Appunti di Sistemi Operativi

Panoramica del calcolatore

- Architettura di un calcolatore

I componenti fondamentali di un calcolatore sono la **memoria centrale**, la **CPU** e il modulo di **input/output**.

- Esecuzione di un istruzione

L'esecuzione di un istruzione è composta di tre fasi: **fetch**, **decode** e **execute**, nella prima fase l'istruzione viene caricata dalla memoria, successivamente viene decodificata e infine viene eseguita.

- Classificazione di un istruzione

Le istruzioni possono essere classificate a seconda della loro funzione:

- **Processor-Memory**: Trasferiscono dati dal processore alla memoria o viceversa
- **Processor I/O**: Trasferiscono dati tra dispositivi di I/O e processore
- **Data Processing**: Elaborano dati con operazioni aritmetico-logiche
- **Control**: Modificano la sequenza di esecuzione delle istruzioni

- Chiamata a procedura

Le chiamate a procedura sfruttano una zona di memoria chiamata **stack** dove vengono allocati in modo ordinato (LIFO) tutti i dati riguardanti le stesse. Lo stack è indirizzato da tre puntatori contenuti in registri della CPU, questi sono: **Stack Base** che indica la base dello stack, **Stack Limit** che indica il limite superiore dello stack e **Stack Pointer** che indica l'attuale ultimo elemento dello stack.

- I/O Programmato

L'I/O programmato è una tecnica di gestione dei dispositivi di input/output estremamente semplice che può però gravare pesantemente sul sistema. Consiste nel verificare ciclicamente ad istanti di tempo predefiniti lo stato dei dispositivi, tramite un ciclo detto **busy waiting**, e servire i dispositivi che lo richiedono.

- Interrupt

La maggior parte dei dispositivi di I/O sono molto più lenti del processore, questo causerebbe, in caso di I\O programmato, tempi di inattività molto lunghi per la CPU, per questo si ricorre agli interrupt, dei segnali inviati alla CPU dai dispositivi quando questi ultimi necessitano di essere serviti.
Gli interrupt possono essere divisi in quattro classi:

- **Programma**: generati da qualche condizione particolare che si viene a verificare nell'esecuzione di un programma, come ad esempio una divisione per 0 o il riferimento ad una zona di memoria inaccessibile.
- **Timer**: generati dal timer di sistema, sono fondamentali per diverse funzioni di sistema.
- **I/O**: generati dal controllore di I/O di sistema per segnalare il completamento di un operazione o il verificarsi di un errore.
- **Hardware Failure**: Generati da un malfunzionamento di sistema, come ad esempio un abbassamento di potenza.

L'**interrupt handler** è la procedura del sistema operativo che viene eseguita quando si riscontra un interrupt, ha il compito di scoprire la fonte dell'interrupt e di gestirlo di conseguenza, al termine della sua esecuzione viene ripristinato lo stato precedente del sistema. E' da evidenziare che un interrupt non spezza mai l'esecuzione dell'istruzione attualmente in elaborazione da parte del processore.
In caso di interrupt multipli le strategie da poter applicare sono diverse, è possibile semplicemente disabilitare gli interrupt durante l'esecuzione dell'interrupt handler ed eseguire in seguito gli interrupt in attesa, questa politica è perfezionabile aggiungendo un sistema di priorità, in modo che un interrupt può essere interrotto solamente da interrupt con priorità più alta.

- Direct Memory Access (DMA)

La tecnica del DMA sfrutta un unità apposita detta DMA controller allo scopo di fornire accesso diretto alla memoria per le periferiche di I/O, in questo modo, durante i trasferimenti di dati, il processore può continuare il suo lavoro venendo interrotto solamente alla fine dell'operazione.

- Disk cache

Nei sistemi odierni un porzione della memoria di sistema viene utilizzata come buffer per i dati letti da disco rigido, in questo modo se si rende necessario un nuovo accesso agli stessi dati dopo un breve lasso di tempo, questi sono leggibili dalla RAM con un conseguente notevole risparmio di tempo.

Panoramica dei sistemi operativi

- Sistema operativo

Il sistema operativo è un programma che controlla l'esecuzione degli altri programmi applicativi, un interfaccia tra questi ultimi e l'hardware, fornendo tutte le funzioni di base necessarie per l'utilizzo del calcolatore.
Gli obbiettivi di un sistema operativo sono di rendere utilizzabile un calcolatore nel modo più semplice, e allo stesso tempo efficiente, possibile ed inoltre di consentire lo sviluppo di nuovi software e funzionalità dello stesso senza che queste attività interrompano i servizi basilari di sistema.
Il sistema operativo può anche essere visto come un gestore delle risorse necessarie all'esecuzione di un applicazione, in particolare la memoria di sistema, il tempo della CPU o i dispositivi di I/O.
I servizi che il sistema operativi fornisce agli utenti sono:

- Esecuzione di programmi
- Sicurezza
- Supporto per lo sviluppo di applicazioni
- Accounting
- Individuazione e notifica degli errori

Mentre i servizi che vengono offerti alle applicazioni sono fondamentalmente due: gestione delle risorse e accesso ai dispositivi di I/O.

- Kernel
Il kernel è una porzione del sistema operativo (viene anche definito come il **nucleo** di quest'ultimo) che viene caricato nella memoria centrale, è costituito da tutte le funzioni fondamentali e più frequentemente utilizzate dell'SO.

