1 Lezione del 29-04-25

Riprendiamo il discorso del DMA nella prospettiva di un esempio concreto.

1.1 Hard disk e DMA

Fra i dispositivi visti finora solo l'hard disk è quello capace di fare DMA nel kernel. Dentro la macchina virtuale QEMU è disponibile un'emulazione dell'hard disk del PC AT (l'HD ATA visto in 4.1). Questo non era capace di fare DMA in autonomia, ma era bensì collegato ad un controllore DMA.

Fra i comandi disponibili per comunicare con l'hard disk ci sono quindi comandi dedicati a letture e scritture in DMA. Quando tali comandi vengono inviati all'hard disk, questo si occupa di coinvolgere il controllore DMA.

Questa non è più la situazione odierna: l'hard disk ATA con cui comunica la macchina emulata è situato sul bus ATA, che si collega al bus PCI con un ponte PCI-ATA. E' quindi il ponte a comportarsi come il controllore DMA, lato bus ATA.

Considerazioni storiche a parte, vediamo la struttura del controllore DMA dell'hard disk ATA, come descritto nella specifica reperibile a `https://calcolatori.iet.unipi.it/deep/idems100.pdf`. Abbiamo che questo può gestire due dischi separati, denominati *primario* e *secondario*, con relativi registri:

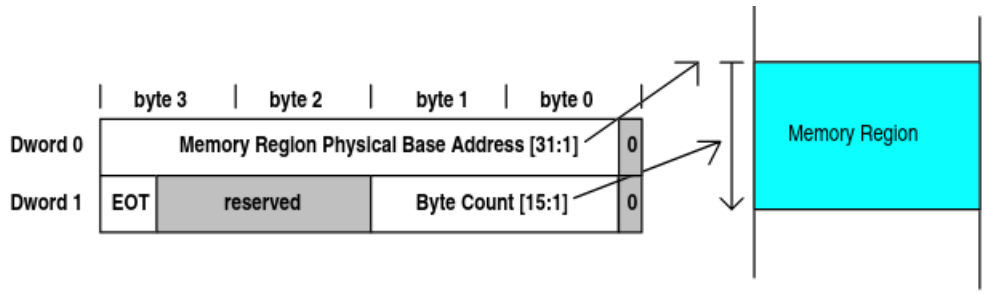
	Primario
`0x?? + 0`	**BMCMD**, *Bus Master Command*
`0x?? + 1`	Specifico al dispositivo
`0x?? + 2`	**BMSTR**, *Bus Master Status Register*
`0x?? + 3`	Specifico al dispositivo
`0x?? + 4-7`	**BMDTPR**, *Bus Master Descriptor Table Pointer*
	Secondario
`0x?? + 8`	**BMCMD**, *Bus Master Command*
`0x?? + 9`	Specifico al dispositivo
`0x?? + a`	**BMSTR**, *Bus Master Status Register*
`0x?? + b`	Specifico al dispositivo
`0x?? + c-f`	**BMDTPR**, *Bus Master Descriptor Table Pointer*

Riguardo a ogni registro avremo:

- **BMCMD**, *Bus Master Command*: questo specifica il tipo di operazione che vogliamo eseguire (lettura o scrittura), e ne specifica l'inizio. Per lanciare un'operazione, infatti, il software dovrà impostare il bit di *Read or Write Control* (bit 3), e successivamente alzare il bit *Start/Stop Bus Master* (bit 0);
- **BMSTR**, *Bus Master Status Register*: indica lo stato corrente del dispositivo a cui corrisponde. In particolare ci sono di interesse i primi 3 bit meno significativi (gli altri danno principalmente informazioni rispetto alle funzioni supportate dai dispositivi). Questi saranno:
  - Bit 2: rappresenta l'**interruzione**, viene alzato quando la trasmissione di dati in DMA è stata completata;
  - Bit 1: rappresenta uno stato di **errore**;
  - Bit 0: indica se il bus mastering è attivo o meno, cioè viene alzato quando il software scrive 1 sul bit start/stop bus master del BMCMD.

Abbiamo poi che i bit 1 e 2 possono essere resettati scrivendovi 1 (ed è questo passo che termina l'handshake col controllore DMA).

- **BMDTPR**, *Bus Master Descriptor Table Pointer*: questo punta alla prima entrata della cosiddetta tabella **PRD**, *Physical Region Descriptor Table*. Questa è una tabella di entrate da 8 byte, allineate ai 4 byte, che indicano l'indirizzo base della regione da trasferire, il numero di byte da trasferire e se l'entrata corrente è l'ultima della tabella (il controllore DMA continua a scorrere le entrate finchè non raggiunge l'ultima). La struttura delle entrate PRD è la seguente:



Notiamo che le regioni indicate dall'indirizzo base dell'entrata PRD può essere al massimo di 64 KiB. Per questo lato hardware si può usare un sommatore a sole 16 cifre. In ogni caso, questo non sarà un problema in quanto vorremo trasferire buffer in memoria virtuale una pagina (4 KiB) alla volta.

A questo punto basterà definire i passaggi di un operazione di trasferimento:

1. Si prepara una tabella PRD in memoria;
2. Si carica l'indirizzo base della tabella PRD nel registro BMDTPR, quindi si ripuliscono i bit di interruzione ed errore del registro di stato BMSTR;
3. Si fornisce il comando appropiato sul registro BMCMD;
4. Si attiva il bit 0 del registro BMCMD per attivare il bus mastering;
5. Il controllore DMA trasferisce i dati secondo quanto disposto finora;
6. Alla fine della trasmissione il controllore segnala la fine dell'operazione su una linea di interruzinoe;
7. In risposta all'interruzione, si resetta il bit 0 del registro BMCMD, e si legge lo stato dal controllore e dal disco per capire se l'operazione è andata a buon fine.

