Lezione 10

Cenni ai sistemi operativi distribuiti
2. Gestione della CPU e della memoria nei multiprocessori

- Gestione dei processi
  - Scheduling
  - Bilanciamento del carico
  - Migrazione dei processi
- Gestione della memoria
  - Mutua esclusione
  - Coerenza delle cache
  - Memoria condivisa virtuale

Scheduling nei sistemi multiprocessori

- Analogamente ai sistemi monoprocessori, lo scheduling nei sistemi multiprocessore ha come obiettivi
  - Massimizzazione del throughput
  - Minimizzazione del tempo medio di attesa
- Diversamente dai sistemi monoprocessori, lo scheduling nei sistemi multiprocessore deve
  - Non solo decidere l'ordine con cui eseguire i processi
  - Ma anche determinare quale processore deve eseguire un dato processo
- Osservazione: di solito su sistemi multiprocessore sono eseguiti applicazioni parallele composte da piu' processi (insieme di processi cooperanti e correlati)

Ruolo chiave dello scheduling nei sistemi operativi per sistemi a molti processori

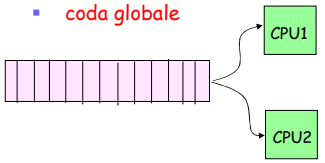
Scheduling dei processi

- Il modo piu' semplice per sviluppare uno scheduler per un sistema multiprocessore e' immaginare
  - un'unica coda di processi pronti (coda globale)
    oppure
  - una coda di processi pronti per ogni CPU (code multiple)

  gestite secondo una qualunque politica di scheduling monoprocessore
- Ogni volta che un processore risulta disponibile viene assegnato un processo in testa alla coda (alle code) al processore

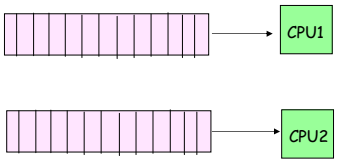
Esempio: P=2 processori

- coda globale



- Vantaggio: condivisione del carico automatica
- Svantaggio: sincronizzazione nell'accesso alla coda

- code multiple



- Vantaggio: possibile riutilizzo dei dati presenti nelle cache
- Svantaggio: possibile sbilanciamento del carico di lavoro

problema

- L'approccio descritto assegna i processi alla CPU "alla cieca", senza tenere conto delle caratteristiche dei processi e delle CPU (scheduling job-blind)

- due grossi svantaggi
  1. Processi cooperanti di una stessa applicazione potrebbero essere schedulati "in sequenza" e non viene sfruttato il parallelismo (soprattutto se comunicano)
  2. Un processo prelazionato, potrebbe non essere schedulato sulla stessa CPU, e non puo' piu' utilizzare i dati che aveva nella propria cache

Condivisione di tempo

Un sistema parallelo e' composto da numerose CPU

Obiettivo 1 : Massimizzazione del parallelismo

Strategia: raggruppare processi cooperanti di una stessa applicazione e schedularli assieme sulle varie CPU

Scheduling a condivisione di tempo

Partizionamento dello spazio

Un processo che gia' ha avuto accesso ad una CPU ha gia' delle pagine nei TLB (o nella cache) di quel processore

Obiettivo 2 : favorire l'affinita' con la CPU

Strategia: assegnare un processo ad una CPU (o a un numero ristretto di CPU che condividono un'area di memoria) in maniera da massimizzare i TLB hit

Scheduling a partizionamento di spazio

Scheduling job aware

Gli algoritmi di scheduling a condivisione di tempo o a partizionamento dello spazio fanno parte di un approccio che tiene conto del tipo di job e per questo e' chiamato

Scheduling job aware

Svantaggio: maggiore complessita' dell'algoritmo di scheduling

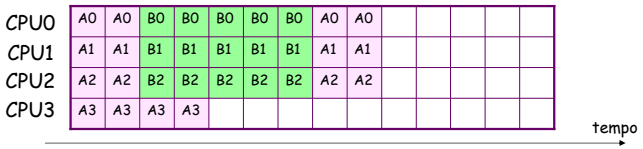
Esempio di scheduling (1)