- User e Kernel mode
I processi possono essere eseguiti in due modalità differenti, i programmi eseguiti dall'utente sono solitamente lanciati in user mode, che non consente l'utilizzo di alcune istruzioni privilegiate e limita la zona di memoria accessibile, il kernel invece, viene eseguito appunto in modalità kernel (anche detta "**supervisor mode**") in modo da poter eseguire qualunque genere di istruzioni e da poter accedere a tutta la memoria. Lo scambio tra queste due modalità di funzionamento è detto **mode switch** e può essere generato da un interrupt o da una chiamata di sistema, a seconda del caso.

- Processi I/O bound e CPU bound
Possono essere classificati come I/O bound quei processi che fanno un uso intensivo, o comunque molto maggiore rispetto all'utilizzo di CPU, dei dispositivi di I/O, viceversa, i processi che richiedono una grande quantità di elaborazioni da parte della CPU sono detti processi CPU bound e non eseguono system call bloccanti (come ad esempio richieste di I/O).

- Time Sharing
Come abbiamo visto, dedicare la CPU ad un singolo processo che fa uso di I/O, comporta un incredibile spreco di tempo di calcolo, per questo nasce il Time Sharing, cioè un sistema che permette l'utilizzo del processore, ad intervalli di tempo prestabiliti, a più applicazioni o più utenti diversi.

- Memoria virtuale
La memoria virtuale è una funzione caratteristica dei sistemi operativi recenti che permette di indirizzare "logicamente" una quantità di memoria molto superiore a quella fisicamente disponibile, questo viene realizzato spostando su una zona apposita della memoria secondaria il contenuto della memoria centrale meno frequentemente utilizzato.
La memoria virtuale è gestita da un unità apposita chiamata **MMU (Memory Management Unit)** che lavora in collaborazione con il kernel.

- Paging
Il paging è un'altra funzione dei sistemi operativi che consente l'indirizzamento da parte del processore di una quantità di memoria maggiore di quella effettivamente possibile (limitata dal numero di bit del bus indirizzi della CPU). Nel paging la memoria centrale viene divisa in blocchi, detti appunto pagine, di dimensioni identiche tra loro, per riferirsi ad una certa locazione di memoria verrà quindi usato il numero di pagina in cui questa è contenuta, sommato all'offset che la distanzia dall'inizio della pagina.

- Struttura di un "sistema" calcolatore
Il calcolatore è schematizzabile come un sistema composto di diversi livelli o moduli, ognuno dei quali aggiunge funzionalità sfruttando i servizi offerti dal livello sottostante, in particolare una schematizzazione possibile può essere:

Livelli hardware
- Livello 1: circuiti elettronici; opera su registri, celle di memoria e porte logiche; operazioni possibili sono la cancellazione di un registro o l'accesso ad una cella di memoria.
- Livello 2: set di istruzioni del processore; effettua operazioni quali somma, sottrazione o memorizzazione.
- Livello 3: aggiunge il concetto di procedura e/o funzione, implementando le funzioni call e return.
- Livello 4: aggiunge gli interrupt.

Livelli sulla multiprogrammazione e gestione della memoria
- Livello 5: gestione dei processi da parte della CPU, aggiunge le operazioni di sospensione e ripristino.
- Livello 6: dispositivi di immagazzinamento secondari, implementa il trasferimento di blocchi di dati.
- Livello 7: crea uno spazio di indirizzamento logico per i processi e organizza lo spazio virtuale in blocchi.

Livelli sulla comunicazione tra i processi e l'I/O

- Livello 8: comunicazione delle informazioni tra i processi.
- Livello 9: supporta la memorizzazione di lungo termine dei file.
- Livello 10: fornisce accesso a dispositivi esterni utilizzando interfacce standard.
- Livello 11: mantiene le associazioni tra identificatori interni ed esterni.
- Livello 12: fornisce supporto ai processi.
- Livello 13: fornisce un interfaccia per l'utente (shell).

- Caratteristiche dei sistemi operativi recenti
I sistemi operativi odierni sono estremamente complessi e possono presentare strutture e funzioni molto particolari, ad esempio:

- sistemi a **microkernel**: il kernel di questi sistemi operativi è, appunto, estremamente ridotto, incorporando solamente le funzioni basilari, tutte le altre vengono implementate come processi.
- sistemi **multithreading**: consentono la suddivisione di un processo in diversi thread che possono essere eseguiti concorrentemente.
- **symmetric multiprocessing**: funzionalità che permette l'utilizzo di più processori, che condividono memoria centrale e dispositivi di I/O, allo stesso momento.

I processi

- Processo
Un insieme di istruzioni (programma) in fase di esecuzione, ogni processo può essere costituito da diversi thread (ma almeno da uno). Un processo è caratterizzato da diverse proprietà: regioni di memoria, file aperti, …

- Thread
Un thread è l'unità costitutiva di un processo che viene eseguita dal processore, è caratterizzato dal valore del PC e dei registri di CPU.

- Interazione tra il sistema operativo e i processi
Il sistema operativo ha il compito di coordinare l'esecuzione di diversi processi massimizzando l'utilizzo della CPU e di fornire loro le risorse necessarie.

