Azzolini Riccardo 2019-09-26

Interrupt

1 Salto dal programma all'interrupt handler

Nella kernel area della RAM è presente, per ogni classe di interrupt, una struttura dati chiamata **interrupt vector**, in cui sono memorizzati i dati che devono essere assegnati ai registri di controllo della CPU per eseguire il salto da un programma qualsiasi all'interrupt handler. In particolare, l'interrupt vector contiene:

- l'indirizzo della prima istruzione dell'handler, da assegnare al PC per effettuare il salto vero e proprio;
- il valore da assegnare alla PSW per:
  - mettere la CPU in modalità kernel (bit PM, Privileged Mode);
  - disabilitare gli interrupt di priorità non superiore (IM, Interrupt Mask).

Gli interrupt vector vengono inizializzati in memoria durante la fase di booting, e, siccome si trovano nella kernel area, non sono accessibili ai programmi user.

Quando arriva un interrupt abilitato:

1. i valori dei registri di controllo (PC, SP e PSW) vengono salvati nell'apposita **Saved Register Information Area** (**SRIA**), situata nella user area della RAM, in modo da poter essere ripristinati quando il programma dovrà ripartire;
2. i registri PC e PSW vengono impostati in base ai valori memorizzati nell'interrupt vector corrispondente alla classe dell'interrupt, avviando così l'esecuzione dell'interrupt handler.

2 Esecuzione dell'interrupt handler

Una volta avviato, l'interrupt handler:

1. salva i registri generali della CPU;
2. esegue il codice di gestione dell'interrupt;
3. salta al codice dello scheduler, che sceglie un programma da eseguire.

3 Salto dallo scheduler al programma

Dopo aver selezionato il programma la cui esecuzione deve riprendere, lo scheduler:

1. ripristina i valori dei registri generali che erano stati salvati quando il programma è stato interrotto;
2. esegue l'istruzione **Interrupt Return** (**IRET**), che ripristina i valori dei registri di controllo, compreso il PC, facendo quindi riprendere l'esecuzione del programma dal punto in cui essa era stata interrotta.

Osservazione: La SRIA contiene i valori dei registri per tutti i programmi interrotti, quindi lo scheduler può scegliere liberamente quale di essi riprendere.

4 Interrupt Code

A volte, il dispositivo che richiede un interrupt vuole passare al SO anche delle informazioni aggiuntive. Ad esempio, se si verifica un disk interrupt in un sistema con più hard disk, è necessario specificare da quale dei dischi proviene l'interrupt: in genere, la classe dell'interrupt non è sufficiente a identificare il singolo dispositivo, perché non ci sono abbastanza classi da permettere a ogni dispositivo di usarne una diversa.

Quest'informazione aggiuntiva viene memorizzata nei bit **Interrupt Code** (**IC**) della PSW. Essi sono poi letti dall'interrupt handler per determinare cosa deve fare.

5 Considerazioni

- I SO assumono che l'hardware supporti gli interrupt: questo è un esempio di evoluzione dell'hardware dettata dallo sviluppo dei SO.
- Il salvataggio del PC e l'impostazione del suo valore dall'interrupt vector devono per forza essere eseguiti dall'hardware (mediante porte logiche attivate dal segnale di richiesta dell'interrupt): questo è l'unico modo di interrompere la normale esecuzione del programma. Invece, tutti gli altri registri potrebbero essere salvati via software, all'inizio dell'interrupt handler.

  Il salvataggio via software è più lento, ma richiede hardware meno complesso. Come compromesso, è spesso possibile salvare tutti i registri di controllo via hardware, mentre quelli generali devono essere salvati dal software.
- I registri della CPU devono essere salvati anche quando il codice interrotto è esso stesso un interrupt handler. Siccome gli interrupt handler che possono essere avviati a cascata sono limitati dalle classi di priorità, è sufficiente che ciascun programma abbia una SRIA per ogni classe.

6 Interrupt su CPU più avanzate

Le CPU moderne non seguono il tradizionale ciclo fetch-decode-execute. Alcuni esempi di CPU più avanzate sono:

- organizzazione *pipeline*: si effettuano contemporaneamente l'esecuzione dell'istruzione i, la decodifica dell'istruzione $i + 1$ e il fetch dell'istruzione $i + 2$;
- CPU *superscalare*: dotata di più unità di fetch, decodifica ed esecuzione che lavorano contemporaneamente.

Rispetto a quelle basate sul ciclo tradizionale, queste tipologie di CPU hanno prestazioni migliori, ma introducono ulteriori complessità nella gestione degli interrupt. Infatti, quando un programma viene interrotto:

- possono esserci istruzioni già prelevate, ma non ancora eseguite;
- nelle CPU superscalari, possono esserci istruzioni eseguite solo parzialmente.

I principi di base del meccanismo degli interrupt rimangono comunque gli stessi, ma devono essere implementati in modi più complicati.