

# Appunti di Sistemi operativi

domenica 18 marzo 2007

15.56



Questo/a opera è pubblicato sotto una Licenza Creative Commons.

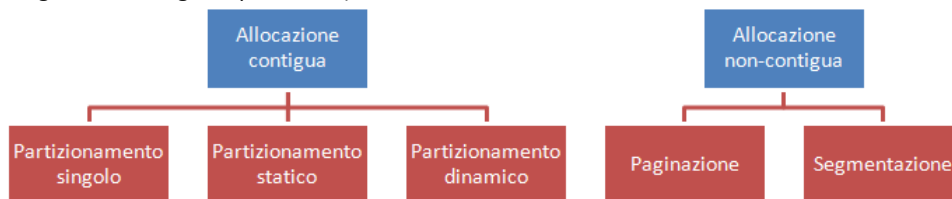
Se queste pagine vi sono state utili considerate di spedirmi una cartolina.

Autori: Michele Di Cosmo – Eleonora Annigoni – <http://www.sugata.eu>

# 8 - Gestione della memoria

domenica 18 marzo 2007  
21.43

- Monoprogrammazione / Multiprogrammazione
- **Binding degli indirizzi**
  - o **Compile time**
  - o **Load time**
  - o **Execution time**
- **Caricamento dinamico** (il segmento non viene caricato finché non viene richiesto)
- **Collegamento dinamico delle librerie** (uso di stub)
- **MMU** (associa gli indirizzi logici a quelli fisici)



- **Allocazione contigua** (almeno due partizioni: una per il kernel, l'altra/le altre per l'utente)
  - o **Partizionamento singolo** (un processo per partizione)
  - o **Partizionamento statico** (partizioni fisse uguali o diverse) (porta a frammentazione interna)
    - First-fit (il primo processo abbastanza grande per il buco)
    - Best-fit (il primo processo più grande uguale al buco)
  - o **Partizionamento dinamico** (decise a run-time)(porta a frammentazione esterna, risolta con la compattazione, possibile solo con rilocazione dinamica. Particolarità con l'uso del DMA)
    - First-fit (alloca il primo buco sufficientemente grande) [migliore]
    - Next-fit (come first-fit, ma partendo da dove era rimasto) [migliore]
    - Best-fit (il più piccolo buco sufficientemente grande) [frammenta molto]
    - Worst-fit (il più grande buco sufficientemente grande) [lento]
- **Allocazione non-contigua**
  - o **Paginazione** (niente frammentazione esterna, ridotta frammentazione interna)
    - Si divide la memoria fisica in frame e la logica in pagine, della stessa dimensione
    - Si implementa una page-table per tener traccia dei collegamenti frame-pagina
    - Indirizzamento tramite numero di pagina (p) e offset nella pagina (d)
    - Permette la condivisione di codice read-only (quindi "rientrante"). Il codice condiviso però appare nelle stesse locazioni logiche
    - Protezione implementata con un bit (di protezione) che indica il diritto di accesso (se la pagina rientra nello spazio logico del processo)
  - o **Segmentazione** (frammentazione esterna, non c'è frammentazione interna)
    - Un programma viene visto come un insieme di segmenti suddivisi per unità logica (funzioni, variabili, ecc...)
    - Indirizzamento tramite numero di segmento e offset nel segmento
    - La segment-table mappa gli indirizzi utente a quelli fisici
    - Implementa rilocazione dinamica sui segmenti
    - Permette la condivisione di segmenti
    - L'allocazione viene implementata come l'allocazione contigua
    - Protezione implementata con un bit (di protezione) e con bit per read/write/execute
    - I segmenti hanno lunghezza variabile in run-time
    - **Differenze con la paginazione:**
      - I segmenti sono rilocabili e di dimensione variabile
- **Implementazione della page-table**



- o **In memoria**
  - Ogni accesso ai dati in memoria richiede due accessi alla memoria (query alla page-table)
- o **Hardware TLB**
  - Il numero di pagina viene confrontato contemporaneamente a tutte le entry del TLB
  - Se non si trova un elemento, si procede con il doppio accesso standard alla memoria (questa query rimane invisibile al SO)

- **Software TLB**
  - Come la versione hardware, ma nel caso di una TLB-miss, si genera un interrupt al processore e quindi è il SO a fare in seguito la query alla memoria principale
  - Abbastanza efficiente con TLB grandi ed estremamente semplice. Molto utilizzato.
- 📖 ○ Tempo effettivo di accesso con TLB
- **Paginazione a più livelli**
  - Per ridurre l'occupazione della page-table, anche questa viene paginata
  - Performance: aumentano gli accessi alla memoria, ma il caching degli indirizzi di pagina permette di ridurre drasticamente questo fattore
- **Tabella delle pagine invertita**
  - Le tabelle hanno entry per i frame e non per le page. Si memorizzano le info sul processo che possiede la pagina
  - Meno memoria, ma più tempo per l'accesso
  - Usato dove le pagine occuperebbero troppo spazio
- **Tabella delle pagine invertita con hashing**
  - Si implementa una funzione di hash per ridurre il tempo di ricerca
- **Segmentazione con paginazione (MULTICS)** (si elimina la frammentazione esterna: combina i vantaggi di entrambi gli approcci)
  - I segmenti vengono paginati
  - Nelle segment-table vengono memorizzati gli indirizzi delle pagine
  - Migliora la gestione di segmenti molto grandi
- Segmentazione con paginazione a due livelli
- 📖 - Sommario

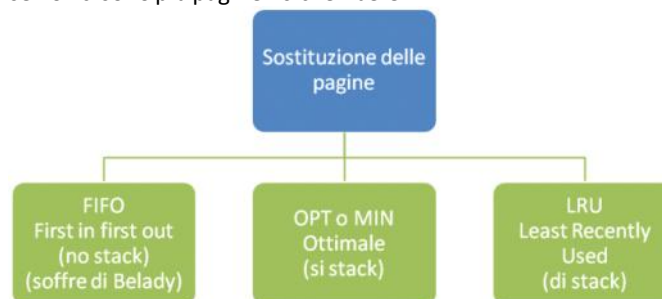
# 9 - Memoria virtuale

lunedì 19 marzo 2007  
20.20

- **Vantaggi:**
  - o Spazio logico molto più grande di quello fisico
  - o Meno consumo di memoria -> più multiprogrammazione
  - o I/O ridotti per il caricamento di un programma
- **Paginazione su richiesta:** (si carica una pagina di memoria solo quando questa viene richiesta)
  - o L'MMU gestisce le pagine non in memoria (e le richieste non pertinenti)
  - o Validità delle pagine (è presente o è nella memoria secondaria?)
  - o Gestione dei page fault
    - Dopo aver reso disponibile la pagina richiesta, l'istruzione deve venir rieseguita in modo consistente (questo può essere più complicato di quanto sembra, nota di sid)
- o Performance
- o Esempi
- o **Considerazioni**
  - Problemi sulla performance se ci sono troppi page fault
  - L'area di swap deve essere molto veloce; preferibilmente senza file system
- **Gestione della memoria per nuovi processi**