- Algoritmo Smallest Number of Process First (SNPF)
- Versioni con e senza prelazioni
- Coda globale a priorita'
  - La priorita' e' inversamente proporzionale al numero di processi che compone l'applicazione
- I processi associati allo stesso job ricevono la stessa priorita' e quindi sono spesso schedulati insieme
  - Massimizzazione del parallelismo ➔ si (se non c'e' prelaz.)
  - massimizzazione dell'affinita' con le CPU ➔ buona ma non garantita
  - Possibilita' di posticipazione indefinita o di asincronia per applicazioni con molti processi

Esempio: 4 CPU

- SNPF con prelazione

appl.	arrivo	proc	burst
A	T=0	4	40
B	T=20	3	50



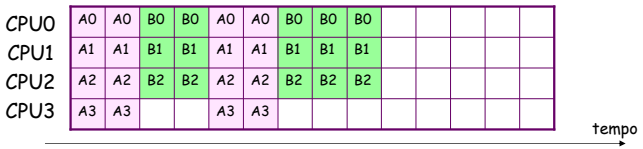
Esempio di scheduling (2)

- Algoritmo di coscheduling (schedulazione gang)
- Basata su round robin
- Processi di una stessa applicazione vengono schedulati sempre insieme
  - Massimizzazione del parallelismo ➔ si
  - massimizzazione dell'affinita' con le CPU ➔ buona ma non garantita
- Non c'e' posticipazione indefinita

Esempio: 4 CPU

- Coscheduling e q=20

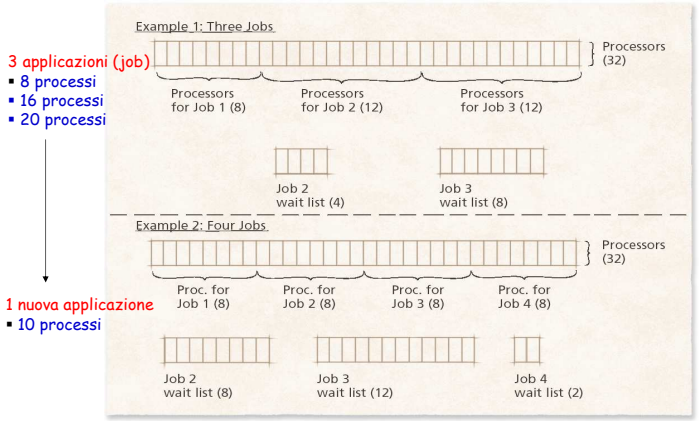
appl.	arrivo	proc	burst
A	T=0	4	40
B	T=20	3	50



Esempio di scheduling (3)

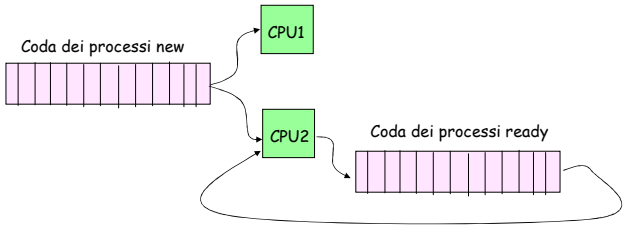
- Partizionamento dinamico
- Distribuisce equamente le applicazioni sulle CPU
- Il numero di CPU assegnate a ogni applicazione e' sempre minore del numero di processi dell'applicazione
  - Una applicazione e' sempre in esecuzione su un gruppo di CPU
  - Quando entra una nuova applicazione il sistema aggiorna il numero di CPU assegnate
  - Se cambia il numero di CPU ogni applicazione continua a utilizzare un sottoinsieme (o un sovrainsieme) del gruppo precedentemente assegnato
- Obiettivo principale: massimizzare l'affinita' con le CPU

Esempio: 32 CPU



osservazione

- Gli approcci descritti non sono mutuamente escusivi



- Appena creato il processo viene assegnato ad una CPU qualunque con il metodo a condivisione di tempo usando una coda globale
- Successivamente rimane in una coda locale alla CPU con il metodo a partizionamento di spazio

