1 Lezione del 08-04-25

Concluso il discorso sulla memoria virtuale, ci concentreremo nuovamente sull'hardware, nella prospettiva di approfondire l'interazione fra nucleo e dispositivi di I/O.

1.1 Bus PCI

Il bus **PCI**, che sta per *Peripheral Component Interconnect*, è uno standard per bus sviluppato da IBM per consentire l'espansione dei loro calcolatori attraverso schede apposite, supportando quindi una cosiddetta *architettura aperta*.

Possiamo quindi immaginare che ogni scheda di espansione sia provvista dei suoi registri, delle sue interruzioni, ecc... che non devono sovrapporsi con quelli di altre schede. Storicamente, questo rappresentava un problema, in quanto potevano crearsi *conflitti* fra più schede.

Inoltre, un problema era rappresentato dai *driver*, in quanto non esisteva un modo standardizzato per rilevare se una certa scheda era installata o no, e quindi se si poteva usare un certo driver.

I produttori stabilirono quindi una sorta di standard *de facto*, che abbiamo già nominato: l'**ISA** (*Industry Standard Architecture*).

L'idea fondamentale è che la scheda non può avere registri fissi, ma deve essere programmabile in questo dal calcolatore. Inoltre, deve esistere una qualche modalità per rilevare le schede correntemente installate nel sistema.

Nei PC moderni sfruttiamo uno standard di derivazione dal vecchio PCI, compatibile con esso, che è il *PCI Express* (e che non studieremo).

1.1.1 Indirizzamento dei dispositivi

Secondo lo standard PCI, separiamo il **bus locale** (quello che abbiamo visto finora) da un eventuale **albero di bus**, collegati fra di loro dai cosiddetti **ponti**. Il **ponte ospite-PCI**, in particolare, collega il **bus principale** (il più vicino al bus locale) al bus locale, mentre questo a sua volta viene collegato ad altri bus attraverso **ponti PCI-PCI**. Ad esempio, molti calcolatori dell'epoca erano dotati di *bus ISA* collegati con appositi ponti al bus principale, per la gestione di vecchie interfacce ISA.

A ogni bus è associato un numero su 8 bit, col bus principale che si prende il numero 0.

Per indirizzare un dispositivo usiamo invece 16 bit, disposti come:

Scopo	**Bit**
Bus	8 bit
Device	5 bit
Function	3 bit

dove il numero di bus è lo stesso di prima. Il numero di funzione è reso necessario da schede che implementano più funzionalità, quali ad esempio le schede grafiche moderne, che si occupano anche dell'audio. In ogni caso, la funzione 0 deve essere implementata obbligatoriamente.

1.1.2 Operazioni coi dispositivi

Veniamo quindi a come funzionano le operazioni più semplici sul Bus PCI. Ogni operazione sul PCI viene detta **transazione**, ed ha un **iniziatore** e un **obiettivo**, cioè il dispositivo che inizia la transazione e il dispositivo che gli risponde. L'iniziatore, notiamo, non sarà più il processore, ma il ponte ospite-PCI.

Sul bus vero e proprio troviamo quindi:

- La linea di **clock**, che tutte le interfacce vedono, originariamente intorno ai 33 MHz. Sul fronte di salita del clock tutte le interfacce (idealmente) campionano i segnali sul bus;
- **FRAME#**, che "incornicia" la transazione corrente;
- **AD**, la linea *condivisa* di indirizzo o dati, a 32 bit;
- **C/BE#**, *controllo* e *byte-enable*, codificano il tipo di operazione in fase di indirizzamento e fanno da byte-enable nel trasferimento dati;
- **IRDY#** e **TRDY#**, supportano l'handshake nella fase di scambio dati;
- **DEVSEL#**, viene attivato dal dispositivo che riconosce, controllando **C** e l'indirizzo su **AD**, una chiamata a sé stesso, quindi dal presunto *obiettivo*;
- **STOP#**, viene attivato dall'*obiettivo* per terminare prematuramente una transazione.

screen transazione

1.1.3 Configurazione di dispositivi

Per poter lavorare con il bus PCI, poi, introduciamo un nuovo spazio indirizzabile, quello di **configurazione**. Avremo quindi che il processore può indirizzare:

- **Memoria**;
- **I/O**;
- **Configurazione**.

Questo spazio è rilevante solo all'avvio del calcolatore (all'esecuzione del BIOS), appunto per effettuare la configurazione delle interfacce PCI installate nel sistema.

