Prova pratica di Calcolatori Elettronici

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

22 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 {
        char v1[4]; char v3[4]; long v2[4];
public:
        cl(st1& ss);
        cl elab1(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] = v1[i] / 2;
                v3[i] = 2 * v1[i];
        }
}
cl cl::elab1(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 e la sua lunghezza in byte. La coda di buffer ha 4 posizioni, numerate da 0 a 3.

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 3 in `TAIL`. In questo modo la scheda può usare i descrittori da 0 a 2 (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 un buffer partendo da quello puntato dal descrittore indicato da `HEAD`, 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). In un momento qualunque 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 (già realizzata) serve ad attendere l'arrivo del prossimo messaggio e a conoscerne la lunghezza, mentre la seconda (da realizzare) serve a copiarlo nel buffer `buf` di lunghezza `len` (se il messaggio è più lungo, i rimanenti byte vengono scartati). Dopo che un messaggio è stato copiato il corrispondente slot può essere reso nuovamente disponibile per la ricezione di nuovi messaggi.

Più processi possono invocare le due primitive in qualunque ordine. Supponiamo che un processo P invochi `waitnet()` e questa restituisca la lunghezza di un messaggio m. Quando P invocherà `receive()`, questa copierà proprio m, anche se nel frattempo altri processi hanno invocato `waitnet()` e/o `receive()`. In particolare, se un processo $Q \neq P$ invoca `waitnet()` prima che P invochi `receive()`, Q riceverà la lunghezza del prossimo messaggio da copiare, sia p, anche se m non è stato ancora copiato. È dunque possibile che p venga copiato (da Q) prima che P copi m.

Fino a quando P non invoca `receive()`, ogni ulteriore invocazione di `waitnet()` da parte di P continuerà a restituire la lunghezza di m, senza attendere nuovi messaggi. È invece un errore invocare `receive()` senza aver prima chiamato `waitnet()`, e in quel caso la primitiva abortisce il processo.

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

```
struct slot {
        natl addr;
        natl len;
};
const natl DIM_RING = 4;
struct des_ce {
        natw iHEAD, iTAIL, iRING;
        slot s[DIM_RING];
        natl procs[DIM_RING];
        natl mutex;
```

```
        natl slots_ready;
        natl tocopy;
        natl towait;

        natl old_head;
} 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. 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`, una coda `procs` per ricordare quale processo deve copiare ogni messaggio (il valore zero può essere usato per indicare "nessun processo"); l'indice di un semaforo di mutua esclusione (`mutex`); l'indice di un semaforo `slots_ready`, inizializzato a zero; il campo `tocopy`, che memorizza l'indice del prossimo slot ancora da copiare; il campo `towait`, che memorizza l'indice del prossimo slot su cui è necessario attendere che la scheda riceva un messaggio; il campo `old_head`, utile a memorizzare l'ultimo valore letto dal registro `HEAD`;

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

ATTENZIONE: In base a quanto detto, i messaggi possono essere copiati in un ordine diverso da quello in cui sono arrivati. Fare attenzione, quando si avanza `TAIL`, a non passare alla scheda degli slot che contengano messaggi ancora da copiare.