- Creazione dei processi
Alla creazione di un nuovo processo, a quest'ultimo, vengono assegnati un identificatore univoco (PID) e lo spazio di memoria, inoltre viene inizializzato il PCB e viene aggiunto alla coda di schedulazione.
Un processo può essere creato in diverse circostanze e per cause diverse, in particolare:

- Nuovo processo batch
- Un nuovo utente effettua un login
- l'SO deve fornire un servizio
- Un processo ne genera uno nuovo

- Terminazione dei processi
Un processo può terminare per diverse cause:

- Il processo è terminato, effettua quindi una chiamata di servizio per indicarlo al sistema
- Superamento del tempo limite, il processo è vissuto oltre il tempo limite concessogli
- Memoria insufficiente, il processo richiede una quantità di memoria maggiore di quella a disposizione del sistema
- Violazione dei confini, il processo cerca di accedere a zone di memoria protette
- Errore di protezione, il processo cerca di utilizzare una risorsa che non gli è concessa (ad es. un file protetto)
- Errore aritmetico, il processo cerca di eseguire operazioni proibite, quali la divisione per 0
- Il processo attende per una quantità di tempo eccessiva l'avvenimento di un evento
- Fallimento di I/O, si verifica un errore durante l'input o l'output richiesto dal processo
- Istruzione non valida, il processo cerca di eseguire un istruzione non esistente
- Istruzione proibita, il processo cerca di eseguire un istruzione riservata al sistema operativo
- Uso non corretto dei dati, il processo utilizza un dato del tipo sbagliato o non inizializzato
- Intervento dell'operatore o del SO, il sistema termina per una qualche ragione il processo
- Terminazione del padre, se il padre del processo termina potrebbe essere richiesta anche la terminazione del processo stesso
- Terminazione da parte del padre, il padre del processo richiede la terminazione del processo figlio

- Scheduler e dipatcher
Lo scheduler è un componente del kernel che ha lo scopo di stabilire il prossimo processo che la CPU deve eseguire, il dispatcher invece ha il compito di settare i registri della CPU per l'esecuzione del processo, in pratica di eseguire il ripristino del contesto.

L'operazione di dispatching, cioè di cambiamento del processo in esecuzione, per fare ciò deve modificare i valori delle code di sistema e aggiornare i PCB dei processi coinvolti deve essere eseguita in kernel mode e richiede dunque due mode switch, per passare da user a kernel mode e viceversa.

- Stato dei processi

I processi si possono trovare in diversi stati:

- **Esecuzione**: il processo è attualmente in esecuzione, possiamo avere un processo in esecuzione per volta (più di uno solo in caso di sistemi multi-core).
- **Pronto**: il processo è pronto per continuare la sua esecuzione ma sta aspettando di poter utilizzare il processore.
- **Bloccato**: il processo è in attesa di un operazione di input o output da un dispositivo esterno e non può quindi continuare la sua esecuzione.

Inoltre possono essere definiti altri due stati riferiti ai processi **sospesi**, cioè "swappati" su disco per liberare memoria centrale, un processo può essere sospeso in stato pronto e in stato bloccato.

La sospensione di un processo può avere diverse cause:

- Swapping: la quantità di spazio in memoria non è sufficiente per i processi in esecuzione, il sistema operativo è costretto a effettuare uno swap su disco.
- Decisione del SO: il sistema può decidere di sospendere un processo se lo ritiene causa di problemi o malfunzionamenti.
- Decisione dell'utente: l'utente può decidere di sospendere un processo per debugging.
- Timing: un processo può essere eseguito a intervalli di tempo predefiniti e sospeso nel frattempo.
- Richiesta da parte di un processo genitore: un processo genitore può richiedere la sospensione di un processo figlio.

- Time Slice

Quantità di tempo contigua dedicata ad un processo dalla CPU

- Process Table

La process table contiene un entry per ogni processo, solitamente collegata al PCB del processo stesso e un limitato numero di informazioni per mantenerla più leggera possibile.

- Esecuzione dell'OS

Anche il sistema operativo è, come qualsiasi altro processo, un insieme di istruzioni da eseguire, le tipologie di OS, esaminandoli sotto questo aspetto sono:

- **Non-process kernel**: il sistema operativo non è un processo ed ha il suo spazio di memoria dedicato, questo tipo di implementazione è obsoleto ed inefficiente, in quanto richiede la riconfigurazione delle tabelle di memoria ad ogni mode switch.
- **Execution within user processes**: le funzioni del sistema operativo sono "aggiunte" ai processi dell'utente, in questo modo non è necessario alcun cambiamento alle tabelle di memoria ed è sufficiente un mode switch per eseguire le funzioni del sistema. In questa implementazione alle informazioni riguardanti il processo contenute in memoria vengono aggiunti lo stack, i dati e le istruzioni del kernel. Tutti i sistemi UNIX utilizzano questo approccio.
- **Process based OS (microkernel)**: in questo caso il sistema è implementato come un insieme di processi, ad eccezione naturalmente delle funzioni di process-switching. Questa soluzione è in genere più modulare, robusta e flessibile, ma sicuramente meno efficiente dell'execution within user processes, inoltre è la soluzione adottata dai sistemi operativi windows.

