Prova pratica di Calcolatori Elettronici

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

16 gennaio 2019

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

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

   Realizzare in Assembler GCC le funzioni membro seguenti.

```
cl::cl(const char *c, st2 s2)
{
        for (int i = 0; i < 4; i++) {
                s.vc[i] = c[i];
                v[i] = s2.vd[i] + s.vc[i];
        }
}
void cl::elab1(st1 s1, st2 s2)
{
        cl cla(s1.vc, s2);
        for (int i = 0; i < 4; i++) {
                if (s.vc[i] < s1.vc[i])
                        s.vc[i] = cla.s.vc[i];
                if (v[i] <= cla.v[i])
                        v[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 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;

Supponiamo che ogni computer collegato alla rete possieda un indirizzo numerico di 4 byte. Ogni computer possiede un'unica perifica di tipo `ce`. I messaggi che viaggiano sulla rete contengono una "intestazione" con due campi (ciascuno grande 4 byte): l'indirizzo del computer che invia e l'indirizzo `dst` del computer a cui il messaggio è destinato. L'intestazione è seguita dal messaggio vero e proprio.

Vogliamo fornire all'utente una primitiva

```
bool receive(natl& src, char *buf, natq& len)
```

che permetta di ricevere un messaggio nel buffer `buf`. Il parametro `len` contiene inizialmente la dimensione del buffer e, dopo la ricezione, contiene la dimensione effettiva del messaggio ricevuto. Il parametro `src` conterrà l'indirizzo del computer che ha inviato il messaggio. Attenzione: l'utente deve ricevere in `buf` solo il messaggio vero e proprio, esclusa l'intestazione. Si noti che ogni invocazione della primitiva restituisce un solo messaggio: eventuali altri messaggi in coda verranno restituiti alla prossime invocazioni. Se, invece, non vi sono messaggi in coda, la primitiva blocca il processo in attesa che ne arrivi almeno uno. La primitiva restituisce `true` se il buffer `msg` è sufficiente a contenere il prossimo messaggio da ricevere; altrimenti copia in `buf` solo la parte di messaggio che vi entra e restituisce `false`.

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;
} 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 buffer da leggere tramite la `receive`.

Si può assumere che la scheda riceva solo messaggi effettivamente destinati al computer a cui è collegata.

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