1.1.1 Controller IDE su bus PCI

Per l'inserzione di un controllore di questo tipo in un bus PCI dobbiamo renderci conto di alcuni dettagli: Nei registri dello spazio di configurazione del dispositivo si devono attivare dei flag particolari per segnalare la possibilità che questo lavori in bus mastering.

1.1.2 Controller IDE nel kernel

Vediamo infine come il controllore DMA dell'hard disk ATA viene gestito nel kernel. La libreria `libce` definisce i registri del controllore:

```
namespace bm {
  extern ioaddr iBMCMD; // Bus Master Command
  extern ioaddr iBMSTR; // Bus Master Status Register
  extern ioaddr iBMDTPR; // Bus Master Descriptor Table Pointer
}
```

e le relative funzioni per l'inizializzazione, l'acknowledge, ecc...

L'unica interfaccia ATA montata nel sistema è quindi descritta dal descrittore:

```
// descrittore di interfaccia ATA
struct des_ata {
  // Ultimo comando inviato all'interfaccia
  natb comando;
  // Indice di un semaforo di mutua esclusione
  natl mutex;
  // Indice di un semaforo di sincronizzazione
  natl sincr;
  // Quanti settori resta da leggere o scrivere
  natb cont;
  // Da dove leggere/dove scrivere il prossimo settore
  natb* punt;
  // Array dei descrittori per il Bus Mastering
  natl* prd;
};
```

che tiene conto dell'operazione corrente.

A questo punto il processo esterno dedicato all'hard disk dovrà limitarsi ad inviare i comandi corretti seguendo la scaletta appena riportata. Unica parte di interesse è quella della preparazione della tabella PRD, per cui bisogna tenere conto che il controllore DMA necessita di indirizzi fisici, e che legge sequenzialmente a partire da tali indirizzi fisici (perciò non si possono superare i 4 KiB della dimensione di pagina). Per fare questo, e tenere conto di buffer in memoria che iniziano potenzialmente a metà pagina, si sfrutta la funzione `prepare_prd()`:

```
bool prepare_prd(des_ata *d, natb* vett, natb quanti)
{
  natq n = quanti * DIM_BLOCK;
  int i = 0;

  while (n && i < MAX_PRD) {
    paddr p = trasforma(vett);
    natq  r = DIM_PAGINA - (p % DIM_PAGINA);
    if (r > n)
      r = n;
    d->prd[i] = p;
    d->prd[i + 1] = r;

    n -= r;
    vett += r;
    i += 2;
  }
  if (n)
    return false;
  // il bit end of table
  d->prd[i - 1] |= 0x80000000;
  return true;
```

```
}
```

A questo punto si possono fornire all'utente primitive per l'accesso all'hard disk sia a controllo interruzione (come avevamo già visto, implementato in `libce`) sia in DMA. Queste saranno:

- **Controllo interruzione:** vediamo ad esempio l'operazione di ingresso.

```
// fondamentalmente un wrapper per hd::start_cmd di libce, che
    aggiorna il descrittore
void starthd_in(des_ata* d, natb vetti[], natl primo, natb quanti)
{
  d->cont = quanti;
  d->punt = vetti;
  d->comando = hd::READ_SECT;
  hd::start_cmd(primo, quanti, hd::READ_SECT);
}

// la primitiva vera e propria
extern "C" void c_readhd_n(natb vetti[], natl primo, natb quanti)
{
  des_ata* d = &hard_disk;

  // controlli (c_access)

  sem_wait(d->mutex);
  starthd_in(d, vetti, primo, quanti);
  sem_wait(d->sincr);
  sem_signal(d->mutex);
}
```

- **DMA:** vediamo sempre l'operazione di ingresso:

```
void dmastarthd_in(des_ata* d, natb vetti[], natl primo, natb quanti)
{
  // passo 1 della scaletta
  if (!prepare_prd(d, vetti, quanti)) {
    flog(LOG_ERR, "dmastarthd_in: numero di PRD insufficiente");
    sem_signal(d->sincr);
    return;
  }

  d->comando = hd::READ_DMA;
  d->cont = 1;

  // passo 2
  paddr prd = trasforma(d->prd);
  bm::prepare(prd, false);

  // passo 3
  hd::start_cmd(primo, quanti, hd::READ_DMA);
  bm::start();
}
```

A operazioni terminate, il processo esterno dovrà chiaramente interpretare correttamente le interruzoni che riceve in base al tipo di comando dato:

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

```
    d->cont--;
    hd::ack();
    switch (d->comando) {
    // questi sono i casi gia visti
    case hd::READ_SECT:
      hd::input_sect(d->punt);
      d->punt += DIM_BLOCK;
      break;
    case hd::WRITE_SECT:
      if (d->cont != 0) {
        hd::output_sect(d->punt);
        d->punt += DIM_BLOCK;
      }
      break;
    case hd::READ_DMA:
    case hd::WRITE_DMA:
      // qui si fa l'acknowledge, passo 7 della scaletta
      bm::ack();
      break;
    }
    if (d->cont == 0)
      sem_signal(d->sincr);
    wfi();
  }
}
```