21-10-2008

Gestione della memoria

- Gestione della memoria

La memoria è una delle risorse più importanti che il SO deve gestire. La gestione della memoria consiste nel conoscere le locazioni di memoria usate e quelle libere per poter deallocare le prime (ad esempio garbage collector) permettendo il riutilizzo della memoria e allocare le seconde quando richiesto. Se non si deallocassero le locazioni di memoria non necessarie avremmo in tempi brevi una saturazione della memoria (**Memory Leak**). La memoria di sistema è utilizzata soprattutto per contenere strutture dati quali quelle del kernel o quelle dei processi ed inoltre contiene i processi in esecuzione stessi.

Quando un processo richiede memoria al SO quest'ultimo gli fornisce 3Gb di memoria "virtuale" divisa in quanttro sezioni: **PCB**, **Stack**, **Heap** e **Program**:

- PCB (Process Control Block): Contiene tutte le informazioni riguardanti il processo (PID, stato, priorità, contesto, puntatori, informazioni di I/O, utente possessore, processo genitore, ...), viene creato e gestito dal sistema operativo.
- Program: Il codice del programma (sequenza di istruzioni) viene salvato in memoria centrale nello spazio Program.
- Heap (mucchio): Quando un programma alloca dati con l'istruzione malloc, questi ultimi vengono allocati nell'heap, una zona della memoria dedicata al programma in cui i dati vengono memorizzati disordinatamente (da

questo il nome: mucchio).
- Stack (pila): Zona di memoria ordinata e organizzata come una pila utilizzata soprattutto per le chiamate di funzioni o procedure all'interno di un programma.

- Fixed Partitioning
Una tecnica di gestione della memoria di base che divide la memoria in partizioni di dimensione fissa che può essere la stessa per ogni partizione o diversa per ottimizzarne l'utilizzo (come ad esempio negli Hard Disk), questa tecnica è però abbastanza inefficiente in quanto comporta un considerevole spreco di memoria dovuto alla **frammentazione interna** cioè quel fenomeno per il quale assegnando una partizione sovradimensionata alla memorizzazione di un dato, tutto lo spazio in eccesso presente non è utilizzabile.

- Dinamic Partitioning
Tecnica in cui il partizionamento della memoria varia nel tempo, eliminando così il problema della frammentazione interna (spazio libero rimanente in un blocco di memoria allocato), ma generando un problema di **frammentazione esterna** (rimane spazio troppo piccolo per essere utilizzato tra due blocchi), per questo deve essere compattato periodicamente per ottenere lo spazio di memoria vuoto contiguo più grande possibile. Il problema è che la compattazione della memoria non è sempre possibile (la malloc non effettua compattazione ad esempio).
Per decidere quale spazio allocare ad un processo si possono utilizzare diverse strategie:
- **Best fit**: Si decide di utilizzare la zona di memoria di dimensione più vicina a quella necessaria al processo, quest'algoritmo comporta però una generazione maggiore di spazio in utilizzabile (spazi troppo piccoli)
- **First fit**: Alloca la prima zona di memoria disponibile in cui il programma entra, è sicuramente la tecnica più veloce comporta però una gestione molto poco omogenea della memoria.
- **Next fit**: Alloca ciclicamente, dopo aver allocato un blocco tiene il puntatore alla zona allocata, iniziando a cercare da questo punto quando dovrà allocare un nuovo blocco.

- Malloc
Funzione di libreria C basata su una System call (non è una System call!!! Ma è basata tipicamente sulle System call "brk" o "mmap") dedicata all'allocazione della memoria. La malloc divide la memoria in blocchi di taglie diverse (crescenti esponenzialmente es. 8kb,16kb,32kb,...) mantenendo in delle liste i dati che le riguardano, non appena viene richiesta un allocazione assegna al programma richiedente uno dei blocchi della dimensione migliore (necessari 50kb alloco un blocco da 64kb). I sistemi operativi non hanno a disposizione la malloc, in quanto fà parte delle API C.

24-10-2008
- Buddy System
Un semplice sistema per l'allocazione della memoria largamente utilizzato dai SO, viene solitamente utilizzato come allocatore di base per altri allocatori più sofisticati (il buddy system alloca un blocco di memoria di grandi dimensioni, che viene poi sfruttato da altri allocatori). Il buddy system mantiene un insieme di liste contenenti l'insieme di celle disponibili, in particolare abbiamo una lista per ogni dimensione (a salti di potenze di due). Quando viene richiesta una quantità q di memoria al buddy system questo arrotonda per eccesso q alla potenza di due più vicina e consulta le liste delle celle disponibili, se è disponibile una cella di dimensione desiderata la alloca direttamente altrimenti prende una cella di taglia superiore e la divide in due "buddyes" (splitting), dei quali uno viene allocato. E' possibile anche il processo inverso (coaleshing, fusione), questo può avvenire ogniqualvolta una zona di memoria viene deallocata, oppure solamente quando è necessario costituire un blocco più grande, questo a seconda dell'approccio adottato, aggressivo nel primo caso e lazy nel secondo.
NB: Non è vero che due blocchi contigui sono sempre buddyes, sono buddyes solo quando provengono dalla stessa divisione.

es.
- Abbiamo inizialmente solo un blocco da 1 Mb
- Ci vengono richiesti 100kb, ne vogliamo allocare quindi 128 (potenza di due più vicina per eccesso)
- Abbiamo blocchi da 128? No, allora da 256? No, da 512? no, da 1M? Si
- Dividiamo allora il blocco da 1 in due buddyes da 512
- Dividiamo i due blocchi da 512 in 2 da 256
- Dividiamo i due da 256 in due da 128
- Allochiamo i 128K.

