

Prova pratica di Calcolatori Elettronici

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

27 gennaio 2025

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(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.

```
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];
            v[i] += cla.v[i];
        }
    }
}
```

2. Colleghiamo al sistema delle periferiche PCI di tipo `ce`, con vendorID `0xedce` e deviceID `0x1234`. Ogni periferica `ce` usa 16 byte nello spazio di I/O a partire dall'indirizzo base specificato nel registro di configurazione BAR0, sia `b`.

Le periferiche `ce` sono semplici periferiche di uscita con un registro TBR (Transmit Buffer Register), nel quale è possibile scrivere il prossimo byte da inviare. La periferica invia una richiesta di interruzione quando ha concluso la trasmissione del contenuto di TBR; fino ad allora, il registro TBR è occupato e ulteriori scritture vengono ignorate.

Nel sistema è installata un'unica periferica `ce`, ma vogliamo fare in modo che gli utenti possano usarne diverse versioni "virtuali", dette VCE, per non dover attendere il completamento delle trasmissioni. Ciascuna periferica VCE contiene un buffer (realizzato con un array circolare) che può contenere un certo numero di byte (al massimo `VCE_BUFSIZE`) in attesa di essere trasmessi sulla periferica reale. Dopo

la conclusione di ogni trasmissione, il processo esterno associato alla periferica estrae un byte da una delle VCE attive e lo trasmette.

Le VCE sono private per processo. I processi che vogliono usare una VCE devono prima allocare la propria istanza usando la primitiva:

```
bool vcenew()
```

La primitiva restituisce **false** se non è stato possibile allocare la VCE. A quel punto il processo può usare la primitiva

```
void vcewrite(char c)
```

per inviare un byte nella VCE. La primitiva non attende la conclusione del trasferimento, ma può bloccare il processo in attesa che si liberi spazio per il byte nel buffer della VCE. La primitiva **vcenew()** abortisce il processo se questo aveva già una VCE; la primitiva **vcewrite()** lo abortisce se non ce l'aveva.

Per descrivere le periferiche **ce** e **vce** aggiungiamo le seguenti strutture dati al modulo I/O:

```
struct vce_des {
    char buf[VCE_BUFSIZE];
    natl head;
    natl tail;
    natl n;
    natl sync;
    bool waiting;
    bool terminated;
};
struct ce_des {
    vce_des *vces[MAX_PROC];
    bool busy;
    ioaddr iTBR;
    natl mutex;
} ce;
```

La struttura **vce_des** descrive i buffer interni alle **vce**: **buf** è l'array circolare; i byte vanno inseriti all'indice **tail** ed estratti dall'indice **head**; il campo **n** conta il numero di byte contenuti nell'array; quando un processo vuole inserire un byte, ma il buffer è pieno, pone **waiting** a **true** e si sospende sul semaforo di sincronizzazione **sync**; il campo **terminated** è **true** se il processo proprietario della VCE è terminato.

La struttura **ce_des** descrive la periferica **ce**: l'array **vces**, indicizzato dai pid dei processi, contiene i puntatori alla **vce_des** di ogni processo (**nullptr** per i processi che non esistono o non hanno una VCE); il campo **busy** è **true** quando c'è una trasmissione in corso; il campo **iTBR** contiene l'indirizzo del registro TBR; il campo **mutex** è l'indice di un semaforo di mutua esclusione per l'accesso al **ce_des** e a tutte le VCE. Per permettere ai vari processi di usare le VCE mentre è in corso una trasmissione precedente, il **mutex** deve essere occupato solo per il tempo strettamente necessario.

Le VCE dei processi devono essere allocate nello heap del modulo I/O (tramite **new**) e deallocate (tramite **delete**) quando il loro processo proprietario termina. L'allocazione avviene nella **vcenew()**, ma la responsabilità della deallocazione ricade sul processo esterno associato alla periferica **ce**: il processo esterno deve deallocare la VCE quando il processo proprietario è terminato e tutti i byte contenuti nella VCE sono stati trasmessi. Per sapere quando il proprietario termina, il processo esterno sfrutta un meccanismo di notifica che è stato aggiunto al modulo sistema. Al momento dell'attivazione, il processo è stato registrato (tramite un nuovo parametro della **activate_pe()**) per la ricezione di notifiche di terminazione. Da quel momento in poi, ogni volta che un processo utente termina, il modulo sistema

notificherà il processo esterno, in particolare rimettendolo in esecuzione se si era bloccato nella `wfi()`. La prima volta che va in esecuzione, e ogni volta che si sveglia dalla `wfi()`, il processo esterno deve invocare la primitiva `evget()` per sapere come mai è stato risvegliato. La primitiva restituisce 0 in caso di richiesta di interruzione (caso normale) e un valore maggiore di zero in caso di notifica di terminazione. Nel secondo caso, il valore restituito è il pid del processo terminato (se si chiama ripetutamente `evget()` senza ripassare dalla `wfi()`, la primitiva restituisce `0xFFFFFFFF`).

Modificare il `io.cpp` in modo da realizzare le parti mancanti.