioni non mascherabili, tipo 2) e restituisce l'identificatore dello slave in pausa a priorità maggiore; restituisce `0xFFFFFFFF` se non ci sono slave (o se sono tutti terminati);
- `void proc_cont(natl id, bool terminate)`: (tipo 0x5c, da realizzare): termina la pausa del processo slave di identificatore `id`. Se `terminate` vale `true` il processo viene distrutto, altrimenti riparte dallo stato salvato alla ricezione dell'interruzione. È un errore invocare questa primitiva se il processo non è master, oppure se il processo di identificatore `id` non esiste, o non è uno slave in pausa del processo che invoca la primitiva.