Prova pratica di Calcolatori Elettronici

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

17 luglio 2024

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

```
struct st {
        long vv2[4];
        char vv1[4];
};
class cl {
        st s;
public:
        cl(char v[]);
        void elab1(st& ss, int d);
        void stampa()
        {
                for (int i = 0; i < 4; i++)
                        cout << (int)s.vv1[i] << ' ';
                cout << '\t';
                for (int i = 0; i < 4; i++)
                        cout << s.vv2[i] << ' ';
                cout << endl;
                cout << endl;
        }
};
```

   Realizzare in Assembler GCC le funzioni membro seguenti.

```
void cl::elab1(st& ss, int d)
{
        for (int i = 0; i < 4; i++) {
                if (d > ss.vv2[i])
                        s.vv1[i] += ss.vv1[i];
                s.vv2[i] = d + i;
        }
}
```

2. Vogliamo permettere ai processi di livello sistema di essere notificati quando si verificano dei particolari eventi all'interno del modulo sistema. Ci limitiamo a considerare solo gli eventi corrispondenti alla terminazione (o abort) di un processo di livello utente. I processi di livello sistema interessati a questo tipo di eventi devono prima *registrarsi* tramite una nuova primitiva `evreg()`; da quel punto in poi verranno *notificati* ogni volta che un processo utente termina (o abortisce).

   Per realizzare il meccanismo modifichiamo la primitiva `wfi()` in modo che possa attendere notifiche di eventi, oltre che richieste di interruzione. La primitiva modificata deve restituire un valore con il seguente significato:

- 1: è stata ricevuta una richiesta di interruzione (comportamento normale della `wfi()`);
- 2: è stata ricevuta una notifica di terminazione;
- 3: entrambe le cose (notifica di terminazione e richiesta di interruzione).

Se la `wfi()` restituisce 2 o 3, il processo deve poi *rispondere* alla notifica invocando la primitiva `evget()`, che restituisce l'id del processo terminato. La primitiva `evget()` può essere invocata più volte (anche senza aver prima invocato `wfi()`) e non è mai bloccante: se non ci sono notifiche pendenti si limita a restituire 0.

Il meccanismo, come descritto, impone anche di modificare gli handler, in quanto ora può accadere che un processo invochi `wfi()`, inviando l'EOI all'APIC e bloccandosi, e poi si risvegli a causa di una notifica. Una richiesta di interruzione può dunque arrivare mentre il processo non è bloccato dentro la `wfi()`; in quel caso l'handler non può mettere il processo forzatamente in esecuzione, ma deve limitarsi a settare un flag nel descrittore del processo. Il processo noterà questo flag e agirà di conseguenza la prossima volta che invoca `wfi()`.

Più processi possono registarsi per gli eventi, e ciascuno di essi deve ricevere tutte le notifiche generate dal momento in cui si è registrato in poi. Diciamo che la notifica di un evento è *in corso* se i processi registrati sono stati notificati, ma non hanno ancora risposto tutti. Quando tutti i processi registrati hanno risposto, diciamo che la notifica è *completata*. Un nuovo evento può essere notificato solo dopo che la notifica del precedente è stata completata. Infine, per evitare che gli id dei processi terminati vengano riusati prima che i processi registrati abbiano avuto il tempo di riceverli, i processi terminati vengono distrutti solo al completamento della notifica.

Per realizzare il meccanismo aggiungiumo i seguenti campi al descrittore di processo:

```
bool registrato;
bool notificato;
bool bloccato;
bool ricevuto_intr;
```

Dove: `registrato` è true se il processo è registrato per la notifica degli eventi; `notificato` è true se il processo ha ricevuto una notifica a cui non ha ancora risposto; `bloccato` è true se il processo è bloccato nella `wfi()`; `ricevuto_intr` è true se è arrivata una richiesta di interruzione mentre il processo non era bloccato nella `wfi()`.

Aggiungiamo inoltre le seguenti variabili globali:

```
des_proc *in_notifica;
natq risposte_mancanti;
des_proc *terminati;
```

Dove: `in_notifica` punta al descrittore del processo (terminato) la cui notifica è ancora in corso (nullptr se non ci sono notifiche in corso); `risposte_mancanti` conta quanti processi registrati devono ancora rispondere alla notifica in corso (0 se non ci sono notifiche in corso); `terminati` è una coda di processi terminati la cui notifica è stata rimandata perché ce n'era già un'altra in corso.

Modifichiamo gli handler e la `wfi()` come descritto e aggiungiamo le seguenti primitive (invocabili solo da livello sistema):

- `bool evreg()`: registra il processo per la ricezione delle notifiche; restituisce false in caso di errore (processo già registrato, notifica in corso);
- `natq evget()`: risponde ad eventuali notifiche; restituisce l'id di un processo terminato, o 0 se non ci sono notifiche o se il processo non era registrato.

Modificare il file `sistema.cpp` per completare le parti mancanti.