Prova pratica di Calcolatori Elettronici

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

15 February 2023

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

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

   Realizzare in Assembler GCC le funzioni membro seguenti.

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

2. Vogliamo aggiungere al nucleo delle primitive che ci permettano di leggere e scrivere dall'hard disk come se questo fosse un unico array di byte, invece che di blocchi. In particolare, il blocco 0 conterrà i byte

nell'intervallo $[0, 512)$, il blocco 1 i byte nell'intervallo $[512, 1024)$, e così via. Supponiamo che l'utente chieda di leggere n byte a partire dal byte b. Le primitive dovranno internamente accedere ai blocchi che contengono tutto l'intervallo $[b, b + n)$ e poi restituire o modificare solo i byte richiesti dall'utente.

Per rendere efficienti queste operazioni le nuove primitive usano una buffer cache che mantiene in memoria i blocchi acceduti più di recente, in modo che eventuali letture di byte contenuti in blocchi che si trovino nella buffer cache possano essere realizzate con una semplice copia da memoria a memoria, senza ulteriori operazioni di I/O. Per quanto riguarda le scritture adottiamo una politica write-back/write-allocate. Per il rimpiazzamento adottiamo la politica LRU (Least Recently Used): se la cache è piena e deve essere caricato un blocco non in cache, si rimpiazza il blocco a cui non si accede da più tempo.

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];
    bool dirty;
};
```

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). Il campo `dirty` è `true` se (e solo se) il blocco è stato modificato e il suo contenuto non è stato ancora ricopiato sull'hard disk. 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.

Aggiungiamo infine le seguenti primitive che realizzano il meccanismo descritto:

- `void bufread_n(natb* vetti, natl first, natl nbytes)` (tipo 0x48, da realizzare): legge dall'hard disk i byte nell'intervallo $[$`first`$,$ `first` $+$ `nbytes`$)$ e li copia in `vetti`, usando la buffer cache per minimizzare il numero di operazioni di I/O;
- `void bufwrite_n(natb* vetto, natl first, natl nbytes)`(tipo 0x49, da realizzare): scrive i byte contenuti in `vetto` nell'intervallo $[$`first`$,$ `first` $+$ `nbytes`$)$ di byte dell'hard disk, usando la buffer cache per minimizzare il numero di operazioni di I/O.

Nota: in nessun caso la cache deve eseguire operazioni di lettura/scrittura dall'hard disk che non siano strettamente necessarie.

Controllare eventuali problemi di Cavallo di Troia e abortire il processo se necessario.

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