Useremo lo spazio di I/O come sempre, ma i segnali del processore viaggeranno attraverso i vari ponti fino all'interfaccia desiderata. Per quanto riguarda i dispositivi mappati in memoria (si pensi al video), invece, ossiamo immaginare che all'avvio il ponte ospite-PCI deve solo sapere la dimensione della memoria RAM installata, in modo da rispondere da lì in poi solo agli indirizzi *di memoria* posti al di *sotto* di essa.

Tutto è comunque strutturato per essere trasparente al processore, e quindi invisibile lato software.

Ogni dispositivo sul bus è obbligato a fornire, per ogni funzione e ad una certa locazione predefinita, un numero di registri a righe da 32 bit, che devono contenere informazioni di configurazione. I primi due dati, su 16 bit (qunidi una riga), saranno il **Vendor ID** e il **Device ID**.

c'è una tabella completa nella dispensa

Device ID	Vendor ID	0
//	//	4
/	//	8
...	...	...

Il Vendor ID è determinato da un'autorità centrale (la *PCI-SIG*): ad eesempio, il vendor ID della Intel è `0x8086`.

Per permettere quindi alla CPU di configurare i dispositivi, cioè accedere ai loro registri di configurazione, il ponte ospite-PCI rende disponibili alla CPU due registri, entrambi su 32 bit:

- Il **CAP**, *Configuration Address Port*, che permette di selezionare una funzione e l'offset della parola a cui si vuole accedere;
- Il **CDP**, *Configuration Data Port*, che permette di accedere alla parola selezionata con *CAP*.

Le operazioni effettuate dalla CPU attraverso questi due registri verranno trasformate automaticamente dal ponte ospite-PCI in operazioni di configurazione sui bus PCI.

Vediamo quindi cosa deve fare il BIOS per la configurazione dei dispositivi PCI, cioè per collocarne nello spazio di I/O o in memoria eventuali registri o porzioni di memoria, rispettivamente. Ogni dispositivo ha una **dimensione naturale** che occupa nello spazio, sia questo di memoria o di I/O. Fornisce quindi al processore un registro, detto **BAR** (*Base Address Register*), che è scrivibile solo in parte: la parte meno significativa, infatti, è fissa a 0 e determina la dimensione naturale della regione che questo occuperà. Per rilevare la dimensione naturale, quindi, basta scrivere tutti 1 sul BAR e controllare quai bit vengono effettivamente modificati.

Il **PCI BIOS** dovrà quindi controllare il BAR di tutte le interfacce, determinarne la dimensione naturale e trovare una regione libera nello spazio di I/O o in memoria, a seconda del tipo di dispositivo, dove collocarle.

1.1.4 Interruzioni PCI

Per la gestione delle interruzioni, lo standard si ferma al dire che ogni dispositivo deve specificare per ogni funzione quale, di quattro linee di interruzione, tale funzione usa, in un apposito registro di configurazione. Queste linee sono dette **INTA**, **INTB**, **INTC** e **INTD**, e lo spazio di configurazione della funzione contiene il dato rispetto a quale linea usa in *Intr. Pin*, con 0 che significa nessuna interruzione, 1 *INTA*, e così via.

capisci sopra e ruzza con QEMU (dispositivi + fase di bootloader e caricamento nucleo)

1.2 I/O nel kernel

Vediamo quindi com'è implementata la gestione dell'I/O da parte del kernel. Avevamo detto che la motivazione principale dietro lo sviluppo del sistema multiprogrammato era che questa permetteva l'interruzione in qualsiasi momento di un *processo* in esecuzione, per permettere al processore di effettuare altre operazioni. Le interruzioni sollevate da i dispositivi esterni, vediamo, possono essere gli eventi che provocano tale cambio di contesto, e quindi la gestione del segnale in ingresso al sistema.

Ad esempio, un processo potrebbe, con la funzione `readconsole()`, specificare un buffer e un numero di caratteri che vuole leggere da tastiera. A questo punto, il sistema

lo metterà in attesa e si occuperà di altro, riempiendo sequenzialmente il buffer via via che i tasti vengono effettivamente premuti (e quindi le relative interruzioni sollevate). Una volta che il buffer sia riempito dal numero di caratteri richiesti dal processo, potrà quindi rimettere il processo in esecuzione, o nella lista pronti, e proseguire.

Per tenere conto di più processi che possono voler leggere contemporaneamente, poi, ci dotiamo di un semaforo che tenga conto di chi sta usando quella risorsa in un certo momento.