Attenzione tutti i dati del buddy system sono contenuti in liste

Procedure get_hole
input: i (taglia del blocco richiesto)
output: un blocco c di taglia 2^i

if (Li is empty)

```
            b = get_hole(i+1)
            splitta b
            metti b1 e b2 in Li
        // A questo punto abbiamo per forza almeno un elemento in Li
        c = first hole in Li
        remove c from Li
        return c
```

- Rilocazione dei processi
I processi devono poter essere rilocabili, a tempo d'esecuzione infatti potrebbero essere spostati su disco e in seguito ricaricati in memoria in una posizione differente, inoltre il programmatore non può sapere in quale zona di memoria verrà allocato il processo una volta caricato in memoria.

- Protezione dei processi
La zona di memoria assegnata ad un processo deve necessariamente essere protetta dall'accesso di altri processi che potrebbero causarne la terminazione. La protezione della memoria deve essere estremamente veloce, viene quindi spesso realizzata via hardware.

- Sharing
Alcune applicazioni necessitano una zona di memoria condivisa con altri processi per la condivisione dei dati.

- Organizzazione Logica
I programmi (ad eccezzione di programmi estremamente semplici) devono essere modulari, questo comporta la possibilità di sviluppo da parte di più gruppi in contemporanea, software engineering più semplice e condivisione dei moduli tra i processi.

- Paginazione
Partiziona la memoria in blocchi (solitamente da 4Kb l'uno) detti frame, quando un frame è occupato da un processo è detto **pagina** di quest'ultimo. Il SO tiene una tabella delle pagine per ogni processo. La paginazione della memoria risolve il problema della frammentazione esterna, in quanto ogni pagina di un processo può essere allocata ovunque in memoria senza perdite di prestazioni. Questa tecnica richiede l'utilizzo di un dispositivo aggiuntivo, la **Memory Management Unit (MMU)** che si occupa di trasformare gli indirizzi interni al processo (n° di pagina) a indirizzi effettivi in memoria (indirizzo del frame).

- Segmentazione
I segmenti sono blocchi di memoria sequenziali abbastanza grandi, di lunghezze diverse allocati a dei processi, la differenza con la paginazione consiste nell'accesso alla cella di memoria, nella segmentazione infatti si fornisce il numero di segmento e l'offset della cella a cui si vuole accedere prendendo come base l'indirizzo iniziale del segmento. Non tutti i processori e i sistemi operativi supportano la segmentazione.

28-10-2008

- Esercizi sui processi
1- Quante volte verrà eseguito il dispatcher prima che tutti i processi terminino l'esecuzione?

Dati:
Abbiamo N processi CPU bound
Ciascun processo necessita per concludere l'esecuzione di un secondo di CPU (t)
Nel sistema non ci sono altri processi
Gli N processi sono in stato di pronto
Il time slice per tutti è di 3 s

R: N volte

2- Quante volte verrà eseguito il dispatcher prima che tutti i processi terminino l'esecuzione?

Dati:
Problema 1 con time slice (ts) di 100ms per ogni processo

R: (t x N) / ts -> (1000xN)/100 = 10 N volte

3- Quanti sono i process switch? E i mode switch?

Dati:
Problema 2

R: 10N process switch in quanto ogni volta che il dispacher viene richiamato cambia processo. 20N volte il mode switch in quanto per passare da processo a dispatcher si ha un cambiamento da user a kernel mode e viceversa per il passaggio da dispatcher a processo.

4- Conta mode e process switch.

Dati:
Due processi nel sistema, P1 e P2
P1 fà N chiamate di sistema bloccanti
P2 non fà alcuna chiamata di sistema
La CPU viene restituita a P1 non appena P1 va in ready

R: 2N process switch in quanto ne abbiamo due per ogni system call. Di conseguenza i mode switch sono 4N (due per ogni process switch).

5- Conta mode e process switch, mancano eventualmente dei dati per rispondere?

Dati:
Problema 4 con modello process based. (sistema operativo costituito da processi).

R:

31-10-2008

-Esercizi sul buddy system e sul paging

1- Dobbiamo allocare un blocco A da 64KB, quante volte viene richiamata la procedura get_hole?
Dati:
Abbiamo uno spazio di memoria pari a 1MB con nessun blocco allocato
Risp:
1MB/2 = 512KB --> 512KB/2 = 256KB --> 256KB/2 = 128KB --> 128KB/2 = 64KB <-- OK
Il numero di chiamate di get_hole è quindi pari a 5.

2- Trova un metodo basato sulla rappresentazione binaria per stabilire se due blocchi della stessa grandezza b1 e b2 sono due buddies
Dati:
I blocchi sono identificati con l'indirizzo del primo byte.
Risp:
Se due blocchi sono buddies la loro rappresentazione binaria deve essere identica fino al bit meno significativo.

