Prova pratica di Calcolatori Elettronici

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

29 giugno 2022

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

```
struct st {
        char vv1[6];
        long vv2[3];
};
class cl {
        st s;
public:
        cl(char v[]);
        void elab1(int d, st& ss);
        void stampa()
        {
                for (int i = 0; i < 6; i++)
                        cout << (int)s.vv1[i] << ' ';
                cout << '\t';
                for (int i = 0; i < 3; i++)
                        cout << s.vv2[i] << ' ';
                cout << "\n\n";
        }
};
```

Realizzare in Assembler GCC le funzioni membro seguenti.

```
cl::cl(char v[])
{
        for (int i = 0; i < 3; i++) {
                s.vv1[i] = s.vv1[i + 3] = s.vv2[i] = v[i];
        }
}
void cl::elab1(int d, st& ss)
{
        for (int i = 0; i < 3; i++) {
                if (d >= ss.vv2[i]) {
                        s.vv1[i] += ss.vv1[i];
                        s.vv2[i] -= d;
                } else {
                        s.vv1[i + 3] -= ss.vv1[i];
                        s.vv2[i] += d;
                }
        }
```

```
}
```

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.

   La periferiche `ce` sono periferiche di ingresso che operano in bus mastering e sono in grado di generare richieste di interruzione. Ciascuna periferica contiene un certo numero di canali, numerati da 0 a `MAX_CHAN` -1, ciascuno in grado di operare independentemente dagli altri. Questo permette di avere più trasferimenti attivi contemporaneamente. Per attivare un trasferimento in bus mastering su un canale è necessario scrivere l'indirizzo di destinazione e la quantità di byte da trasferire nei registri BMPTR e BMLEN, rispettivamente, e poi scrivere il numero del canale nel registro CHN. La periferica possiede un registro STS con un bit per ogni canale (bit 0 per il canale 0, bit 1 per il canale 1, e così via). Quando si scrive il numero di un canale in CNH, la periferica mette a 1 il corrispondente bit di STS. Quando il traferimento in bus mastering è terminato, la periferica mette in ogni caso a zero il corrispondente bit di STS, quindi invia una richiesta di interruzione, ma solo se non ve n'è già un'altra in attesa di risposta. La lettura di STS funge da risposta alle richieste di interruzione. Si noti che, quando si risponde ad una richiesta di interruzione, più di un canale potrebbe aver terminato il proprio trasferimento, cioè più di un bit di STS potrebbe essere passato da 1 a 0.

   I registri accessibili al programmatore sono i seguenti:

   1. **CHN** (indirizzo b, 4 byte, lettura/scrittura): scrivendo i in questo registro si attiva il canale i-esimo, con i valori correnti di BMPTR e BMLEN; la periferica ignora la richiesta se il canale era già attivo, o se i non corrisponde a nessun canale;
   2. **STS** (indirizzo $b + 4$, 4 byte, lettura): registro di stato (vedere sopra);
   3. **BMPTR** (indirizzo $b + 8$, 4 byte, lettura/scrittura): indirizzo di destinazione del trasferimento;
   4. **BMLEN** (indirizzo $b + 12$, 4 byte, lettura/scrittura): numero di byte da trasferire;

   La periferica usa il contenuto di BMPTR e BMLEN solo quando si scrive in CHN il numero di un canale non precedentemente attivo.

   Vogliamo fornire all'utente una primitiva

   ```
   void ceread(natl id, char *buf, natl quanti);
   ```

   Il parametro `id` identifica una delle periferiche `ce` installate. La primitiva permette di leggere da tale periferica una sequenza di `quanti` byte dal primo canale (in ordine di identificatore) non attualmente già attivo. Se tutti i canali sono attivi la primitiva attende che se ne liberi uno. I byte letti saranno scritti a partire dall'indirizzo `buf`. La primitiva deve funzionare correttamente anche se `buf` attraversa più pagine virtuali.

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

   ```
   static const int MAX_CHAN = 4;
   struct des_chan {
           natl sync;
           char *buf;
           natl quanti;
           bool active;
   };
   struct des_ce {
           ioaddr iCHN, iSTS, iBMPTR, iBMLEN;
           natl mutex;
           natl free_chan;
   ```

```
        des_chan chan[MAX_CHAN];
};
static const int MAX_CE = 16;
des_ce array_ce[MAX_CE];
natl next_ce;
```

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. I descrittori `des_ce` contengono gli indirizzi dei registri della periferica (`iCHN`, `iSTS`, `iBMPTR` e `iBMLEN`), l'indice di un semaforo `mutex` per l'accesso in mutua esclusione alle risorse condivise tra i canali (compresi, in particolare, i registri della peferica), un semaforo `free_chan` che contiene un gettone per ogni canale libero, e un array di descrittori `des_chan`, con un elemento per ogni canale. I descrittori `des_chan` contengono i dati del trasferimento attivo sul canale (`buf` e `quanti`), un semaforo `sync` su cui si sospende il processo che ha richesto il trasferimento, e un booleano `active` per marcare i canali attualmente attivi.

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