Scheduling a due livelli

Bilanciamento del carico

- Un obiettivo generale in tutti i sistemi con molte CPU e' il bilanciamento del carico:
  - Distribuire equamente il carico dei processi su tutte le CPU, in maniera che nessuna di esse rimanga inutilizzata mentre le altre sono sovraccariche
  - Riduce il tempo medio di risposta dei processi
- Due tipi
  - Bilanciamento statico
  - Bilanciamento dinamico

Bilanciamento statico

- Bilanciamento statico
  - Una applicazione viene assegnata ad un gruppo di CPU e vi rimane per tutta l'esecuzione
  - Basso sovraccarico
  - Non tiene conto delle variazioni del numero e del tipi di processi di una applicazione
  - Utile quando l'applicazione ha un comportamento predicibile e noto prima dell'esecuzione (es sistemi con esigenze real time)

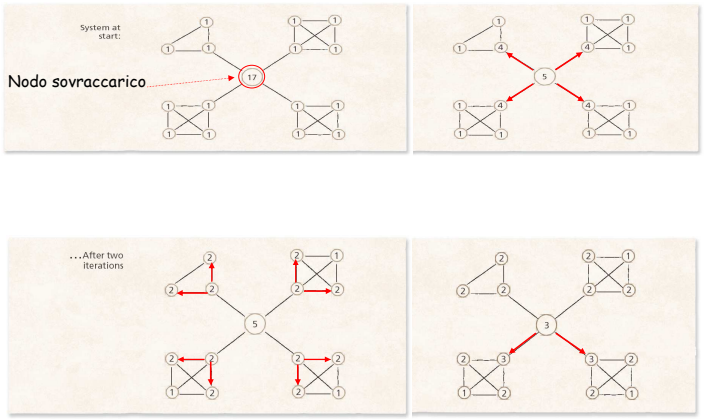
Bilanciamento dinamico

- Il sistema misura il carico di lavoro di ogni processore
- Quando il carico tra i nodi e' sbilanciato il sistema trasferisce i processi dai nodi sovraccarichi a quelli scarichi (migrazione dei processi)
- Utile in applicazioni dove le comunicazioni tra i nodi non sono costanti e quando i processi hanno un tempo di esecuzione non predicibile

- Problemi
  - Bilanciare il costo della migrazione con il vantaggio che ne puo' derivare (funzione di costo)
  - Migrazione su nodi molto lontani puo' essere dispendiosa
  - Carico delle cpu puo' variare nel tempo e rendere inutile la migrazione

- soluzione
  - Comunicazione solo tra nodi vicini
  - Diffusione del carico attraverso il sistema

Esempio di diffusione del carico a nodi vicini



problema

- Quando iniziare la migrazione dei processi?

- Due politiche:
  - Iniziata dal sender:
    - la migrazione ha luogo quando il sistema registra un nodo sovraccarico .
    - Utile per sistemi generalmente scarichi (ci saranno pochi mittenti)
  - Iniziata dal receiver:
    - la migrazione ha luogo quando il sistema registra un nodo scarico .
    - Utile per sistemi generalmente carichi (ci saranno pochi destinatari)

- Possibili soluzioni ibride

Algoritmo ad offerta

- Esempio di migrazione iniziata dal sender
- I processori con carico di lavoro elevato cercano un numero fissato (ad es. 10) di processori che hanno un carico basso, partendo dai nodi vicini
- un processore che ha un carico di lavoro basso, segnala la sua disponibilita' pubblicando una "offerta"
  - Il valore dell'offerta e' calcolata in base alla differenza di carico tra i due processori e in base alla distanza tra i nodi
- Il sender invia il processo al processore che ha offerto di piu'

Algoritmo drafting

- Esempio di migrazione iniziata dal receiver
- Ogni processore mantiene informazioni del carico degli altri processori in apposite tabelle
- al variare del carico, le informazioni sono fatte scorrere attraverso la rete, ed ogni processore aggiorna le proprie tabelle
- I processori sottoutilizzati possono richiedere processi ai processori sovraccarichi

