Prova pratica di Calcolatori Elettronici

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

11 gennaio 2023

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

```
struct st1 { char vc[4]; }; struct st2 { int vd[4]; };
class cl {
        st2 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.vd[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(char c, st2 s2)
{
        for (int i = 0; i < 4; i++) {
                s.vd[i] = c;
                v[i] = s2.vd[i] + s.vd[i];
        }
}
void cl::elab1(st1 s1, st2& s2)
{
        cl cla(s1.vc[3], s2);
        for (int i = 0; i < 4; i++) {
                if (s.vd[i] < s1.vc[i])
                        s.vd[i] = cla.s.vd[i];
                if (v[i] <= cla.v[i])
                        v[i] += cla.v[i];
        }
}
```

2. Il modulo I/O del nucleo contiene le primitive `readhd_n()` e `writehd_n()` che permettono di leggere o scrivere blocchi dell'hard disk. Vogliamo velocizzare le due primitive introducendo una *buffer cache* che mantenga in memoria i blocchi letti più di recente, in modo che eventuali letture di blocchi che si trovino nella buffer cache possano essere realizzate con una semplice copia da memoria a memoria,

invece che con una costosa operazione di I/O. Per quanto riguarda le scritture adottiamo una politica no-allocate/write-through. Per il rimpiazzamento adottiamo la politica LRU (Least Recently Used): se la cache è piena e deve essere letto un blocco non in cache, si rimpiazza il blocco a cui non si accede da più tempo (nota: per acesso ad un blocco si intende una qualunque `readhd_n()` o `writehd_n()` che lo ha coinvolto).

Per realizzare la buffer cache definiamo la seguente struttura dati nel modulo I/O:

```
struct buf_des {
    natl block;
    bool full;
    int next, prev;
    natb buf[DIM_BLOCK];
};
```

La struttura rappresenta un singolo elemento della buffer cache. I campi sono significativi solo se `full` è `true`. In quel caso `buf` contiene una copia del blocco `block`. I campi `next` e `prev` servono a realizzare la coda LRU come una lista doppia (si veda più avanti). Aggiungiamo poi i seguenti campi alla struttura `des_ata` (che è il descrittore dell'hard disk):

```
buf_des bufcache[MAX_BUF_DES];
int lru, mru;
```

Il campo `bufcache` è la buffer cache vera e propria; il campo `lru` è l'indice in `bufcache` del prossimo buffer da rimpiazzare (testa della coda LRU) e il campo `mru` è l'indice del buffer acceduto più di recente (ultimo elemento della coda LRU). I campi `next` e `prev` in ogni elemento di `bufcache` sono gli indici del prossimo e del precedente buffer nella coda LRU.

Modificare il file `io.cpp` in modo da realizzare il meccanismo descritto.