- o **Copy on Write**
  - È una modalità in cui può venire creato un processo figlio: permette la condivisione iniziale delle pagine di memoria
  - Nel caso in cui uno dei processi scrive su una pagina, questa viene duplicata prima della scrittura.
  - Permette una creazione più veloce dei processi
  - Le pagine libere devono essere azzerate per eliminare possibili informazioni sensibili (Windows azzerava le pagine periodicamente)
- o **Memory-Mapped I/O**
  - Permette di gestire I/O di file tramite accessi alla memoria: ogni blocco viene mappato su una pagina virtuale
  - Permette di leggere una sola volta un file: le parti lette rimangono in memoria
  - La gestione dell'I/O è molto più semplice
  - Permette di condividere le informazioni di un file da parte di più processi
- **Sostituzione delle pagine:** se non ci sono più pagine fisiche libere...



- o Il 'dirty bit' indica che una pagina è stata cambiata e pertanto, quando la pagina viene scaricata, deve essere copiata nello swap.
- o **Algoritmi** (devono tendere a minimizzare i page fault)
  - **FIFO** - First in First out
    - o Si scarica la pagina che da più tempo è in memoria
    - o Non è un algoritmo di stack. Quindi soffre dell'anomalia di Belady
  - **OPT o MIN** - Algoritmo ottimale (usato nei confronti fra algoritmi)
    - o Si scarica la pagina che non verrà riusata per il periodo di tempo più lungo
    - o È un algoritmo di stack
  - **LRU** - Least Recently Used
    - o Studia il passato per prevedere il futuro (approssimazione di OPT)
    - o Si scarica la pagina che da più tempo non viene usata
    - o Buona soluzione, ma richiede molta assistenza hardware
    - o È un algoritmo di stack

- Matrice di memoria
- Implementazioni di LRU



- **Implementazione a contatore**
  - La MMU umenta un contatore e lo imposta nel registro 'reference time' della entry della page acceduta ad ogni accesso alla memoria
  - Si determina che pagina scaricare in base al numero più basso
  - Implementazione molto dispendiosa se la ricerca viene parallelizzata nell'hardware (?)
- **Implementazione a stack**
  - Si crea uno stack con i numeri di pagina usati (in una lista double-linked)
  - Al riferimento ad una pagina, si sposta il suo riferimento all'inizio dello stack
  - Si scarica la pagina più in basso nello stack
  - Implementazione software molto costosa (spostare sul top richiede 6 modifiche ai puntatori)
- **Approssimazione con bit di riferimento** (impreciso perché non si conosce l'ordine di accesso)
  - Al riferimento ad una pagina, il suo bit 'reference bit' viene settato
  - Si scarica la pagina con reference bit vuoto
- **Approssimazione NFU - Not frequently used**
  - Ad ogni pagina si associa un contatore che si auto-incrementa ogni 10~20 ms.
  - Al riferimento il contatore viene azzerato
  - Problema: le pagine usate molto tempo fa contano come quelle recenti
- **Approssimazione con aging** (basato sul bit di riferimento)
  - Ad ogni pagina si associa un array di bit vuoti che ad intervalli regolari viene shiftato e il primo bit sostituito con il 'reference bit'
  - Si scarica la pagina con il numero più basso
  - Buona approssimazione
- **Approssimazione CLOCK** (o Second chance) (basato sul bit di riferimento)
  - Scrolla tutte le pagine dalla prima. Se il 'reference bit' è vuoto, la pagina viene sostituita; se non il 'reference bit' viene azzerato e si passa alla pagina successiva
  - Vengono lasciate in memoria le pagine accedute molte volte e velocemente
  - Buona approssimazione, usato spesso
- **Approssimazione CLOCK migliorata**
  - Si usa il 'reference bit' e il 'dirty bit'
  - Vengono eseguite più scansioni alla ricerca di una pagina che soddisfi una di queste condizioni:

	'reference bit'	'dirty bit'	
1)	Non usata	Non modificata	
2)	Non usata	Modificata	Viene azzerato il 'reference bit'
3)	Usata	Non modificata	
4)	Usata	Modificata	

Nota: dopo il punto due si ritorna al punto 1 implementando di fatto il 3 e il 4

#### - Trashing

- Viene speso molto tempo in swap -> basso carico alla CPU
- Il basso carico della CPU può far pensare all'algoritmo per la multiprogrammazione di dover aumentare i processi per sfruttare meglio la CPU, risultando in un aggravamento del trashing
- **Trashing del processo**
  - Avviene quando un processo ha meno memoria fisica di quanto la sua località richiede
- **Trashing del sistema**
  - Avviene quando il sistema ha meno memoria fisica di quanto tutte le località dei processi in esecuzione richiedono
  - Può essere causato da un processo che si espande quando viene usata una politica di rimpiazzamento globale

- **Principio di località**
  - Località: Insieme di pagine utilizzate attivamente assieme
- **Come impedire il trashing**



- **Modelli**
  - **Modello del working-set**
    - ◆ Vengono esaminati i più recenti delta (working set window) riferimenti alle pagine. L'insieme di queste pagine è il working set
    - ◆ **Algoritmo** basato sul working-set:
      - ◇ Viene controllato il working-set e vengono allocati sufficienti pagine
      - ◇ Alla creazione il processo viene messo in ready solo se ci sono sufficienti pagine
      - ◇ Se le pagine richieste sono maggiori delle pagine disponibili si sospende il processo per diminuire la multiprogrammazione
    - ◆ **Modelli approssimativi** del working-set
      - ◇ **Algoritmo basato su registri a scorrimento**
        - ▶ Si mantengono due bit aggiuntivi per pagina oltre al 'reference bit'
        - ▶ Ogni tick di un timer i bit vengono shiftati
        - ▶ Se uno dei tre bit è settato, si considera la pagina nel working set
        - ▶ Si può migliorare aumentando i bit e diminuendo l'unità di tempo
      - ◇ **Algoritmo basato sul tempo virtuale**
      - ◇ **Algoritmo WSCLOCK (variante del CLOCK con working-set)**
    - **Modello della frequenza di page-fault**
      - ◆ Definito un page-fault-rate accettabile, se questo è superiore viene regalata una pagina al processo, altrimenti gli viene rubata
  - **Tipi di sostituzione**
    - **Sostituzione locale:** le pagine dei processi vengono sostituite solo fra di loro
      - ◆ Il comportamento non è influenzato da quello degli altri processi
      - ◆ Questo metodo supporta i modelli con working-set
    - **Sostituzione globale:** le pagine dei processi vengono sostituite fra quelle del sistema
      - ◆ Non supporta modelli con working-set
  - **Algoritmi di allocazione**

