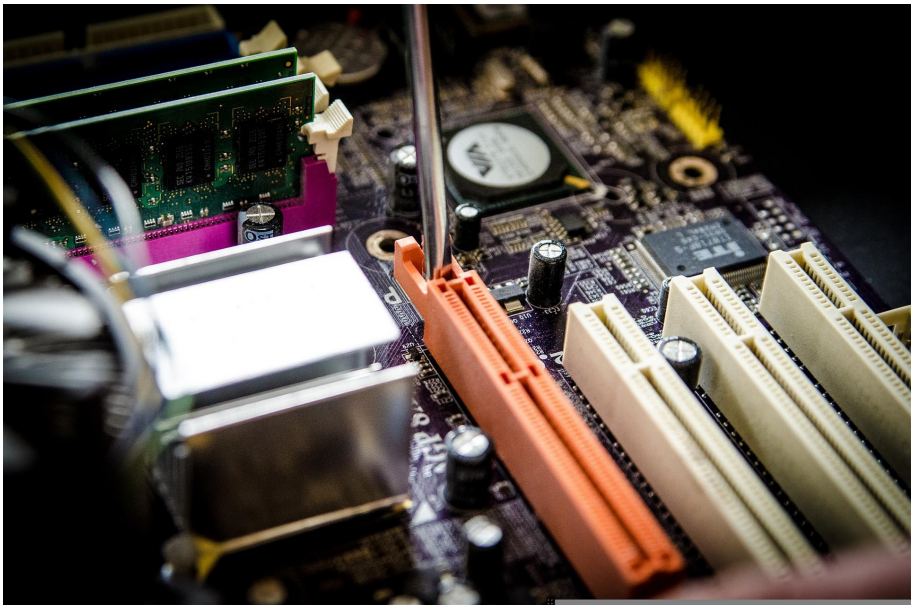
Dal Sistema Monoprogrammato al Sistema Multiprogrammato

Dal corso di Calcolatori Elettronici dell'Universita' di Pisa

Matteo Bernini
https://github.com/mattebernini/
https://t.me/mattebernini



21 settembre 2022

Indice

L'idea di questa dispensa quella di dare una spiegazione ad alto livello di come si passa da un sistema monoprogrammato ad uno multiprogrammato, per far ci necessaria una minima introduzione a come funzioni un sistema monoprogrammato semplificando al massimo tale spiegazione.

Disclaimer

Questa dispensa frutto della preparazione dell'esame di Calcolatori Elettronici[1] dell'Universit di Pisa, la sua finalit non quella di sostituire le dispense e le spiegazioni del professor Lettieri bens quella di avere una visione ad alto livello di ci che viene spiegato durante tutto il corso, come al solito le cose sono molto pi complesse e l'intento quello di di dare soltanto un'idea di come funzioni un sistema multiprogrammato senza scendere nel dettaglio. Posso preparare l'esame con questa dispensa? No, al massimo pu essere usata come ripasso per fissare i concetti e dare una visione d'insieme del corso.

1 Introduzione

Partiamo con la definizione di hardware: tutte quelle parti elettroniche, elettriche, meccaniche, magnetiche, ottiche che consentono il funzionamento di un computer, in questo caso supersemplificato avremo il processore (o CPU), la memoria ram e le periferiche (tastiera, schermo, ecc...). Per spiegare pi semplicemente possibile a costo di sembrare banale il funzioanmento di un computer (e quindi riassumendo il corso di reti logiche[2] in una riga) in sostanza la CPU preleva dalla ram, decodifica ed esegue sequenzialmente una istruzione alla volta, tali istruzioni sono scritte anch'esse da un'altra parte della ram ed alcune di esse fanno uso delle periferiche tramite le interfacce, esse sono molto pi lente della CPU visto che devono sottostare ai tempi dell'utente i quali sono diversi da quelli della CPU (esegue milioni di istruzioni al secondo, noi no). L'hardware alla base del funzionamento di un computer ma ci che lo rende funzionante il software, questo non altro che il programma che sta scritto in memoria[3], pertanto il programmare un computer non altro che scrivere dei bit in memoria con una determinata logica. Un sistema monoprogrammato un sistema hardware e software che prevede l'esecuzione di un unico programma per volta. Facciamo un esempio: voglio mostrare in output il risultato di una funzione matematica di n numeri, per far ci scriver il mio programma che per n volte si calcola il risultato e poi lo manda in output. Nella suddetta architettura la CPU quindi svolger delle istruzioni per calcolare il risultato, a quel punto "invier" il risultato alla periferica addetta all'output che ci metter tantissimo tempo (in relazione a quello che ci mette la CPU a fare il calcolo) a mostrare il risultato e a rendersi disponibile alla CPU per un nuovo output, in tutto questo tempo la CPU dovr aspettare la

