Prova pratica di Calcolatori Elettronici

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

15 giugno 2016

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

```
struct st1 {
        char vc[8];
};
struct st2 {
        long vd[4];
};
class cl {
        st1 s;
        int v[4];
public:
        cl(char c, st1& s2);
        void elab1(st1 s1, st2& s2);
        void stampa()
        {
                for (int i = 0; i < 8 ;i++) cout << s.vc[i] << ' '; cout << endl;
                for (int i = 0; i < 4; i++) cout << v[i] << ' '; cout << endl << endl;
        }
};
```

Realizzare in Assembler GCC le funzioni membro seguenti.

```
cl::cl(char c, st1& s2)
{
        for (int i = 0; i < 8; i++) {
                s.vc[i] = c + i;
        }
        for (int i = 0; i < 4; i++) {
                v[i] = s2.vc[i] - s.vc[i];
        }
}
void cl::elab1(st1 s1, st2& s2)
{
        cl cla('f', s1);
        for (int i = 0; i < 4; i++) {
                if (s.vc[i] < s1.vc[i])
                        s.vc[i] = cla.s.vc[i];
                if (v[i] < cla.v[i])
                        v[i] = cla.v[i] + 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 periferiche di ingresso in grado di operare in PCI Bus Mastering. I registri accessibili al programmatore sono i seguenti:

   1. **BMPTR** (indirizzo b, 4 byte): puntatore al buffer di destinazione;
   2. **BMLEN** (indirizzo $b + 4$, 4 byte): numero di byte da trasferire;
   3. **CMD** (indirizzo $b + 8$, 4 byte): registro di comando;
   4. **STS** (indirizzo $b + 12$, 4 byte): registro di stato.

   L'interfaccia è in grado di trasferire in memoria BMLEN byte, partendo dall'indirizzo fisico contenuto in BMPTR e proseguendo agli indirizzi fisici contigui. Per iniziare il trasferimento è necessario scrivere 1 nel registro di comando. L'interfaccia invia una richiesta di interruzione dopo aver trasferito l'ultimo byte. Le interruzioni sono sempre abilitate e la lettura del registro di stato funziona da risposta alle richieste di interruzione (l'interfaccia non invia una nuova richiesta se una richiesta precedente non ha ancora avuto risposta).

   Vogliamo fornire all'utente una primitiva

   ```
   cedmaread(natl id, char *buf, natl quanti)
   ```

   che permetta di leggere `quanti` byte dalla periferica numero `id` (tra quelle di tipo `ce`) copiandoli nel buffer `buf`. Notare che `buf` è un indirizzo virtuale e il buffer potrebbe attraversare più pagine virtuali: la primitiva dovrà funzionare in ogni caso, eventualmente eseguendo più trasferimenti.

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

   ```
   struct des_ce {
           ioaddr iBMPTR, iBMLEN, iCMD, iSTS;
           natl sync;
           natl mutex;
           char *buf;
           natl quanti;
   };
   des_ce array_ce[MAX_CE];
   natl next_ce;
   ```

   La struttura `des_ce` descrive una periferica di tipo `ce` e contiene al suo interno gli indirizzi dei registri BMPTR, BMLEN, CND e STS, l'indice di un semaforo inizializzato a zero (`sync`), l'indice di un semaforo inizializzato a 1 (`mutex`), il numero di byte che restano da trasferire (`quanti`) e l'indirizzo virtuale a cui vanno trasferiti (`buf`).

   I primi `next_ce` elementi del vettore `array_ce` contengono i descrittori, opportunamente inizializzati, delle periferiche di tipo `ce` effettivamente rilevate in fase di avvio del sistema. Ogni periferica è identificata dall'indice del suo descrittore.

   Modificare i file `io.s` e `io.cpp` in modo da realizzare la primitiva come descritto.

   Nota: il modulo sistema mette a disposizione la primitiva

   ```
   addr trasforma(addr ind_virtuale)
   ```

   che restituisce l'indirizzo fisico che corrisponde all'indirizzo virtuale passato come argomento, nello spazio di indirizzamento del processo in esecuzione.