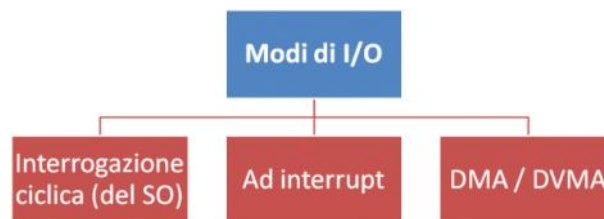


- **Allocazione libera:** vengono assegnate tutti i frame richiesti
- **Allocazione equa:** a tutti un numero fisso di frame: sprechi
- **Allocazione proporzionale:** numero di frame proporzionale alla dimensione del processo e alla sua priorità
- **Ottimizzazione**
  - **Buffering delle pagine libere**
    - Se la pagina è stata messa nella free-list, ma non è stata modificata, la si toglie semplicemente dalla free-list (in questo caso si parla di soft-page-fault, altrimenti di hard-page-fault)
  - **Pre-paging** (al lancio del programma si sa già quale pagina servirà, come anche al ripristino successivo ad uno swap-out)

# 10 - Sistemi di I/O (appunti generali)

martedì 20 marzo 2007  
12.56

- **Categorie**
  - o Human readable (terminale, mouse)
  - o Machine readable (disco, nastro)
  - o Comunicazione (modem, schede di rete)
- **Visione astratta** dei sistemi di I/O da parte dell'utente
- **Tipi di comunicazione**
  - o A blocchi (dischi)
    - Tramite File System
    - Operazioni: Read, Write, Seek
    - Può essere mappato in memoria
  - o A carattere (tastiera)
    - Tramite stream di dati
    - Operazioni: get, put di caratteri e parole
  - o Dispositivi di rete
    - Tramite socket
  - o Altro (timer, nastri)
- **Modalità di comunicazione**
  - o Istruzioni I/O dedicate (necessita passaggio dal kernel)
  - o I/O mappato in memoria
  - o I/O separato in memoria (come I/O mappato in memoria, ma viene usato un segmento dedicato)
- **Modi di I/O**



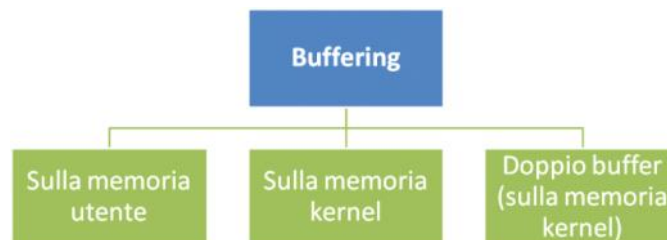
- o **Interrogazione ciclica** (programmed I/O)
    - Il SO invia il comando al dispositivo
    - Il SO interroga ciclicamente lo stato del dispositivo per vedere se ha terminato l'operazione
  - o **Ad interrupt**
    - Il SO invia il comando al dispositivo e sospende il processo (proseguono altri thread/processi)
    - Il SO riceve un interrupt al termine dell'operazione del dispositivo
  - o **DMA / DVMA**
    - Nel DVMA (Direct virtual memory access) lo scambio di informazioni avviene non sulla memoria fisica ma su quella virtuale (ad esempio l'AGP)
- **Gestione degli interrupt**



- o Salvataggio della CPU
  - Dove?
    - **Su dei registri shadow** (niente annidamento)
    - **Su stack**
      - ◆ **Dell'utente** (problemi di sicurezza e page-fault)
      - ◆ **Del kernel** (overhead per MMU e cache)
  - **Problemi dello stato**
    - con CPU con pipeline (sto puntando alla prossima istruzione da mettere nella pipeline, non a quella da eseguire)
    - con CPU superscalari (ho già eseguito alcune istruzioni successive e ho modificato lo stato di conseguenza)
  - **Tipi di interrupt**
    - Interrupt precisi
      - ◆ Istruzioni passate eseguite, future non eseguite. Conosco il PC.
    - Interrupt imprecisi
      - ◆ Il compito di gestire gli interrupt viene demandato al SO

- **Struttura della comunicazione** con i dispositivi di I/O
  - 1) Applicativo
  - 2) Kernel
  - 3) (Kernel - Sottosistema di I/O) (è indipendente dai dispositivi)
  - ★4) **Driver del dispositivo** (si occupano di comandare il controller e controllare cosa viene chiesto) (devono essere rientranti) (devono essere uniformi con quello che si aspetta il SO)
    - 1) Controllare i parametri passati
    - 2) Accodare le richieste in una coda di operazioni (soggette a scheduling)
    - 3) Eseguire le operazioni, accedente al controller
    - 4) Passare il processo in wait (I/O interrupt driven) o attendere la fine dell'operazione busy-wait
    - 5) Controllare lo stato dell'operazione nel controller
    - 6) Restituire il risultato
  - ★5) **Driver delle interruzioni** (si occupa di mantenere lo stato)
    - 1) Salva i registri della CPU
    - 2) Imposta un contesto per la procedura di servizio (TLB, MMU, stack, ...)
    - 3) ACK al controller degli interrupt (per avere interrupt annidati)
    - 4) Creare la copia dei registri nel PCB
    - 5) Eseguire la procedura di servizio (che accede al dispositivo; una per ogni tipo di dispositivo)
    - 6) *Eventualmente cambiare lo stato di un processo in attesa*
    - 7) Organizzare un contesto per il processo successivo
    - 8) Caricare i registri del nuovo processo dal suo PCB
    - 9) Continuare il processo selezionato
- 6) Controller
- **Modalità di gestione dell'I/O**
  - o I/O bloccante (processo bloccato)
  - o I/O non bloccante (ritorna con i primi dati) (implementabile con thread e chiamate bloccanti)
  - o I/O asincrono (ritorna e continua)
- **Compiti del kernel**
  - o Scheduling
  - o Buffering
  - o Caching
  - o Spooling (buffer per i dispositivi che non supportano I/O interleaved, come le stampanti)
  - o Accesso esclusivo
  - o **Gestione degli errori**
    - Errori di programmazione
    - Errori del dispositivo
      - Transitori
      - Non recuperabili
      - Che richiedono l'intervento dell'utente

★- **Buffering**



- o Sulla memoria dell'utente
  - Il processo riserva della memoria per il buffer
  - Nel caso in cui le pagine del buffer non siano presenti in memoria quando arrivano dati può far perdere parte di questi
- o Sulla memoria del kernel
  - Non è soggetto a swap, quindi sono sempre in memoria
  - Potrebbe però essere che le pagine di buffer siano piene e il tempo impiegato per copiarle nelle relative pagine dati dello spazio utente
- o Doppio buffer (sulla memoria del kernel)
  - Si usano due spazi per il buffer nel kernel
  - Quando uno è pieno, viene copiato nella memoria utente e se arrivano altri dati vengono scritti nel secondo buffer

