Supporti al Time Sharing

lunedì 13 marzo 2023 18:48

SUPPORTI AL TIME SHARING

PER IL TIME SHARING ABBIAMO NECESSARIAMENTE BISOGNO DEL CONCETTO DI INTERRUZIONE.

Iterruzioni

- permettono il ritorno del controllo al monitor (ora chiamato **sistema operativo**)

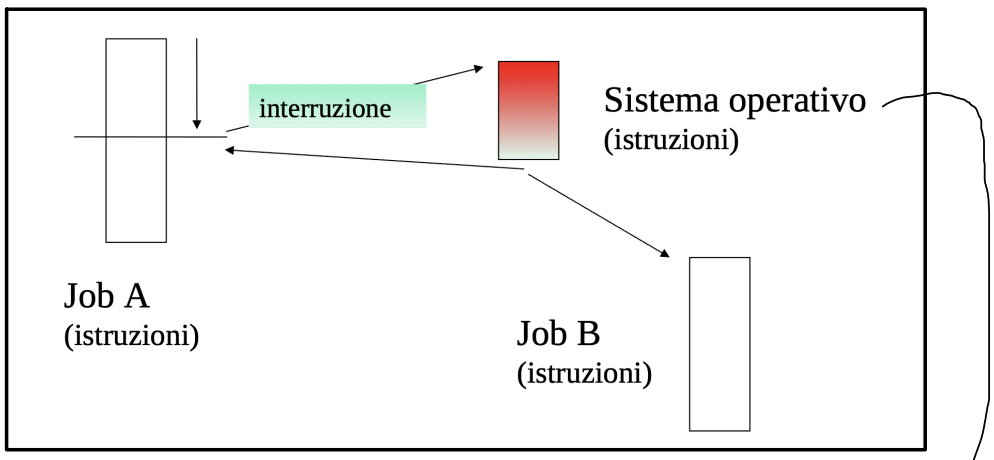
NOI DOBBIAMO AVERE LA POSSIBILITÀ DI INTERROMPERE QUESTO FLUSSO D'ESECUZIONE.

Quando noi abbiamo un JOB attivo e tramite il fetch noi andiamo ad eseguire le sue istruzioni macchina, è possibile che qualcosa nello stato dell'hardware, indichi di NON ESEGUIRE più a fetchare non più in maniera successiva.

Quindi un'interruzione può portare il controllo ad un modulo del sistema operativo che gira l'algoritmo di time-sharing, e quindi l'algoritmo di scheduling della CPU, e questo software può andare a vedere i programmi che eventualmente sono attivi, nella fatto JOBA e JOB B, e può decidere se ridare il controllo al JOBA o passare il controllo al JOB B.

Se ridiamo il controllo al JOBA, stiamo dicendo che quest'ultimo è più prioritario del JOB B.

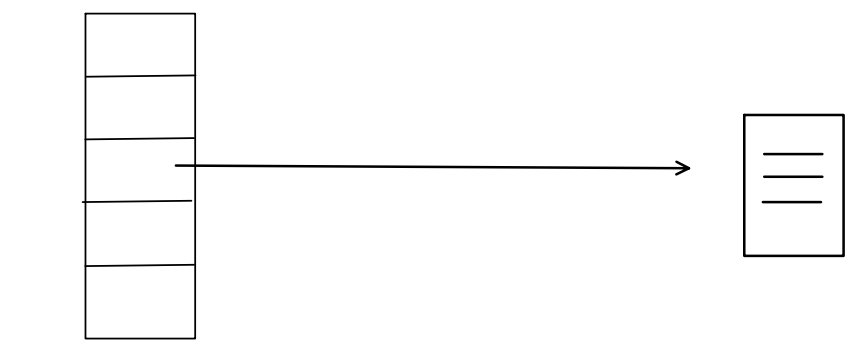
Le interruzioni sono la base per la costruzione di sistemi Time-Sharing.



FUNZIONALITÀ DELLE INTERRUZIONI!

- L'interruzione è possibile che trasferisca il controllo ad una routine di interrupt, il tutto avviene tramite un INTERRUPT VECTOR dove noi registriamo gli indirizzi di memoria di quali sono le routine che devono poter eventualmente gestire le interruzioni:

quando arriva l'interruzione, essa è identificata da un codice numerico che ci identifica uno degli elementi di quel vettore e il singolo elemento che identifichiamo, al suo interno ha l'indirizzo di memoria del blocco di software che deve partire. L'hardware, quando questa interruzione arriva, sapendo dove la tabella è presente in memoria, può andare a caricare nel program counter l'indirizzo di memoria che ci permette di andare ad eseguire quel blocco di software.