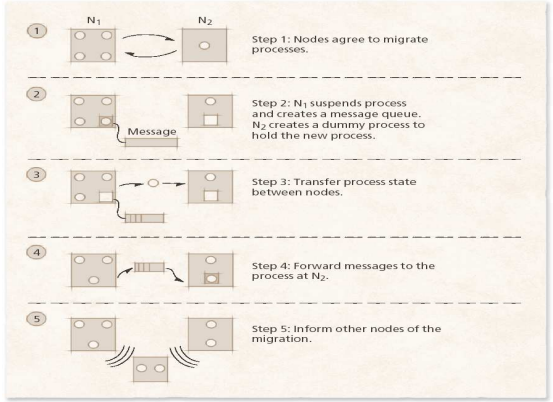
Motivazioni della migrazione dei processi

- Il bilanciamento dinamico del carico e' basato sulla migrazione di processi tra le CPU
- Oltre a cio' sussistono altri aspetti che richiedono la migrazione dei processi
  - Se un nodo si guasta, il lavoro svolto si perde
  - Comunicazone tra processi su CPU distanti
  - Uso di risorse di nodi distanti
- Benefici
  - Miglioramento della tolleranza ai guasti
  - Bilanciamento del carico
  - Riduzione dei costi di comunicazione
  - Condivisione delle risorse

Migrazione freezing e restarting

- obiettivo: trasferire un processo da un nodo ad un altro nodo
- Cosa migra (lo stato del processo)
  - Pagine del processo nellamemoria virtuale
  - Contenuto dei registri della CPU
  - Stato dei file aperti
- Come migra una volta che i processi sorgente e destinatario si accordano per la migrazione
  - Il sorgente sospende il processo che deve migrare
  - Il sorgente crea una coda per i messaggi diretti al ricevente
  - Il sorgente trasmette lo stato del processo ad un processo "vuoto" creato dal destinatario destinatario
  - Il sorgente e il destinatario comunicano a tutti gli altri nuovi la nuova posizione del processo
  - Il sorgente elimina l'istanza locale del processo

Migrazione di un processo



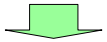
Meccanismo freezing e restarting (o migrazione impaziente)

Caratteristiche di un buon meccanismo di migrazione

- Trasparenza
  - Altri processi cooperanti non dovrebbero notare se un processo e' migrato
- Minima dipendenza residua
  - Il processo migrato non dovrebbe piu' usare informazioni sulla vecchia CPU (info sulla paginazione, comunicazione con processi sul vecchio nodo)
- Efficienza
  - Minimo tempo di trasferimento e minimo costo di rilocazione

Costo della migrazione

- In generale la migrazione freezing e restarting e' un meccanismo semplice da implementare ma costoso e poco efficiente
  - Contesto (registri, program counter) pochi KB
  - Spazio di indirizzamento (pagine della memoria) molti MB



- Una soluzione: migrazione pigra
  - Viene trasferito solo il contesto
  - Le pagine della memoria vengono trasferite al momento del riferimento
  - Basso overhead ma alta dipendenza residua

Meccanismi alternativi

- Migrazione sporca
  - Viene trasferito il contesto e le pagine modificate (sporche)
  - Le pagine non modificate risiedono su un disco comune ai due processori
  - Overhead maggiore ma minore dipendenza residua
- Migrazione flushing
  - Viene trasferito solo il contesto
  - le pagine sporche in memoria vengono salvate su un disco comune ai due processori
  - Basso overhead ma molti page fault in seguito
- Migrazione con precopia
  - Si trasferiscono le pagine sporche prima della sospensione del processo, sovrapponendo esecuzione e migrazione
  - Pagine trasferite ed eventualmente modificate vengono ritrasmesse
  - Ottimale se durante il trasferimento delle pagine vengono modificate poche pagine

La gestione della memoria

- Un sistema multiprocessore e' composto da piu' CPU che condividono un'unica memoria

  Problema della sincronizzazione degli accessi
  (mutua esclusione)

- le tecniche per sistemi monoprocessori non sono sempre utilizzabili:
  - Disabilitazione delle interruzioni
    - ➔ non garantisce mutua esclusione
  - Test and set
    - ➔ saturazione del buffer se molte CPU testano la variabile lock