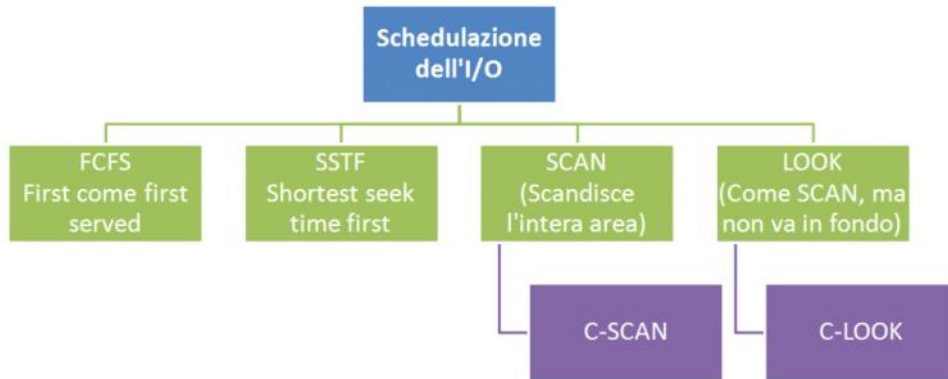
📖- **Performace**

📖- **Livelli di implementazione**

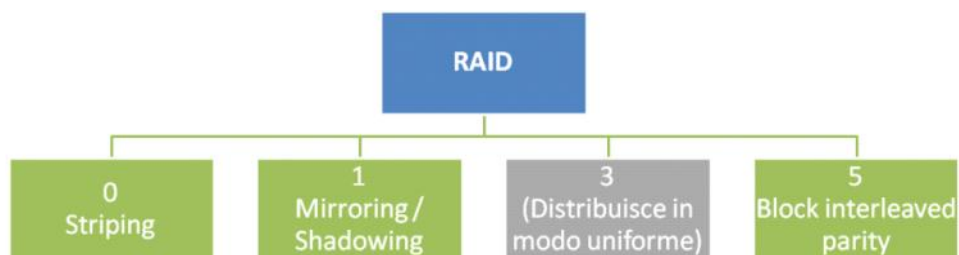
# 11 - Struttura dei dischi

domenica 18 marzo 2007  
23.04

- Dischi come **array di blocchi logici**, mappato sui settori del disco in modo sequenziale
- **Tempi di accesso:**
  - o **Seek time** (tempo per spostare le testine)
  - o **Latenza di rotazione** (tempo aggiuntivo per mettere il settore sotto la testina)
  - o Il compito del SO è quello di minimizzare il seek time (perché tener traccia della posizione angolare del disco è difficile)
- **Algoritmi per la schedulazione delle richieste di I/O su disco**



- o **FCFS** - First come first served
  - o **SSTF** - Shortest seek time first (minor tempo di seek dalla posizione corrente)  
[molto comune, semplice, abbastanza efficiente]
  - o **SCAN** (scandisce l'intera superficie del disco servendo sempre le richieste man mano che si trova nella posizione adatta. Agli estremi inverte la direzione)
  - o **C-SCAN** (SCAN circolare: i cilindri vengono visti come una lista circolare: agli estremi si sposta sull'altro estremo del disco senza servire nulla durante lo spostamento, poi riprende)  
[migliore con grande carico: si evita starvation: il non eseguire mai una richiesta]
  - o **LOOK, C-LOOK** (miglioramento di SCAN, si sposta solo fino alla richiesta attualmente più estrema, non fino alla fine del disco)  
[abbastanza efficiente]
- **Gestione dell'area di swap**
    - o **Alloca lo spazio subito** per il processo e per i segmenti text, lo stack invece su richiesta (4.3BSD)
    - o **Alloca lo spazio solo al page-out** (Solaris 2)
    - o **Alloca lo spazio per ogni pagina non corrispondente ad un file sul FS** (es. DLL) (Windows 2000)
  - **RAID** (Redundant Array of Inexpensive/Independent Disks) - implementa affidabilità memorizzando informazioni ridondanti



- o 0 - Striping (dati parallelizzati: alta performance, nessuna ridondanza)
- o 1 - Mirroring/Shadowing (duplicato intero; basse performance in scrittura)
- o 5 - Block interleaved parity (come striping, ma ad ogni stripe un disco memorizza l'informazione di parità della rimanente stripe; alta resistenza, discrete performance)

## 12 - Implementazione del File System

lunedì 19 marzo 2007  
13.37

### - Struttura dei file system



- Programmi
- File system logico
  - Presenta i file system in un'unica struttura
  - Implementa la protezione
- Organizzazione dei file
  - Effettua la traduzione degli indirizzi logici in fisici
- File system di base
  - Comanda i driver
- Controllo dell'I/O (driver)
- Dispositivo (controller hardware)
- **Tabella dei file aperti** (tabella di file control block) [problemi di affidabilità se manca la corrente]
  - All'open si caricano i dati relativi al file
  - Alle altre operazioni si useranno questi dati
  - Al close le informazioni vengono riportate sul disco e il file control block deallocato
- **Mounting** (del FS fisico in quello logico)
  - Tipi:
    - al boot (freddo)
    - dinamico (caldo)
  - Punto di montaggio
    - fisso (ad esempio A:, B:)
    - configurabile (in qualsiasi punto, ad es. linux)
  - Il kernel determina il tipo di FS
  - Viene richiesto l'unmount
- **Tipi di allocazione**



- **Allocazione contigua** (ogni file occupa un insieme di blocchi contigui sul disco)
  - Semplice l'accesso sia sequenziale che random
  - Porta a frammentazione esterna
  - Problema nella crescita dei file (non possono crescere o portano frammentazione)
  - Frammentazione interna se sono obbligati ad allocare tutto lo spazio che potenzialmente useranno
- ★ ○ **Allocazione concatenata** (ogni file è una lista concatenata di blocchi, situati ovunque nel disco)
  - Lo spazio viene allocato su richiesta
  - Semplice l'accesso
  - Non supporta l'accesso diretto tramite seek
  - Non c'è frammentazione esterna, né interna
  - *Variante:* La FAT è simile: i link della lista viene però memorizzata in una struttura dedicata all'inizio della partizione
- ★ ○ **Allocazione indicizzata** (i puntatori ai blocchi di un file (sparpagliati) si mantengono in una tabella indice)

- Supporta l'accesso random
- Supporta l'allocazione dinamica senza frammentazione esterna
- La struttura per memorizzare l'indice porta a overhead (non so a priori quanto grande sarà il mio file) -> Non è sufficiente un solo blocco indice
  - **Schema concatenato**: l'ultimo indirizzo è un puntatore al prossimo blocco indice
  - **Indice a più livelli**: ogni indirizzo del blocco indice punta ad un'altro blocco indice di livello inferiore