[1] https://calcolatori.iet.unipi.it/

[2] $http://docenti.ing.unipi.it/\ a080368/Teaching/RetiLogiche/index_RL.html$

[3] Come si distingue se una procedura fatta in hardware o in software? Se so scrivere un programma per farlo in software, altrimenti in hardware.

periferica senza far niente di utile. Questo sistema non molto efficiente... Ci servirebbe un modo per non sprecare tutto quel tempo.

2 Il meccanismo delle interruzioni

Per risolvere il suddetto problema dovremo fare un piccolo cambiamento hardware, ovvero delegare a quest'ultimo avvisare la CPU quando la periferica pronta per non farle perdere tempo a controllare, interrompendo i suoi calcoli, lo stato della periferica di output. Per far ci basta semplicemente collegare la CPU con la periferica, ogni volta che la periferica pronta mandare in output un nuovo risultato questa avvisa la CPU la quale svolge automaticamente un programma di routine (detta semplicemente routine) situata ad un indirizzo specifico della memoria ram e poi torna al programma principale. Ora il programmatore dovr scrivere due programmi differenti: uno principale che calcola i risultati e li mette in un posto specifico della memoria e una routine che viene eseguita come detto prima, la quale si occuper semplicemente di mandare in output i risultati salvati nella memoria tramite la periferica. In questo modo la CPU non spreca pi tempo ad aspettare che la periferica sia pronta e viene utilizzata soltanto per il calcolo dei risultati. Cercando di ottimizzare un problema di spreco di tempo della CPU in realt abbiamo gettato le basi per un sistema multiprogrammato, infatti come detto sopra adesso abbiamo due programmi che si alternano in memoria: quello principale e la routine. Il programma principale far il suo dovere e verr interrotto (da qui il nome di meccanismo delle interruzioni) dalla routine, una volta eseguita quest'ultima la CPU torner ad eseguire il programma principale proprio dal punto in cui si era interrotta.

2.1 Interruzioni da pi sorgenti

Nella realt avremo bisogno che la nostra CPU sia in grado di rispondere a pi di una richiesta di interruzione e gestire tutto ci che ne deriva. Una delle possibili soluzioni (quella studiata nel corso di Calcolatori) di non collegare direttamente la CPU alla periferica ma di aggiungere un componente detto APIC (Advanced Programmable Interrupt Controller) il quale, sempre via hardware, si occuper di gestire le richieste di interruzione. Questo nuovo componente non far altro che ricevere le richieste, ordinarle per priorit[4] in base al tipo associato alla periferica ed inoltrare la richiesta alla CPU, la CPU a quel punto consulter una tabella in memoria ram detta IDT (interrupt descriptor table) che assocer ad ogni tipo l'indirizzo dell'inizio della routine corrispondente. Per sua definizione l'APIC programmabile (la P sta per quello) infatti il programmatore, oltre ad avere la possibilit di modificare la IDT, potr assegnare un tipo ad ogni periferica e scegliere il trigger mode[5] e la mascherabilit[6] visto che tali informazioni sono contenute in memoria ram.

[4] La gestione della priorit interamente gestita in hardware in questo caso.

[5] Richiesta al fronte o al livello, attiva alta o bassa.

[6] Disabilitare le interruzioni durante l'esecuzione della routine.

2.2 Il caso particolare delle Eccezioni

Le prime 32 entrate della IDT sono riservate alle eccezioni, queste non sono altro che condizioni di errore o speciali che il processore rileva mentre sta eseguendo le normali istruzioni. Le eccezioni sono gestite allo stesso modo delle interruzioni ma con piccole differenze, infatti in questo caso il tipo non modificabile dal programmatore ma standard e dipende dall'architettura (nel nostro caso ad esempio la divisione per zero al gate[7] 0), a questi tipi standard per il programmatore potr associare la routine che preferisce. Inoltre le eccezioni non sono gestite dall'apic ma direttamente dalla CPU, esse sono di tre tipi: **trap** sollevata tra due istruzioni, **fault** sollevata durante un'istruzione e **abort** sollevate in qualsiasi momento e particolarmente gravi.