Lock pausa/risveglio

- Simile alle condizioni basate su attesa attiva ma con minor spreco di cicli di CPU e minor traffico sul bus
  - Il primo processo che chiede una risorsa protetta ottiene il lock
  - Processi successivi che richiedono la risorsa e non la ricevono si pongono in attesa
  - Il processo che rilascia la risorsa sveglia anche il processo in attesa
- Non esclude la race condition

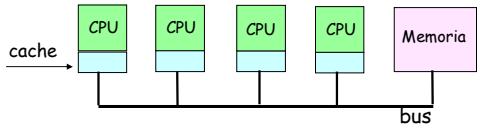
Lock read/write

- Obiettivo: rilassare le condizioni di accesso
- Basato sul protocollo di sincronizzazione lettori/scrittori
  - Una CPU alla volta puo' accedere alla memoria condivisa se deve scrivere
  - Molte CPU possono leggere contemporaneamente

Sistemi multiprocessori NUMA

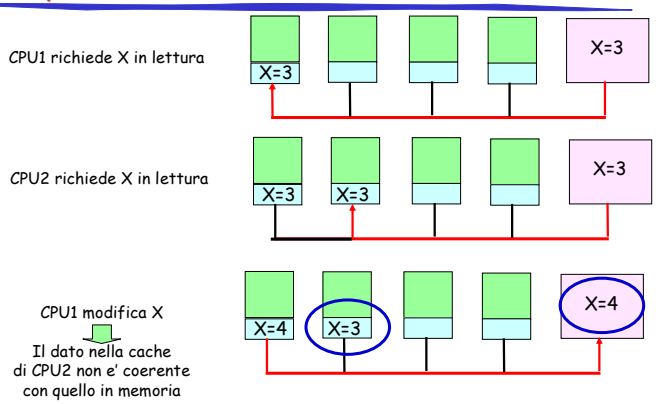
- Alcuni multiprocessori hanno una propria memoria locale (o una cache) per ridurre il traffico sulla rete o per ridurre i problemi di conflitto in memoria centrale

Sistemi Non Uniform Memory Access



PROBLEMA:
Coerenza delle cache

esempio



Necessita' di un protocollo di coerenza

Protocollo write through

Azione della CPU che richiede l'operazione	altre CPU: Il dato e' nella cache?	altre CPU: azione
Preleva il dato dalla memoria e conservalo nella cache	no	Non fare niente
Usa il dato della cache	si	Non fare niente
Scrivi il dato in memoria e conservalo nella cache	no	Non fare niente
Aggiorna il dato in memoria e nella cache	si	Invalida la cache

In caso di successivi riferimenti si preleva nuovamente il dato (aggiornato) dalla memoria

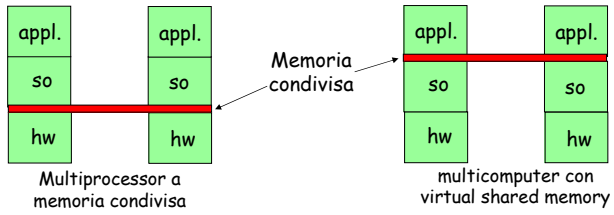
Memoria condivisa virtuale

- Le cache delle CPU di un sistema multiprocessore possono essere viste come l'analogo delle memorie locali di un sistema multicomputer
- I protocolli di coerenza della cache forniscono un meccanismo per condividere dati che sono in realta' distribuiti
- La memoria condivisa virtuale e' una simulazione di una memoria condivisa reale in un sistema distribuito
- Vantaggi:
  - Astrazione piu' semplice
  - Movimento dei dati solo quando serve
  - Spazio piu' ampio di memoria
  - Migrazione dei processi piu' economica (migra solo il codice)

Memoria condivisa virtuale

- Molti programmatori ritengono piu' naturale un ambiente di programmazione a memoria condivisa

Simulazione via software della memoria condivisa
(virtual shared memory)



Idea di base

- Le pagine di un processo possono essere presenti nella memoria di un nodo qualunque
- Un modulo sw (mapping manager) fa corrispondere le pagine dello spazio di memoria condivisa alle pagine fisiche
  - Se la pagina e' locale la si usa
  - Se la pagina e' remota si blocca il processo e la si carica dalla cpu che la possiede (analogamente ad un page fault)
- Protocolli di coerenza analoghi a quelli delle cache