- **Inodes** (unix) (gli inodes vengono alloccati in numero finito alla creazione del FS)

- Mantengono le seguenti informazioni: modo, UID, GID, dimensione, timestamp, numero di link hard che puntano all'inode, puntatori ai primi 12 blocchi (diretti) del file, puntatore al primo blocco indice del file, del secondo, del terzo (questo non viene mai usato)
- Gli indici indiretti vengono alloccati su richiesta
- Risulta in un accesso più veloce per i file piccoli
- Risulta in una dimensione massima indirizzabile pari a 4 TB (è oltre la capacità dei PC a 32 bit)

★ - **Gestione dello spazio libero**

- Implementato come
  - **Vettore di bit** (block map o bitmap): 1 bit per ogni blocco [comoda, facile per trovare blocchi liberi, facile per trovare blocchi contigui, ma c'è spreco di spazio]
  - **Linked list**: [lenta se dobbiamo attraversare la lista, ma buona per l'uso dello spazio]

- **Directory** (collegano i nome dei file ai loro attributi e al loro blocco dati)



○ **Attributi**:

- Memorizzati nelle entry delle directory (ad esempio Ms-Dos o Windows)
- Memorizzati in strutture esterne, quali gli inodes, puntate dalle entry nelle directory

○ **Tipologie**:

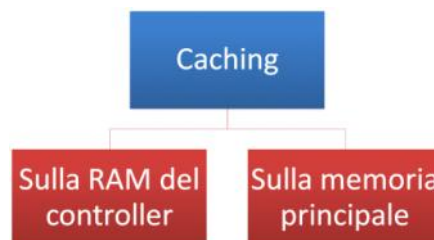
- **Lista lineare** (di file, c'è solo un puntatore ai blocchi dati)
  - Semplice
  - Lenta nella ricerca, inserimento e cancellazione. Può essere migliorata con la cache.
- **Tabella di hash** (come la lista lineare, ma con una struttura di hash per velocizzare l'accesso: la chiave è il nome del file)
  - Più veloce
  - Più difficile a causa delle collisioni (le entry sono realizzate con liste)
- **B-tree** (albero binario bilanciato, la ricerca viene effettuata con una ricerca binaria)
  - Breve tempo di accesso
  - Complessità nel mantenere l'albero bilanciato

○ **Efficienza e performance influenzate da**

- Algoritmi di allocazione dello spazio (e gestione delle directory)
- Tipo di dati
- Grandezza dei blocchi (paginazione a parte)
  - Piccoli abbassano la frammentazione interna (più efficienza)
  - Grandi aumentano le performance

- **Caching** (usare della RAM per bufferizzare i blocchi più usati)

○ **Dove?**



- RAM sul controller come buffer di traccia per ridurre la latenza a quasi 0
- Sulla memoria principale in modo da usare parte (meglio tutte) le pagine libere

○ **Organizzazione dei buffer**

- Coda con accesso di hash gestita con LRU o CLOCK, ecc...
- Un blocco viene scritto sul disco quando deve venir rimosso dalla coda
- **Pericolo se avviene un crash**: i blocchi critici possono essere inconsistenti: viene implementata una variante dell'LRU che divide in:
  - Blocco usato fra poco: posizionato in fondo alla lista
  - Blocco critico per la consistenza (tutti eccetto dati): posizionato all'inizio: la modifica viene trasferita immediatamente

- **Sincronizzazione automatica**

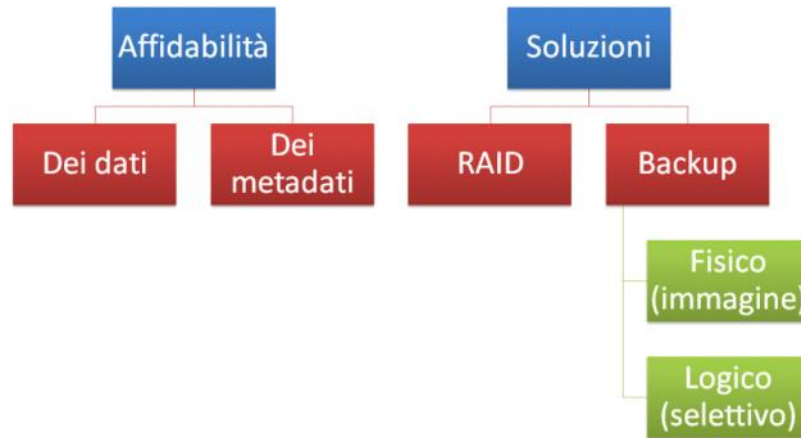
- Asincrona: ogni 20~30 secondi (ad esempio in Unix o Windows)
- Sincrona: immediatamente anche al disco (ad esempio Ms-Dos) (write-through)

- **Altre tecniche**

- **Read-ahead**
  - I blocchi vengono letti prima che siano richiesti (poco efficiente se accesso random)
- **Ridurre il movimento della testina**
  - Raggruppando i blocchi in gruppi (chiamati cluster), in modo da leggere interi cluster
  - Scrivere vicini i blocchi che verranno letti sequenzialmente
- **Collocare gli inode presso i rispettivi dati** (nello stesso cilindro?)



- **Affidabilità**



- **Tipologie**
  - Dei dati (perdere dati è costoso)
  - Dei metadati (perdere metadati spesso compromette l'integrità dei dati, irreparabile)
- **Soluzioni**
  - RAID
  - Backup
    - Fisico (veloce, ma difficilmente incrementale e non selettivo)
    - Logico (più selettivo, ma a volte troppo astratto)

- **Inconsistenza: soluzioni**

- Recuperare i dati sfruttando la ridondanza dei metadati (lenti e non sempre funzionali)
- Prevenire l'inconsistenza usando i file system journalled (JFS)

- **Journalled file system (JFS)**

- I metadati vengono scritti nel journal (o log: area a parte) prima che vengano modificati. Se avviene un crash è sufficiente ripercorrere il journal
- Adatti se serve una alta affidabilità e minimi tempi di recupero
- Esempi: XFS, JFS, ReiserFS, EXT3

# 13 - Il File System (*appunti generali*)

martedì 20 marzo 2007  
12.46

- **Tipi di file**
  - o Nella FAT (si assume il tipo dall'estensione)
  - o In Unix (non si assume il tipo dall'estensione)
  - o In MacOS (molti metadati, fra cui l'applicazione creatrice)
- **Struttura dei file**
  - o Imposta da SO
  - o Libera per l'utente
- **Operazioni sui file**



- **Possibili soluzioni ai problemi legati alle directory a grafo ciclico**
  - o Permettere la creazione di link ai soli file (soluzione di unix per i link hard)
  - o Permettere la creazione di link attraversabili (soluzione di unix per i link simbolici)
  - o Implementare un garbage collector che elimini le parti non raggiungibili dal SO (attenzione ai cicli!)
  - o Verificare la presenza di cicli alla creazione di un link [costoso]