3- Trova un algoritmo per la deallocazione free_block ricorsivo per aggiornare le liste Li al seguito del rilascio di un blocco.
Dati:
Indirizzo di b = b.addr
Taglia di b = 2^b.i
Due buddies vengono uniti appena possibile
Risp:

```
free_block(int b.addr,int i){
        int b1.addr = get_buddy(b.addr)
        if(b1.addr <> null){
                remove(b1.addr,Li)
                if(b.addr<b1.addr)
                        free_block(b.addr,i+1)
                else
                        free_block(b1.addr,i+1)
        }else
                insert(b.addr,Li)
}
```

4- Quanti bit uso per l'offset? Quanti frame abbiamo in memoria fisica? Se un processo usa tutte le pagine quanto è grande ogni singola entry della page table?
Dati:
32 bit per indirizzi fisici e logici

Frame di 4 KB
Memoria fisica a disposizione massima (4GB)
Risp: Uso 12 bit per l'offset. Abbiamo un milione di frame. Ogni entry è grande 4 byte.

1-12-2008

Memoria virtuale

- Memoria dal punto di vista del processo

Il processo vede la memoria divisa in regioni, una per il programma (codice e librerie), e una per i dati (heap e stack), queste regioni sono caratterizzate da diritti di lettura e/o scrittura; l'accesso a queste regioni avviene tramite istruzioni macchina.
Esempio, linux a 32bit
Dimensione memoria: 4Gb
Zone di memoria:
- Kernel (proibita)
- Stack (lett,scritt)
- Code (lett)
- Init data (lett,scritt)
- Heap (lett,scritt)
- ...

- Memoria virtuale

La memoria virtuale ci porta numerosi vantaggi, in primo luogo consente l'esecuzione di più processi contemporaneamente inoltre consente l'esecuzione di programmi anche più grandi della memoria centrale stessa.

- Page fault

Un interrupt particolare lanciato quando un programma cerca di accedere ad una pagina non presente in memoria centrale, che deve quindi essere recuperata dal disco, il processo che ha fatto richiesta quindi viene messo in blocco perchè in attesa di un input. I page fault possono essere di due tipi, quello appena descritto è detto **major page fault**, ed è il più diffuso, ma potremmo avere anche un **minor page fault** dove l'input dal disco non è richiesto (ad esempio per un allocazione).

- Thrashing

Quando il numero (e la dimensione) dei processi è troppo elevato per la memoria a disposizione si verifica il fenomeno di thrashing, cioè lo swap su disco di processi che invece dovrebbero essere mantenuti in memoria centrale.

- Fetch policy

Meccanismo che cerca di determinare quando una pagina deve essere caricata in memoria, in particolare il **prepaging** carica più pagine di quante richieste, in questo modo se vengono richieste le pagine successive a quella richiesta queste sono già presenti in memoria, il prepaging non è molto usato in quanto spesso comporta una grande quantità di lavoro sprecata, il **demand paging** invece si limita a caricare in memoria la pagina richiesta.

- Placement policy

Determina dove fisicamente deve essere allocato un segmento o una pagina.

- Eviction policy

Meccanismo dell'SO che sceglie le pagine da cancellare dalla memoria centrale, un buon meccanismo di eviction cancella pagine che non verrano riutilizzate nel prossimo futuro.

- Cleaning policy

Meccanismo dell'SO che si occupa di scrivere su disco le pagine di memoria modificate (dirty) in memoria centrale. Anche in questo caso abbiamo due strategie possibili, il **precleaning**, in cui vengono sfruttati tutti i momenti di pausa del disco per fare cleaning, e il **demand cleaning** in cui una pagina viene "pulita" solo quando viene richiesto (cancellazione imminente).

- Page buffering

Il sistema operativo mantiene due liste di pagine: una per le pagine "pulite" ed una per le pagine "sporche", quando è necessaria l'allocazione di una pagina, quest'ultima viene allocata al posto di una pagina pulita.

- Load control

Il load control determina il numero massimo di processi che possono essere eseguiti, cercando di evitare quindi il thrashing continuo ed allo stesso tempo sfruttare a pieno la CPU. Se sul sistema vengono eseguiti troppi processi il load control sospende uno o più di questi ultimi spostando tutte le pagine che li riguardano su disco e stoppando il processo. I processi da sospendere possono essere scelti con strategie differenti come ad esempio la priorità o l'ultimo processo attivato.

- Disk caching

Il disk caching ha gli stessi obbiettivi della memoria virtuale ma agisce sui file, utilizzando la memoria centrale come una

cache del disco.

- Memory mapped files
Un punto di incontro tra disk caching e memoria virtuale sono i memory mapped files, in cui appunto un processo può richiedere che un file, o parte di esso venga caricato in una data zona di memoria. La mappatura in memoria di un file viene richiesta tramite una system call (**mmap**) la quale si occupa direttamente però solo di allocare una zona di memoria, infatti il caricamento vero e proprio viene realizzato quando si tenta di accedere per la prima volta al file e viene quindi generato un page fault, questo per consentire il caricamento da memoria più veloce possibile (passando attraverso la CPU). Gli svantaggi del memory mapped files sono che i file vengono scritti su disco solamente durante il cleaning della pagina e soprattutto che non consente cambiamenti delle dimensioni dei file. I memory mapped files sono solitamente utilizzati per il caricamento di librerie o per applicazioni quali i DBMS.

- Supporto hardware per la memoria virtuale
La memoria virtuale è in parte implementata via hardware, in particolare la paginazione (traduzione degli indirizzi virtuali in indirizzi fisici) deve essere realizzata a basso livello, in particolare dalla **MMU (Memory Management Unit)** contenuta all'interno della CPU.