3 Interruzione di Programma

Col meccanismo delle interruzioni abbiamo la possibilit di utilizzare la stessa tecnica non solo per le periferiche e le eccezioni ma anche per dare la possibilit al nostro computer di ospitare programmi diversi di utenti diversi. In questo modo ci buttiamo alle spalle la programmazione a controllo di programma e passiamo alla programmazione ad interruzione di programma. Per esempio: consideriamo l'esempio iniziale del calcolo della funzione ed estendiamolo a questo caso, diciamo che nello stesso momento vogliamo eseguire sullo stesso computer un programma che legge da tastiera un carattere e stampa il numero corrispondente nell'alfabeto. Abbiamo due programmi pi le due routine delle due periferiche (tastiera e periferica di output) in contemporanea nel nostro computer, diciamo che iniziamo col programma che legge (programma 1), questo ci metter un po' di tempo ad essere eseguito tutto, quanto tempo? Boh, dipende dall'utente, chi ci garantisce che scriva un carattere da tastiera? Nessuno. Quindi sarebbe bene che mentre il programma 1 aspetta l'utente potessimo eseguire il programma 2 (quello del calcolo della funzione).

3.1 Il meccanismo della Protezione

Chi ci garantisce che una volta che il programma 1 ha passato il controllo al programma 2 esso gli dar di nuovo il controllo? Chi ci garantisce che il programma 1 quando in esecuzione non modifichi la porzione di memoria del programma 2 e quindi anche le routine? Dalle domande sorte spontanee nelle righe sopra possiamo intuire che non possiamo aspettarci collaborazioni tra diversi utenti ne collaborazione tra programmi diversi (anche se scritti dallo stesso utente), quindi dobbiamo trovare un modo per regolamentare tutto ci. Questo modo si chiama meccanismo della protezione, questo meccanismo richiede una piccola modifica hardware ovvero: la CPU deve iniziare a distinguere chi richiede l'istruzione. Per chi richiede l'istruzione non si intende quale programma o utente la richiede ma quale livello di privilegio. Infatti da ora in poi distingueremo due differenti livelli di privilegio per eseguire le istruzioni: il livello utente che il livello a cui sono eseguiti i programmi degli utenti appunto e il livello sistema, quest'ultimo ha maggiori privilegi di quello utente infatti non soggetto a nessun tipo di restrizione mentre il livello utente non pu ne utilizzare direttamente le periferiche ne

[7] Le entrate della IDT sono chiamate gate (cancello).

scrivere in qualsiasi parte della memoria[8]. I programmi scritti dall'utente andranno quindi in esecuzione a livello utente, qualora abbiano bisogno di accedere alle periferiche chiameranno delle funzioni fornite dal livello sistema (detti driver) che non sono altro che le routine dell'APIC. Oltre ai driver i programmi utente possono usufruire di altri servizi forniti dal livello sistema dette primitive, esse sono chiamate durante il programma grazie all'istruzione *int tipo* che introduce quindi l'interruzione via software (fondamentale per esempio per passare da un programma utente ad un altro). Tutto il codice a livello sistema detto nucleo del sistema operativo (o Kernel), esso consente di avere un livello di astrazione tra il programmatore e l'hardware.

Conclusioni

L'aver differenziato due contesti diversi, quello utente e quello sistema, ha permesso di arrivare al sistema multiprogrammato, infatti con l'introduzione delle primitive il sistemista ha tutti gli strumenti per gestire pi programmi differenti allo stesso tempo[9], tramite l'uso delle primitive e dei driver il sistema pu schedulare al meglio i vari programmi in corso e quindi eseguirne molti in parallelo senza che gli utenti se ne rendano conto, inoltre vengono protetti i programmi tra loro e viene anche protetto il codice del Kernel. In questo sistema fondamentale comprendere chi fa cosa, l'hardware come abbiamo visto ha piccole mansioni di gestione (ad esempio l'APIC gestisce le richieste di interruzioni esterne[10]) ma il grosso del lavoro viene fatto via software dal Kernel, infatti il livello sistema, quello pi privilegiato, in realt ha il compito di mettersi al servizio dei vari programmi utente affinch tutto funzioni correttamente. Da notare per che sebbene l'hardware abbia poche mansioni di gestione proprio quest'ultimo con le modifiche apportate[11] che ci ha permesso di scrivere il software in questo particolare modo.

[8] Per ora diciamo che il livello utente ha una parte della memoria riservata a se e pu agire solo su quella.

[9] In realt manca la memoria virtuale.

[10] Le richieste di interruzione delle periferiche sono dette anche esterne.

[11] Modificando la CPU per abilitare la protezione e il meccanismo delle interruzioni e inserendo l'APIC tra le periferiche e la CPU.