# 14 - Esempi di File System

lunedì 19 marzo 2007

14.59

## - FAT12, FAT16, FAT32

- o La FAT viene caricata interamente in memoria (in Ms-Dos)
- o Il numero di blocchi si calcola elevando 2 al numero di bit:  $2^{32} = 512$  byte -> limite superiore = 2 TB

## - Directory in Windows 98

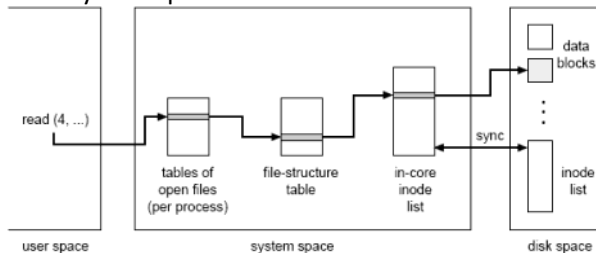
- o Supporto per i nomi lunghi, compatibile all'indietro: vengono concatenati i descrittori delle directory in modo che il primo indichi il nome e corto e gli altri il nome lungo

## - Virtual File System

- o Implementa una visione omogenea dei file system fisici
- o I file vengono identificati dal kernel tramite la coppia "ID logico del dispositivo" e "ID metadato"

## - Inodes:

- o Timestamp in formato epoch (seconda dal 01/01/1970 UTC, supporta fino al 19/01/2038 se 32 bit)
- o Tabelle del system space:



La tabella intermedia è necessaria per la condivisione dei file tra processi

- Ad un fork i figli ereditano una copia della tabella dei file aperti del padre

- o Hard Link: Entry differenti delle directory possono puntare allo stesso inode
- o Massimo 8 link simbolici (i link simbolici sono soft link, simili ai link di windows, solo che si possono riferire uni agli altri)

## - UFS: Unix File System

- o Esistono sia blocchi (di 4~6 KB) che i frammenti (di 0,5~1 KB): l'ultima parte dei file viene memorizzata in frammenti -> Meno frammentazione interna, aumenta la velocità di I/O
- o Implementa una cache di directory per aumentare l'efficienza di traduzione
- o Ogni cilindro del disco possiede un superblock, una tabella degli inode e i dati -> si riduce il tempo di seek

## - EXT2 (derivato da UFS)

- o Come UFS, ma ha blocchi solo della stessa dimensione (niente frammenti)
- o Il disco viene suddiviso in gruppi (di 9182 blocchi), ma senza rispettare la geometria del disco
- o Il superblock memorizza tipo di FS, 1° inode, n. Gruppi, n. Blocchi e inode liberi (abbiamo superblock copie quindi)

## - NTFS

- o Il file è visto come oggetto costituito da attributi. Ogni attributo è uno stream di byte
- o Indirizzamento a 64 bit
- o Nomi dei file in Unicode di lunghezza massima di 255 caratteri
- o E' possibile raggruppare partizioni logiche in un volume logico
- o Meccanismo transazionale per i metadati (logging) (semberebbe journalled quindi)
- o Allocazione a cluster
- o **MFT**
  - File di sistema con record di 1 KB descrittivi un file o directory
  - Posizione indifferente nel disco e dimensione dinamica
  - Il primo blocco è indicato nel boot block
  - Le prime 16 entry descrivono l'MFT stesso e il volume (come il superblock in unix)
  - **File residenti e non-residenti**
    - I valori degli attributi possono essere scritti direttamente nell'MFT (resident attribute, 'short file') o memorizzati in un blocco separato (non-resident attribute, 'long file'). Anche l'attributo data (per i dati) può essere residente.
  - **Directory**
    - Directory corte
      - ◆ Implementate come una lista in un record MFT
    - Directory lunghe

◆ Implementate come un file non-resident, strutturati a B-tree

- Partizionamento del disco
  - Usi diversi
  - Se uno non funziona più...
  - Limitare la dimensione
  - Aumentare la velocità di boot e di backup

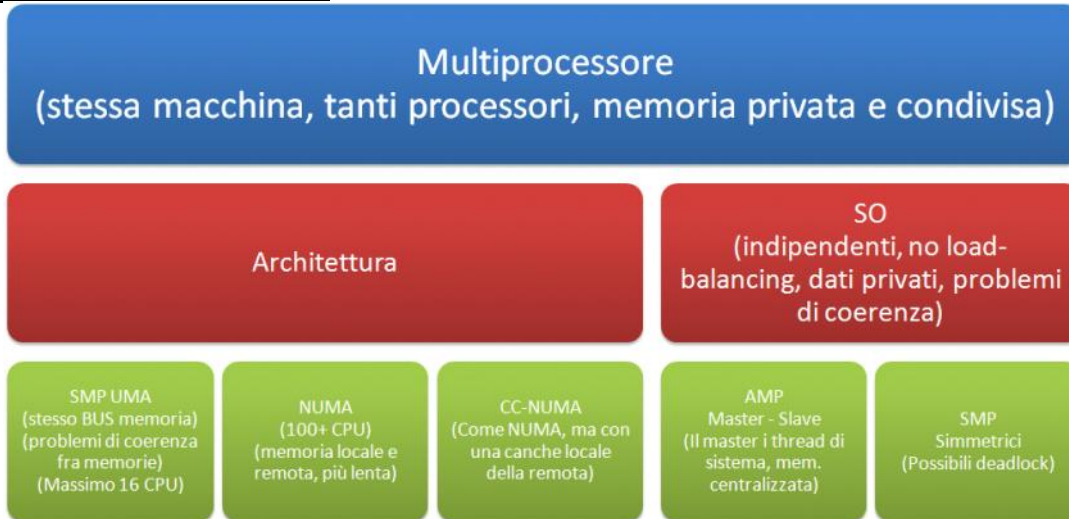
# 15 - Sistemi a più processori

domenica 18 marzo 2007  
15.55

Vari modi (cambia costo, scalabilità, complessità)