L'architettura che utilizza gli interrupt per andare ad interrompere il flusso di esecuzione di una applicazione, per esempio per implementare il time sharing, deve implementare anche meccanismi di salvataggio dell'indirizzo della prossima istruzione del programma interrotto e del valore dei registri di CPU.

Ogni sistema operativo moderno è INTERRUPT DRIVEN.

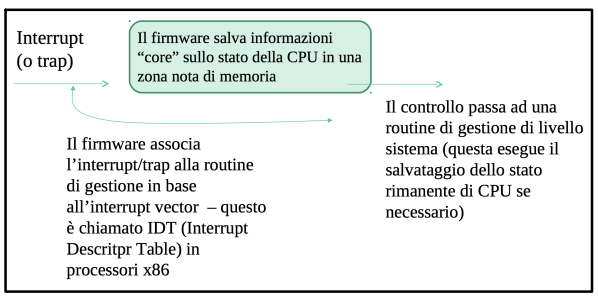
Quando arriva un interrupt (trap) il firmware salva informazioni "core" sullo stato della CPU in una

NOTA ZONA di memoria E PASSIAMO poi IL CONTROLLO ad una ROUTINE di gestione della INTERRUZIONE.
IL FIRMWARE ASSOCIA all'INTERRUZIONE che si è verificata quale è la ROUTINE di gestione utilizzando IL MECCANISMO dell' INTERRUPT VECTOR (IDT IN PROCESSORI x86 è IL NOME della TABELLA).

Ricorda: IL MONITOR ERA UNA VERSIONE PRIMORDIALE di UN SISTEMA OPERATIVO. VENIVA CHIAMATO MONITOR la componente software di SISTEMA dei SISTEMI OPERATIVI BATCH.
IL MONITOR NON CARATTERIZZA i SISTEMI TIME-SHARING, solo quelli BATCH!
ORA SI CHIAMA SISTEMA OPERATIVO.

Un esempio di algoritmo time-sharing è il *Round-Robin,* letteralmente "carosello", che può supportare l'esecuzione di 4 job attivi in un certo intervallo di tempo, a turno questi job prendono il controllo della CPU. Con l'implementazione del time-sharing nei sistemi batch multitasking si ha il primo esemplare di sistema operativo, non più monitor.
Ciò che fa quindi il S.O. nel modello time-sharing è, basandosi su un intervallo di tempo stabilito, togliere il controllo al programma in esecuzione indipendentemente dal suo stato e dal suo set di istruzioni in esecuzione. Quest'attività si denomina con il termine di *pre-emption,* il concetto fondamentale per una pre-emption è quello di *interrupt* (o trap), che cambia lo stato della CPU bloccando il flusso d'esecuzione del job e passando il controllo di nuovo all'algoritmo di scheduling, a seguito della gestione dell'interrupt stesso. Un interrupt può essere causato da un dispositivo (interrupt hardware) o da un comando (interrupt software); infatti esistono proprio *istruzioni di trap,* usate per richiedere l'intervento del sistema operativo, una di queste è l'istruzione *exit(0)* la quale invoca il sistema operativo per bloccare definitivamente il flusso di esecuzione corrente e dismettere il job dalla memoria.

Il sistema operativo reagisce all'interrupt invocando un comando appropriato (routine), che è associato a quell'interrupt specifico tramite un *interrupt vector,* contenente gli indirizzi di memoria che identificano le routine che gestiscono l'interrupt che si è scatenato. Nei sistemi x86 l'interrupt vector si chiama *Interrupt Descriptor Table (IDT)*.
Un'architettura Hw/Sw che usa gli interrupt deve essere in grado di ripristinare il flusso originale una volta che il job in questione riprende il controllo, cioè il job deve tornare nello stato in cui si trovava prima di essere interrotto; questo significa salvare e ripristinare informazioni di esecuzione, come il Program Counter e i registri della CPU in aree di memoria note; notare che il gestore dell'interrupt, cioè la routine chiamata secondo l'interrupt vector, può salvare informazioni aggiuntive a seconda della criticità dell'interrupt scatenato. Ogni S.O. odierno è *interrupt-driven*, cioè è basato sulla gestione degli interrupt.



- un'interruzione trasferisce il controllo ad una "routine di interrupt"
- tipicamente questo avviene attraverso un **interrupt vector**, che contiene gli indirizzi delle rispettive routine nel codice del sistema operativo
- un'architettura che gestisca gli interrupt deve fornire supporti per il salvataggio dell'indirizzo della prossima istruzione del programma interrotto (PC) e del valore dei registri di CPU
- ogni sistema operativo moderno e' **interrupt driven**