- Tabella delle pagine gerarchica
Per affrontare le dimensioni spropositate (4Mb) della tabella delle pagine completa, viene realizzata un mini tabella da 4Kb che punta alle entry della tabella completa, in questo modo l'indirizzo fisico sarà composto da: *indEntry + indFrame + Offset* in questo modo abbiamo una paginazione a due livelli. Questo meccanismo comporta però tre accessi in memoria con relativa perdita di prestazioni, si cerca di ovviare a questo problema con la **TLB**, **translation lookaside buffer** una memoria cache per la page table che contiene le ultime entry della tabella utilizzate.

- Inverted page table (IPT)
Questo meccanismo alternativo non fà utilizzo delle tabelle delle pagine, basandosi sull'assunzione che la maggior parte delle entry della tabella delle pagine non viene utilizzata, ed è quindi conveniente memorizzare solamente le entry utilizzate, questo comporta però la necessità di ricercare, dato il PID del processo e il numero di pagina, il frame voluto, questo viene realizzato tramite hashtable. Il maggior difetto della IPT è la complessità di gestione di una hashtable che richiede quindi un implementazione hardware di costo non indifferente. Questa scelta è stata applicata in PowerPC, UltraSPARC, Intel Itanium ed è inoltre molto utilizzato nelle architetture a 64bit per evitare le enormi tabelle che comporterebbe la tabella delle pagine classica.
Nell'applicazione reale il valore di hash è proprio il numero del frame, questo comporta l'eliminazione di una colonna nella hashtable.

- SAS (Singol Address Space) e MAS (Multiple Address Space)
Nei classici sistemi a 32bit, come abbiamo già visto, ogni processo ha la propria tabella delle pagine (MAS), nei sistemi a 64bit invece, data l'enorme quantità di spazio indirizzabile, tutti i processi condividono la stessa tabella delle pagine (SAS), questo comporta il non dover più "switchare" le tabelle delle pagine dei vari processi.

- Translation Lookaside Buffer (TLB)
La TLB è semplicemente una cache dedicata alla tabella delle pagine in cui si cerca di caricare le entry della tabella maggiormente usate tramite meccanismi di semi-predizione analoghi a quelli utilizzati dalla cache del processore.
Lo svuotamento della TLB rappresenta il costo vero e proprio del process switch nei sistemi MAS.

- Dimensionamento delle pagine
E' fondamentale scegliere una dimensione ottimale per le pagine di memoria, tenendo presente che pagine troppo piccole, permettono si, di ridurre la frammentazione della memoria, ma generano una tabella delle pagine enorme, viceversa per pagine di dimensioni troppo grandi.

- Segmentazione
La segmentazione divide lo spazio interno della memoria di un processo in, appunto, segmenti che possono anche essere di dimensioni diverse, esiste anche una tabella dei segmenti per ogni processo che, esattamente come per la tabella delle pagine contiene un entry per ogni segmento. Nonostante alcuni vantaggi quali: facilità di condivisione tra processi, possibilità di ricompilare indipendentemente ogni modulo di un programma o l'implementazione di meccanismi di protezione, la segmentazione è obsoleta e quindi non utilizzata.

- Resident Set Management (RSS)
La gestione del resident set può essere realizzata in due modi diversi, **globale** o **locale**. Gli algoritmi di gestione del resident set si basano sul **principio di località** che a sua volta è divisa tra la località spaziale, data dall'accesso a zone di memoria vicine tra loro, e la località temporale, data dall'accesso alle stesse zone di memoria in un intervallo ristretto di tempo. La dimostrazione del principio di località ci fa capire che è possibile tentare una "predizione del futuro" basandosi sul comportamento passato del sistema. Una gestione del RSS locale consiste, al momento di una richiesta di allocazione, nel

dis-allocare una pagina appartenente allo stesso processo che ha richiesto la nuova allocazione, in una gestione globale invece la pagina da dis-allocare viene scelta tra tutte le pagine presenti in memoria con criteri diversi. In particolare una gestione del resident set globale, rispetto a quella locale, è più facilmente implementabile e proprio per questo è la più utilizzata dai sistemi operativi.

- Replacement policy
Le replacement policy hanno lo scopo di stabilire quale pagina togliere dalla memoria in favore di una nuova pagina, possiamo adottare diverse strategie, tra le quali:

- Optimal policy:
La optimal policy è in realta una strategia utopistica di gestione del RSS, in quanto richiede una conoscenza completa degli eventi futuri, è quindi utilizzata solamente come termine di paragone. Consiste nel togliere dalla memoria la pagina che verrà richiamata più in là nel tempo, questo risulta naturalmente in una gestione ottimale e nel minor numero di page faults possibile.

- Least Recently Used (LRU):
Un metodo di gestione di rimpiazzamento della memoria che incarna il principio di località temporale, l'LRU può essere implementato in due modi o ad ogni pagina di memoria viene applicato un time-stamp che indica l'ultimo istante in cui è stata acceduta, o viene gestita una coda di pagine in cui se una di queste ultime viene acceduta è spostata in fondo alla coda. L'LRU non è implementato a causa dell'eccessiva complessità di gestione in entrambe i casi di realizzazione visto che ad ogni accesso è necessaria una modifica dello stamp o della coda.

