Prova pratica di Calcolatori Elettronici

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

6 febbraio 2019

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

```
struct st1 { int vi[4]; };
struct st2 { char vd[4]; };
class cl {
        long v2[4]; char v1[4]; char v3[4];
public:
        cl(st1 ss);
        cl(st1& s1, int ar2[]);
        cl elab1(const 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(st1 ss)
{
        for (int i = 0; i < 4; i++) {
                v1[i] = ss.vi[i]; v2[i] = ss.vi[i] * 2;
                v3[i] = 4 * ss.vi[i];
        }
}
cl::cl(st1& s1, int ar2[])
{
        for (int i = 0; i < 4; i++) {
                v1[i] = s1.vi[i]; v2[i] = s1.vi[i] * 8;
                v3[i] = ar2[i];
        }
}
cl cl::elab1(const 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 una perifierica PCI di tipo `ce`, con vendorID `0xedce` e deviceID `0x1234`. Le periferiche `ce` sono schede di rete che operano in PCI Bus Mastering. Il software deve preparare dei buffer vuoti, che la scheda riempie autonomamente con messaggi ricevuti dalla rete.

   Per permettere al software di operare in parallelo con la ricezione, la scheda usa una coda circolare di descrittori di buffer. Ogni descrittore deve contenere l'indirizzo fisico di un buffer, la sua lunghezza in byte e un flag di "fine messaggio". La coda di buffer ha 8 posizioni, numerate da 0 a 7. Il flag è necessario perché la scheda può usare più di un buffer per un singolo messaggio, se questo è più lungo della dimensione del buffer.

   La scheda possiede due registri, `HEAD`, di sola lettura, e `TAIL`, di lettura/scrittura. I due registri contengono numeri di posizioni all'interno della coda. La scheda può usare soltanto i descrittori che vanno da `HEAD` in avanti (circolarmente) senza toccare `TAIL`.

   All'avvio il software prepara tutti i buffer e inizializza tutti i descrittori, quindi scrive 7 in `TAIL`. In questo modo la scheda può usare i descrittori da 0 a 6 (uno deve essere sempre non utilizzato, per distinguere gli stati di coda piena e coda vuota).

   Ogni volta che la scheda ha terminato di ricevere un messaggio lo copia in uno o più buffer partendo da quello puntato dal descrittore indicato da `HEAD` e andando avanti (circolarmente) e incrementando ogni volta `HEAD`, eventualmente fermandosi se raggiunge `TAIL`. Oltre a copiare il messaggio, la scheda modifica i descrittori utilizzati per scrivervi il numero di byte scritti nel corrispondente buffer (sovrascrivendo il campo del descrittore che conteneva la lunghezza del buffer) e settando opportunamente il flag di "fine messaggio". In un momento qualunque (anche prima di aver terminato un messaggio), la scheda può inviare una richiesta di interruzione per segnalare che `HEAD` è stato modificato (e dunque alcuni descrittori sono stati usati). La lettura di `HEAD` funge da risposta alla richiesta. I descrittori utilizzati saranno quelli che si trovano tra l'ultima posizione letta da `HEAD` (inclusa) e la nuova (esclusa).

   Ogni periferica `ce` usa 16 byte nello spazio di I/O a partire dall'indirizzo base specificato nel registro di configurazione BAR0, sia b. I registri accessibili al programmatore sono i seguenti:

   1. **HEAD** (indirizzo b, 4 byte): posizione di testa;
   2. **TAIL** (indirizzo $b + 4$, 4 byte): posizione di coda;
   3. **RING** (indirizzo $b + 8$, 4 byte): indirizzo fisico del primo descrittore della coda circolare;

   Vogliamo fornire all'utente due primitive

   ```
   natq waitnet();
   void receive(char *buf, natq len);
   ```

   La prima primitiva attende l'arrivo di un messaggio completo e ne restituisce la lunghezza. La seconda copia nel buffer `buf` i primi `len` byte dell'ultimo messaggio ricevuto (se il messaggio è più lungo, i rimanenti byte vengono scartati). Se si chiama più volte `waitnet` senza chiamare `receive`, la `waitnet` continua a restituire la lunghezza dell'ultimo messaggio senza attenderne uno nuovo. Se si chiama più volte `receive` senza aver chiamato `waitnet`, il comportamento è indefinito (si può assumere che gli utenti non facciano quest'ultima cosa). I messaggi sono sempre lunghi almeno 8 byte.

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

   ```
   struct slot {
           natl addr;
           natw len;
           natw eop;
   };
   const natl DIM_RING = 8;
   struct des_ce {
           natw iHEAD, iTAIL, iRING;
   ```

```
        slot s[DIM_RING];
        natl mutex;
        natl slots_ready;
        natl old_head;
        natl toread;
        natl last_idx;
        natl last_len;
} net;
```

La struttura `slot` rappresenta un descrittore di buffer (indirizzo fisico in `addr` e lunghezza in `len`). Dopo che la scheda ha usato un descrittore, `len` contiene il numero di byte scritti nel buffer e `eop` è diverso da zero se il messaggio è terminato con questo descrittore. La struttura `des_ce` descrive una periferica di tipo `ce` e contiene al suo interno: gli indirizzi dei registri `HEAD`, `TAIL` e `RING`; la coda circolare di descrittori, `s`; l'indice di un semaforo di mutua esclusione (`mutex`); l'indice di un semaforo `slots_ready`, inizializzato a zero; il campo `old_head`, utile a memorizzare l'ultimo valore letto dal registro `HEAD`; il campo `toread`, utile a memorizzare l'indice del prossimo slot da leggere; i campi `last_idx` e `last_len` che contengono, rispettivamente, l'indice del primo slot e la lunghezza complessiva (in byte) dell'ultimo messaggio ricevuto ma non ancora copiato (se `last_len` vale zero non ci sono messaggi da copiare).

Modificare i file `io.s` e `io.cpp` in modo da realizzare la primitiva come descritto. Controllare eventuali problemi di Cavallo di Troia.