## - Multiprocessore

Sistemi strettamente accoppiati

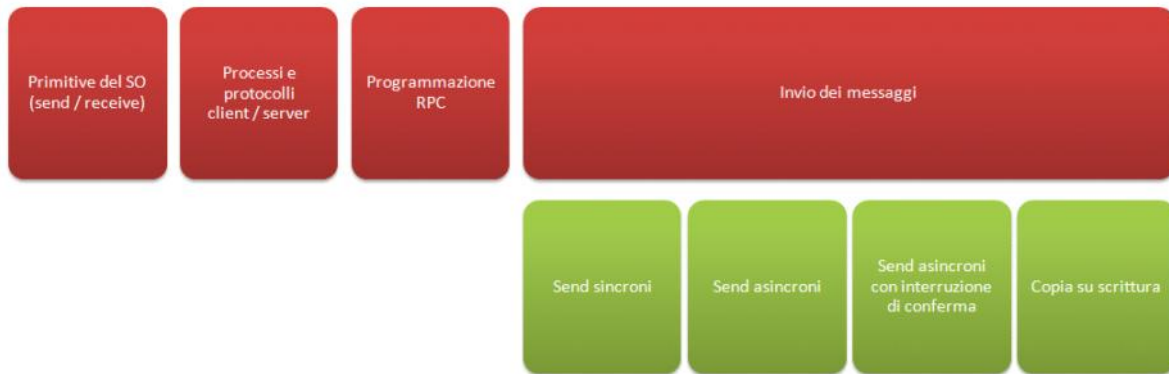


- Caratteristiche
  - Tanti processori con memoria privata e condivisa, sulla stessa macchina
- Tecnologie
  - **SMP UMA**
    - Stesso bus di accesso alla memoria: stesso tempo di accesso per tutta la memoria
      - ◆ Uso di cache
    - Problemi di coerenza a causa della cache e delle memorie private
    - Massimo 16 CPU. Se superiori, bisogna adottare la NUMA
  - **NUMA**
    - Supporta 100+ CPU
    - Singolo spazio di accesso alla memoria, ma che si divide in:
      - ◆ Locale (per ogni CPU o gruppo di CPU)
      - ◆ Remota (condivisa fra le CPU, 2-15 volte più lenta della locale)
  - **CC-NUMA**: Come NUMA, ma ha una cache locale della memoria remota
- **Sistemi operativi**
  - Caratteristiche
    - Ciascuna CPU ha un suo SO, con dati privati
    - Non c'è load-balancing o condivisione della memoria
    - Problemi di coerenza fra i SO (per esempio la cache su disco)
  - Tipi
    - **AMP**: Master-Slave
      - ◆ 1 processore esegue processi e thread di sistema, gli altri quelli utente
      - ◆ Il Master alloca i processi sugli slave e bilancia il carico
      - ◆ La memoria è centralizzata
      - ◆ Problema: il Master crea un collo di bottiglia a causa del OS-kernel
    - **SMP**: Simmetrici
      - ◆ Tutte le CPU sono simmetriche
      - ◆ Problema: il SO distribuito crea colli di bottiglia (sulla memoria?) -> si può suddividere il kernel in sezioni indipendenti. -> Possibili deadlock

## - Multicomputer

Sistemi debolmente accoppiati

# Multicomputer (diverse macchine, senza memoria condivisa)



- Caratteristiche
  - Computer separati, senza memoria condivisa, che comunicano (ad es. Ethernet: clusters)
  - Esistono vari modi di cablare questi computer. Molto usato è l'ipercubo ( $2p = \log_2$  nodi)
  - Le interfacce di rete hanno RAM privata e spesso una CPU
  - Problema: per comunicare fra tutti i computer vengono fatte parecchie copie dei dati: il metodo è lento -> Il DMA può ridurre il problema
- Comunicazione
  - Il SO offre delle primitive (quali send e receive) per la comunicazione
  - Devono essere implementati processi client/server e i loro protocolli di comunicazione
  - Il programmatore vede la decentralizzazione del processore
  - **Modalità di invio dei messaggi:**
    - Send sincroni (migliore se usiamo thread)
    - Send asincroni; il messaggio viene copiato nel buffer del sistema (spreco CPU)
    - Send asincroni; con interruzione di conferma
    - Copia su scrittura; il buffer viene copiato quando viene modificato
- Programmazione RPC (Chiamate di procedure remote)
  - Nasconde in parte all'utente la decentralizzazione dell'elaborazione: eseguire codice in remoto in modo simile a quello locale (elimina i send/receive all'utente)
  - **RMI**: Remote Method Invocation permette una RPC su oggetti
  - Funzionamento: La chiamata viene invocata da locale, i dati vengono impacchettati e inviati al server RPC, alla ricezione vengono spaccettati ed eseguiti
  - Problemi: potenziali problemi con l'uso dei puntatori, delle procedure polimorfe, delle variabili globali (che sarebbero globali a chi?)

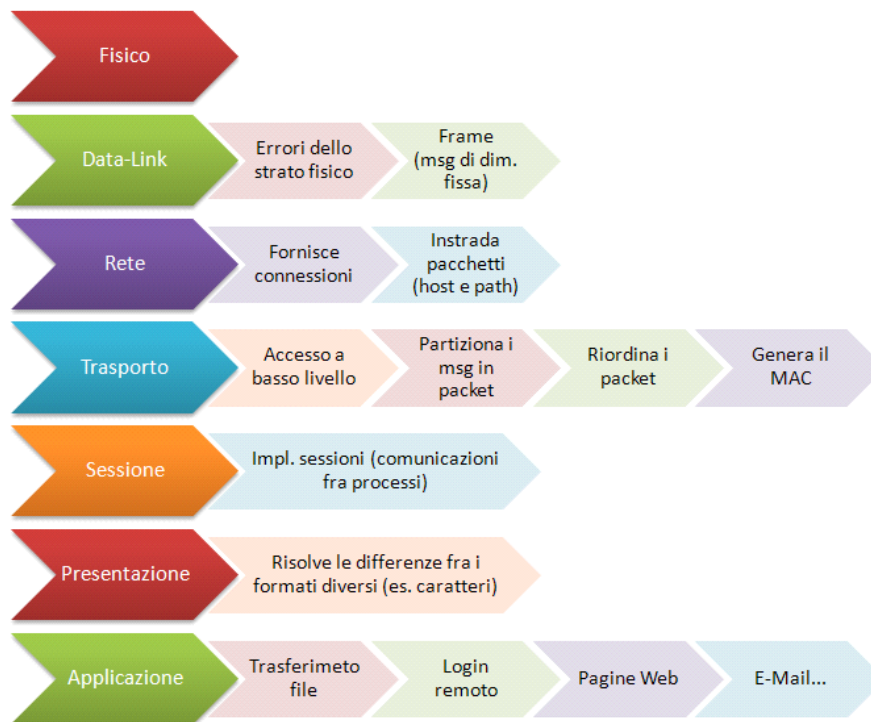
## - Sistemi distribuiti

Insieme di unità di elaborazione debolmente accoppiate (ad esempio reti locali, WAN, ecc...)

