Prova pratica di Calcolatori Elettronici

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

19 settembre 2018

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

```
struct st1 { char vc[4]; }; struct st2 { int vd[4]; };
class cl {
        st1 c1; st1 c2;
        long v[4];
public:
        cl(char c, st2& s);
        void elab1(st1 s1, st2 s2);
        void stampa()
        {
                for (int i=0; i < 4; i++) cout << c1.vc[i] << ' '; cout << "\n";
                for (int i=0; i < 4; i++) cout << c2.vc[i] << ' '; cout << "\n";
                for (int i=0; i < 4; i++) cout << v[i] << ' '; cout << "\n\n";
        }
};
```

   Realizzare in Assembler GCC le funzioni membro seguenti.

```
cl::cl(char c, st2& s2) {
        for (int i = 0; i < 4; i++) {
                c1.vc[i] = c; c2.vc[i] = c++;
                v[i] = s2.vd[i] + c2.vc[i];
        }
}
void cl::elab1(st1 s1, st2 s2) {
        cl cla('a', s2);
        for (int i = 0; i < 4; i++) {
                if (c2.vc[i] <= s1.vc[i]) {
                        c1.vc[i] = i + cla.c2.vc[i];
                        v[i] = i - cla.v[i];
                }
        }
}
```

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 "pacchetti" in dei buffer di memoria e la scheda è in grado di spedirli autonomamente. Ogni buffer ha una dimensione massima di 64 byte, e dunque questa è anche la massima dimensione di ogni pacchetto.

Per permettere al software di operare in parallelo con l'invio, la scheda usa una coda circolare di descrittori di buffer. Ogni descrittore deve contenere l'indirizzo fisico di un buffer 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. È possibile inviare più di un pacchetto per volta. Per inviare n pacchetti il software deve: 1) inizializzare i descrittori dal numero `TAIL` al numero $\texttt{TAIL} + n - 1$; 2) incrementare di n (modulo 8) il contenuto di `TAIL`. Ogni volta che la scheda ha terminato di inviare uno o più pacchetti, 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.

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`. La primitiva permette all'utente di inviare messaggi più grandi del massimo permesso dalla dimensione dei buffer della scheda, preparando e inviando una *sequenza* di pacchetti, ciascuno dei quali contiene parte del messaggio. Non è comunque possibile inviare un messaggio che richieda più pacchetti di quanti ne possono essere contenuti in un ring: in questo caso, la primitiva abortisce il processo.

I pacchetti sono composti da una "intestazione", creata dalla primitiva, seguita dai byte che l'utente vuole inviare, presi da `msg`. L'intestazione ha quattro campi, ciascuno grande 4 byte: l'indirizzo del computer che invia (preso da una variabile globale, `myaddr`), l'indirizzo `dst`, il campi `len`, che contiene il numero di byte del messaggio contenuti in questo pacchetto, e il campo `seq`, che contiene un numero progressivo, a partire da 0, per ogni pacchetto relativo allo stesso messaggio.

I buffer destinati a contenere i pacchetti sono allocati dalla primitiva stessa e la primitiva restituisce `false` se una allocazione fallisce. La primitiva attende (eventualmente sospendendo il processo) che vi sia un numero di descrittori liberi sufficiente per tutti i pacchetti da inviare, prepara tutti i pacchetti e ordina alla scheda di inviarli, ma poi ritorna *senza attendere* che siano stati effettivamente spediti. La memoria allocata per ogni buffer dovrà essere liberata *dopo* che il pacchetto contenuto 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_net {
        natw iHEAD, iTAIL, iRING;
        slot s[DIM_RING];
        natl mutex;
        natl sync;
natl old_head;
bool sender_waiting;
msg *m[DIM_RING];
} net;
```

La struttura `slot` rappresenta un descrittore di buffer (indirizzo fisico in `addr` e lunghezza in `len`). La struttura `des_net` 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 inizializzato a 1 (`mutex`) di uno inizializzato a 0 (`sync`). Il capo `old_head` contiene il valore letto dal registro `HEAD` nell'ultima interruzione. mentre l'array `m` serve a ricordare gli indirizzi virtuali dei buffer allocati nella coda. Il campo `sender_wairing` deve valere `true` se un processo sta aspettando che si liberino degli slot per poter trasmettere.

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