Prova pratica di Calcolatori Elettronici (nucleo v6.*)

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

13 giugno 2018

1. 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 i messaggi in dei buffer di memoria e la scheda è in grado di spedirli autonomamente.

   Per permettere al software di operare in parallelo con l'invio, la scheda usa una coda circolare di descrittori di messaggi. Ogni descrittore deve contenere l'indirizzo fisico di un messaggio e la sua lunghezza in byte. La coda ha 8 posizioni, numerate da 0 a 7. La scheda possiede due registri, `HEAD`, di sola lettura, e `TAIL`, di lettura/scrittura.

   I due registri contengono numeri di posizioni e inizialmente sono entrambi pari a zero. Per inviare un nuovo messaggio il software deve: 1) inizializzare il descrittore numero `TAIL`; 2) incrementare il di 1 (modulo 8) il contenuto di `TAIL`. Più messaggi possono essere accodati mentre altri (accodati precedentemente) sono ancora in fase di invio da parte della scheda. Ogni volta che la scheda ha terminato di inviare un messaggio incrementa (modulo 8) il contenuto di `HEAD` e, se non sta aspettando una risposta ad una richiesta di interruzione precedente, invia una nuova richiesta di interruzione. La lettura di `HEAD` funge da risposta alla richiesta. È dunque possibile (e, anzi, normale) che quando il software legge `HEAD` questo sia avanzato di più di una posizione rispetto all'ultima lettura: vuol semplicemente dire che tra le due letture la scheda ha finito di inviare più di un messaggio. I descrittori dei messaggi inviati 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;

   Supponiamo che ogni computer collegato alla rete possieda un indirizzo numerico di 4 byte. Ogni computer possiede un'unica perifica di tipo `ce`. Vogliamo fornire all'utente una primitiva

   ```
   bool send(natl dst, char *msg, natl len)
   ```

   che permetta di inviare il messaggio puntato da `msg`, e lungo `len` byte, al computer di indirizzo `dst`. Per far questo, la primitiva alloca un buffer e vi scrive prima una "intestazione" con tre campi (ciascuno grande 4 byte): l'indirizzo del computer che invia (preso da una variabile globale, `myaddr`), l'indirizzo `dst` e la lunghezza `len`. Subito dopo l'intestazione, la primitiva copia nel buffer il messaggio `msg`. Infine, invia il buffer usando la periferica `ce`. La lunghezza del messaggio dell'utente (esclusa l'intestazione) deve essere inferiore a `MAX_PAYLOAD`. La primitiva restituisce `false` in caso di messaggio troppo lungo o se non è stato possibile allocare il buffer, o se la coda circolare era piena; restituisce `true` altrimenti. La primitiva ritorna *senza attendere* che il messaggio sia stato spedito. La memoria allocata per il buffer dovrà essere liberata *dopo* che il messaggio è stato effettivamente spedito.

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

```
struct slot {
        natl addr;
        natl len;
};
struct des_ce {
        natw iHEAD, iTAIL, iRING;
        natl mutex;
        slot *s;
// altri campi
} net;
```

La struttura `slot` rappresenta un descrittore di buffer (indirizzo fisico in `addr` e lunghezza in `len`). La struttura `des_ce` descrive una periferica di tipo `ce` e contiene al suo interno gli indirizzi dei registri HEAD, TAIL e RING, il puntatore `s` alla coda circolare di descrittori, l'indice di un semaforo inizializzato a 1 (`mutex`) ed eventuali altri campi che dovessero essere utili.

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