- Caratteristiche
  - Sono composti da server, client e protocollo. I ruoli di client e server possono scambiarsi
- Esempi di utilizzo
  - Condivisione risorse (stampanti, database, dispositivi hardware)
  - Load-balancing
- Hardware utilizzato: LAN (100 Mbit) e WAN (34 Mbit)
- Programmazione:
  - Complessa perché non c'è un modello uniforme -> Le applicazioni reimplementano le stesse funzionalità
  - ★ Servizi del sistema operativo:
    - Servizi di rete: funzionalità per la comunicazione (molto usati recentemente) (ad esempio l'interfaccia fornita dai socket)
    - Servizi distribuiti: modelli comuni di comunicazione (ad esempio remote file system) (poco usati recentemente) - formano il middleware (ad esempio il file system remoto)

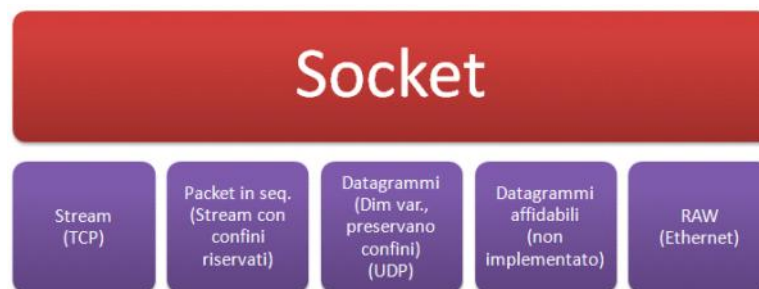
## - Servizi di rete

- Gli utenti devono accedere direttamente all'altra macchina (es. Telnet, http, ecc...)
- Tipologie:
  - Servizi con connessione - senza connessione
  - Servizi affidabili - non affidabili (verifica della ricezione, dell'ordine, della non-duplicazione dei pacchetti) L'affidabilità dei pacchetti è a discapito dell'overhead (a volte inutile per flussi di dati non necessariamente perfetti)
- **Stack / Suite / Strati: (modello ISO/OSI)**



○ **Socket**

- Rappresentano un endpoint di comunicazione
- Bounded ad un indirizzo.
- **I domini** regolano il formato degli indirizzi (AF\_INET, AF\_INET6, AF\_IPX, AF\_APPLETALK, ...)
- **Tipologie**



- **Servizi distribuiti**

○ **Caratteristiche** da implementare nei sistemi distribuiti

- Trasparenza: l'utente non dovrebbe poter distinguere fra i due né fra risorse locali e remote
- Mobilità dell'utente: l'utente dovrebbe poter vedere lo stesso ambiente da qualunque postazione
- **Tolleranza ai guasti**: i sistemi dovrebbero continuare a funzionare anche in seguito a guasti
  - Individuare failures (ad esempio di link)
    - ◆ Viene usato l'hand-shaking
  - Riconfigurare il sistema per poter proseguire
    - ◆ Una volta rilevato il guasto, l'informazione viene diramata. (ad esempio le tabelle di routing vengono adattate o servizi smettono di far richieste alla risorsa guasta)
  - Recuperare lo stato precedente dopo la riparazione
    - ◆ Deve venir rilevato un nuovo funzionamento, e questa informazione deve venir diramata. (ecc...)
- Scalabilità: si dovrebbero poter aggiungere nodi per aumentare i carichi sostenibili
- Sistemi su larga scala: il carico per ogni componente deve essere indipendente dal numero di nodi
- Struttura dei server: efficienti anche con grandi richieste (tanti processi / thread quindi)

○ **Servizi implementati dal middleware (formato dai servizi distribuiti)**

- Migrazione di dati: non so dove sono i dati
  - Basato su documenti (es. WWW, Lotus Notes)
  - Basato su file system distribuiti
- Migrazione delle computazioni: non so dove vengono eseguite le istruzioni
  - RPC/RMI (vengono implementati dati/oggetti astratti accessibili dai client RPC)
  - CORBA (l'utente vede oggetti condivisi, senza sapere dove essi sono localizzati)
    - ◆ Cross-platform
    - ◆ Ogni oggetto è localizzato su un solo server (limitata scalabilità)
  - Globe
    - ◆ Come CORBA, ma gli oggetti possono essere localizzati su più server
- Migrazione dei processi: non so dove vengono eseguiti i processi (esempio MSOIX, MS Wolfpack)

- Gli scheduler mantengono un carico omogeneo
- Coordinazione distribuita: non so dove sono i dati consumabili
  - Uno spazio condiviso formato da liste di scalari dove posso eseguire operazioni elementari (inserimento, cancellazione, lettura, creazione di un processo)
  - Difficili da implementare in modo realmente distribuito: di solito si usa un server centrale
- Si può usare l'RPC anche qui (ad esempio per un file system distribuito)
  - La porta su cui l'RPC lavora può essere prefissata o dinamica (tramite rendezvous: servizio del SO)
- Esempio: NFS: file system trasparenti che si montano sul una dir locale. (stateless, problemi di coerenza dovuti alla cache, lock demandato a lockd e non a nfs)

# 16 - Sicurezza (*appunti generali*)

martedì 20 marzo 2007

13.41

- **Sicurezza**
  - Controllo degli accessi
  - Controllo sulla modifica / cancellazione
  - Perdita di dati (incendi)
- **Autenticazione**
  - Password
    - Hashing della password (MD5, SHA)
    - Salting
    - One-time password (per le comunicazioni, o per i servizi bancari)
  - Challenge-response (l'algoritmo di cifratura cambia in base a determinate condizioni)
  - Schede
    - Schede magnetiche (~140 byte)
    - Chip card (circuiti integrati)
      - Stored value card (~1 KB)
      - Smart Card (CPU a 8 bit a 4MHz, ROM 16KB, EEPROM 4KB, RAM 512 byte, Transfer rate 9600 bps)
  - Biometrica (richiede sempre un login)
- **Attacchi**
  - **Interni**
    - Trojan horse
    - Trap door
    - Buffer overflow
    - "Bombe logiche"
  - **Esterni**
    - Worms
    - Virus (il periodo di diffusione senza attacco si chiama payload)
      - Tipi
        - ◆ Companion
          - ◇ si basa sul fatto che l'esecuzione con estensione implicita cerca prima i .com e poi i .exe: è sufficiente creare un .com; oppure modificando i link
          - ◇ Non infetta
        - ◆ Su eseguibili
          - ◇ Overwriting virus
          - ◇ Parasitic virus
        - ◆ Residenti in memoria
        - ◆ Del settore di boot
        - ◆ Del device driver
        - ◆ Macro
        - ◆ Source code
      - Tecniche
        - ◆ Compressione del codice
        - ◆ Criptazione del codice
        - ◆ Inserimento di istruzioni NOP o ADD0
        - ◆ Mutation engine (modificano l'ordine delle istruzioni senza modificare l'effetto)
- **Antivirus**
  - Firma
  - Checksum
  - Controllo delle system call sospette
- **Crittografia**
  - Data encryption standard
  - Crittografia a chiave pubblica
  - RSA
- **Codice mobile**
  - Viene eseguito in modo sicuro usando le seguenti tecniche:

- Sandbox (1 per il codice (bloccata), 1 per i dati; inoltre vengono controllati i riferimenti alla memoria in modo da isolare l'applicazione, sia nell'accesso ai dati, sia nell'esecuzione)
- Interpretazione (ogni istruzione viene controllata prima di essere eseguita, ad esempio Java nei browser)
- Firma