Prova pratica di Calcolatori Elettronici

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

19 luglio 2023

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. 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.

Nel sistema è installata un'unica perififerica `ce`, ma vogliamo fare in modo che gli utenti possano usarne diverse versioni "virtuali", dette `vce`. Ciascuna periferica `vce` contiene un buffer (realizzato con un array circolare) che può contenere un certo numero di byte letti dalla periferica `ce` (al massimo `VCE_BUFSIZE`). Ad ogni istante una sola periferica `vce` è *attiva*. Le interruzioni della periferica `ce` sono sempre abilitate e il modulo I/O provvede a leggere i byte e a depositarli nel buffer della `vce` attiva (se il buffer è pieno il byte letto viene perso).

Ciascuna `vce` è identificata da un numero da 0 a `VCE_NUM` meno 1. Gli utenti possono usare la primitiva

```
void vceread_n(natl vn, char *vetti, natq quanti)
```

per leggere `quanti` byte dalla `vce` numero `vn`, ricevendoli nel vettore `vetti`. La periferica copierà i byte dal buffer interno della `vce`, sospendendosi fino a quando non sono stati ricevuti tutti. I processi possono accedere concorrentemente a `vce` diverse, ma solo uno alla volta su ciascuna `vce`. In qualunque momento, un processo può cambiare la `vce` attiva invocando la primitiva `vceswitch(vn)`.

Per descrivere le periferiche `ce` e `vce` aggiungiamo le seguenti strutture dati al modulo I/O:

```
struct vce_buf {
        char buf[VCE_BUFSIZE];
        natl head;
        natl tail;
        natl n;
        natl mutex;
};
struct vce_des {
        vce_buf buf;
        natl mutex;
        natl sync;
};
struct ce_des {
        vce_des vces[VCE_NUM];
        natl active;
        ioaddr iRBR, iCTL;
        natl mutex;
} ce;
```

La struttura `vce_buf` descrive i buffer interni alle `vce`: `buf` è l'array circolare; i byte vanno inseriti all'indice `tail` ed estratti dall'indice `head`; il campo `n` conta il numero di byte contenuti nell'array; il campo `mutex` è l'indice di un semaforo di mutua esclusione per gli accessi ai campi del `vce_buf`.

La struttura `vce_des` descrive una periferica `vce`: `buf` è il buffer interno; `mutex` è l'indice di un semaforo di mutua esclusione per l'utilizzo del `vce` e `sync` è un semaforo che contiene un gettone per ogni byte contenuto nel buffer.

Infine, la struttura `ce_des` descrive la periferica `ce`: il campo `vces` contiene un descrittore per ogni `vce`; il campo `active` è l'indice della `vce` attiva; i campi `iRBR` e `iCTL` contengono gli indirizzi dei rispettivi registri; il campo `mutex` è l'indice di un semaforo di mutua esclusione per l'accesso al campo `active`.

Modificare i file `io.s` e `io.cpp` in modo da realizzare le parti mancanti.