Prova pratica di Calcolatori Elettronici

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

10 gennaio 2024

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::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. Colleghiamo al sistema delle periferiche PCI di tipo `ce`, con vendorID `0xedce` e deviceID `0x1234`. Ogni periferica `ce` usa 16 byte nello spazio di I/O a partire dall'indirizzo base specificato nel registro di configurazione BAR0, sia b.

   Le periferiche `ce` sono semplici periferiche con un registro RBR di ingresso, tramite il quale è possibile leggere il prossimo byte disponibile. Se abilitata scrivendo 1 nel registro CTL, la perifica invia una richiesta di interruzione quando il registro RBR contiene un nuovo byte. La periferica non invia nuove richieste di interruzione fino a quando il registro RBR non viene letto.

   Vogliamo realizzare una primitiva `ceread_n_to()` che permetta di leggere n byte da una periferica `ce`, ma specificando un *timeout*: se l'operazione non si conclude entro il timeout, la primitiva annulla il trasferimento e restituisce soltanto i byte ricevuti fino a quel punto (eventualmente nessuno).

   Per evitare che il vincolo temporale venga violato a causa dell'esecuzione di processi a più alta priorità, realizziamo la primitiva `ceread_n_to()` con un driver, invece di usare handler e processo esterno: in

questo modo sia il driver che la primitiva si trovano nel modulo sistema, il driver è atomico e la primitiva può essere interrotta solo quando chiama le primitive semaforiche. Si noti che la primitiva potrebbe comunque dover attendere a causa della mutua esclusione sull'accesso alla periferica, e questo tempo entra nel conteggio del timeout.

Per descrivere le periferiche `ce` aggiungiamo le seguenti strutture dati al modulo sistema:

```
struct des_ce {
    ioaddr iCTL, iSTS, iRBR;
    bool enabled;
    char *buf;
    natl quanti;
    natl sync;
    natl mutex;
};
des_ce array_ce[MAX_CE];
natl next_ce;
```

Ogni `des_ce` descrive una periferica `ce`. I campi `iCTL`, `iSTS` e `iRBR` contengono gli indirizzi nello spazio di I/O dei registri dell'interfaccia (il registro STS può essere ignorato); il campo `enabled` vale `true` solo se è in corso una operazione di trasferimento, e dunque l'interfaccia è abilitata a inviare richieste di interruzione; se è in corso un trasferimento, `quanti` conta quanti byte restano da leggere e `buf` dice dove andrà scritto il prossimo byte; `sync` è l'indice di un semaforo di sincronizzazione e `mutex` è l'indice di un semaforo di mutua esclusione. Le periferiche `ce` installate sono numerate da 0 a `next_ce` meno uno. Il descrittore della periferica numero `x` si trova in `array_ce[x]`.

La primitiva da aggiungere è

- `bool ceread_n_to(natl id, char *buf, natl& quanti, natl to)` (tipo 0x2a): avvia una operazione di trasferimento di `quanti` byte dalla periferica `ce` numero `id` verso il buffer `buf`, con timeout `to`. Restituisce `true` se l'operazione si è conclusa entro il timeout, altrimenti restituisce `false` e scrive in `quanti` il numero di byte effettivamente trasferiti; abortisce il processo in caso di errore (`id` non valido, Cavallo di Troia, timeout pari a zero).

È possibile utilizzare la primitiva `natl sem_wait_to(natl sem, natl to)` che esegue una wait sul semaforo `sem` con timeout `to` (che deve essere maggiore di zero). La primitiva si comporta come una normale `sem_wait(sem)`, ma se il processo non viene risvegliato entro `to` termina l'attesa e restituisce 0; altrimenti restituisce il tempo che mancava allo scatto del timeout. Si consideri trascurabile il tempo speso eseguendo istruzioni non interrompibili (in altre parole, l'unico tempo che conta ai fini del timeout è quello speso mentre il processo è bloccato su qualche semaforo).

Modificare i file `sistema.s` e `sistema.cpp` in modo da realizzare le parti mancanti (incluso il driver `void c_driver_ce(natl id)`).

Suggerimento: quando si annulla l'operazione `ceread_n_to()` si faccia attenzione a non lasciare la periferica con le interruzioni abilitate, e ci si assicuri che il driver gestisca le richieste solo se è ancora in corso una operazione di trasferimento.