2023-02-01

cache hard disk

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 write-back/write-allocate. 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];

    bool dirty; // Non dato dal prof
};
```

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):

```
struct des_ata {
    ...
    /// buffer cache
    buf_des bufcache[MAX_BUF_DES];
    int lru,    ///< indice del prossimo buffer da rimpiazzare
        mru;    ///< indice del buffer acceduto più di recente
    natl rd_count;  ///< non usare: solo per debug e test
    natl wr_count;  ///< non usare: solo per debug e test
};

void bufcache_promote(buf_des *b)
{
    des_ata *d = &hard_disk;
    int i = b - d->bufcache;
```

```
    if (b->next >= 0 || b->prev >= 0) {
        // d era già in lista: estraiamolo
        if (b->next >= 0)
            d->bufcache[b->next].prev = b->prev;
        else
            d->mru = b->prev;
        if (b->prev >= 0)
            d->bufcache[b->prev].next = b->next;
        else
            d->lru = b->next;
    }
    if (d->mru >= 0)
        d->bufcache[d->mru].next = i;
    else
        d->lru = i;
    b->prev = d->mru;
    b->next = -1;
    d->mru = i;
}
buf_des* bufcache_search(natl block)
{
    des_ata *d = &hard_disk;

    for (int i = 0; i < MAX_BUF_DES; i++) {
        buf_des *b = &d->bufcache[i];
        if (b->full && b->block == block)
            return b;
    }
    return nullptr;
}
```

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 la primitiva `void synchd()` (tipo 0x48, senza argomenti), che "sincronizza" l'hard disk con la cache, cioè aggiorna il contenuto di tutti i blocchi che ne hanno bisogno.

```
void estern_hd(natq)
{
    des_ata* d = &hard_disk;
    for(;;) {
        d->cont--;
```

```
        hd::ack();
        switch (d->comando) {
        case hd::READ_SECT:
            d->rd_count++;
            hd::input_sect(d->punt);
            d->punt += DIM_BLOCK;
            break;
        case hd::WRITE_SECT:
            d->wr_count++;
            if (d->cont != 0) {
                hd::output_sect(d->punt);
                d->punt += DIM_BLOCK;
            }
            break;
        case hd::READ_DMA:
        case hd::WRITE_DMA:
            bm::ack();
            break;
        }
        if (d->cont == 0)
            sem_signal(d->sincr);
        wfi();
    }
}
```

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

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

```
extern "C" void c_readhd_n(natb vetti[], natl primo, natb quanti)
{
    des_ata* d = &hard_disk;
    ...
    sem_wait(d->mutex);
    for (natl i = 0; i < quanti; i++) {
        // cerchiamo il blocco nella buffercache. Se non lo
        // troviamo rimpiazziamo l'lru
        buf_des *b = bufcache_search(primo + i);
        if (!b) {
            b = &d->bufcache[d->lru];
            if (b->dirty) {
                starthd_out(d, b->buf, b->block, 1);
                sem_wait(d->sincr);
            }
            starthd_in(d, b->buf, primo + i, 1);
            sem_wait(d->sincr);
```

```
            b->block = primo + i;
            b->full = true;
            b->dirty = false;
        }
        memcpy(vetti + i * DIM_BLOCK, b->buf, DIM_BLOCK);
        // ora b è l'mru
        bufcache_promote(b);
    }
    sem_signal(d->mutex);
}
extern "C" void c_writehd_n(natb vetto[], natl primo, natb quanti)
{
    des_ata* d = &hard_disk;
    ...
    sem_wait(d->mutex);
    for (natl i = 0; i < quanti; i++) {
        // politica write-back: scriviamo solo in buffercache
        // non c'è bisogno di caricare il blocco dall'hard disk,
        // in quanto dobbiamo sovrascriverlo interamente.
        buf_des *b = bufcache_search(primo + i);
        if (!b) {
            b = &d->bufcache[d->lru];
            if (b->dirty) {
                starthd_out(d, b->buf, b->block, 1);
                sem_wait(d->sincr);
            }
            b->block = primo + i;
            b->full = true;
        }
        memcpy(b->buf, vetto + i * DIM_BLOCK, DIM_BLOCK);
        b->dirty = true;
        // c'è stato un accesso al buffer e dobbiamo promuoverlo
        bufcache_promote(b);
    }
    sem_signal(d->mutex);
}
extern "C" void c_synchd()
{
    des_ata *d = &hard_disk;

    sem_wait(d->mutex);
    for (int i = 0; i < MAX_BUF_DES; i++) {
        buf_des *b = &d->bufcache[i];
        if (b->dirty) {
            starthd_out(d, b->buf, b->block, 1);
            sem_wait(d->sincr);
```

```
            b->dirty = false;
        }
    }
    sem_signal(d->mutex);
}

bool hd_init()
{
    des_ata* d = &hard_disk;
    // conviene inserire anche i buffer vuoti nella coda LRU (in un ordine
    // qualsiasi), in modo da non dover considerare a parte il caso di buffer
    // vuoto.
    d->lru = d->mru = -1;
    for (int i = 0; i < MAX_BUF_DES; i++) {
        buf_des *b = &d->bufcache[i];
        b->full = false;
        b->next = b->prev = -1;
        bufcache_promote(b);
        b->dirty = false;
    }
    return true;
}
```

```
fill_io_gates:
    ...
    fill_io_gate    IO_TIPO_SHD a_synchd
    ...

.extern c_synchd
a_synchd:
    .cfi_startproc
    .cfi_def_cfa_offset 40
    .cfi_offset rip, -40
    .cfi_offset rsp, -16
    call c_synchd
    iretq
    .cfi_endproc
```