- First-In, First-Out (FIFO):
Politica di gestione della cancellazione della memoria, è la più semplice da implementare ma anche la meno efficiente, ed è basata sulla gestione di un buffer circolare (una coda), non basata sul tempo di accesso alla pagina, ma sul tempo di caricamento in memoria.

- Clock policy (Second chance):
Questa politica di gestione della memoria aggiunge un bit addizionale ad ogni pagina che indica se la pagina è stata acceduta. Quando viene richiesto un rimpiazzamento la clock policy effettua una scansione di tutte le pagine, settando tutti i bit da 1 a 0 ed utilizzando per il rimpiazzamento la prima pagina il cui bit è già pari a 0. La clock policy è una molto grossolana approssimazione di LRU. Esiste una versione più evoluta della clock policy basata sulla constatazione che è conveniente togliere dalla memoria le pagine non modificate (e che quindi non richiedono cleaning), viene quindi utilizzato un secondo bit che indica se le pagine vengono modificate, le pagine che verranno prima di tutto eliminate saranno quindi quelle non usate e non modificate.

- Aging policy:
Nell'aging policy, oltre al bit di uso, vengono mantenuti 8 bit per ogni pagina in una struttura separata, la lista delle pagine viene periodicamente scansionata e ad ogni "passata" il bit di uso viene aggiunto in testa (con un right shift) agli 8bit corrispondenti alla sua pagina. Questo consente di avere un informazione più precisa sull'utilizzo di una pagina, la pagina che viene eliminata è quella con gli 8bit di valore più basso. Questa strategia risulta poco efficiente per memorie di grandi dimensioni, questo perchè è necessaria una scansione di tutta la memoria ad ogni "sweep".

Queste strategie vengono implementate nella porzione di kernel chiamata **pager** o **swapper**.
Si fa notare inoltre che è possibile “bloccare” determinate pagine (ad esempio quelle del kernel) per evitare che vengano tolte dalla memoria centrale settando un bit, chiamato appunto **lock-bit**.

- Working set
Insieme delle pagine accedute negli ultimi n istanti, programmi molto locali avranno ws molto piccoli e viceversa per i programmi poco locali. Se il ws è più piccolo dell'rs si verificherà un numero di page faults piuttosto elevati, se invece il ws è più grande dell'rs i page faults saranno nettamente minori.

- PFF policy
La **page fault frequency** policy ha lo scopo di gestire la dimensione del resident set di un processo mantenendola ad un valore ottimale, è basata sull’analisi del numero di page fault generati da un processo e consiste nell’aumentare l’RS del processo se il PFF è eccessivo, diminuirlo se è troppo ridotto.

- Anomalia di Belady
Belady afferma che non è necessariamente vero che aumentando il numero di frame disponibili diminuiscano i page fault.

Scheduling

- Scheduling
Lo scheduling può essere di tipi diversi:

- A **lungo termine**: determina se aggiungere/rimuovere o no un processo a/da quelli correntemente in esecuzione. Controlla quindi il livello della multiprogrammazione, con più processi avremmo meno tempo a disposizione per ognuno, e viceversa.
- A **medio termine** : determina tra i processi del pool in esecuzione quali sospendere e quali ripristinare. Controlla lo swap, anche qui regolando il livello di multiprogrammazione del sistema.
- A **breve termine (dispatcher)**: determina per ciascuna CPU quale tra i processi in stato di ready deve essere

eseguito e per quanto tempo, questa decisione è presa molto frequentemente, il dispatcher deve quindi essere molto efficiente e veloce. Può essere caratterizzato da due diverse **decision mode**: **preemptive** o **non preemptive**, nel primo caso i processi possono essere interrotti durante la loro esecuzione, mentre nel secondo caso non è così, permettendo quindi la monopolizzazione della CPU da parte di un processo, sistemi preemptive sono solitamente più efficienti.
- di I/O: determina quale richiesta di utilizzo di un dispositivo di I/O debba essere servita.

- Criteri di ottimalità per lo scheduler
Gli scheduler, in particolare quelle a breve termine, devono essere estremamente efficienti e veloci, alcuni dei criteri di cui bisogna tener conto progettando uno scheduler sono i seguenti:
- tempo di completamento di un processo, soprattutto in caso di programmi batch, deve essere il più ridotto possibile.
- tempo di risposta in caso di programmi interattivi, deve risultare il più ridotto possibile.
- il tempo di esecuzione di un processo deve essere pressappoco lo stesso, qualunque sia il carico del sistema.
- la quantità di processi terminati nell'unità di tempo deve essere la massima possibile.
- lo sfruttamento della CPU deve essere massimo.
- ogni processo dovrebbe essere trattato alla pari degli altri (in caso di assenza di priorità).
- rispetto delle priorità dei processi quando presenti.
- mantenere un livello di occupazione accettabile di tutte le risorse di sistema.
- in condizioni normali lo scheduler dovrebbe preferire processi I/O bound a processi CPU bound.

- Priorità
E' utile assegnare ad ogni processo una priorità in modo da ottimizzare lo scheduling dando la precedenza a processi con priorità maggiore, solitamente per ogni livello di priorità abbiamo una coda di stato ready.

- Politiche di scheduling
Uno scheduler può seguire politiche differenti, tra cui:
- **FCFS (First-Come First-Served)**: I processi vengono immagazzinati in un unica coda e serviti in ordine di arrivo, questa soluzione favorisce I processi CPU bound e penalizza quelli corti.
- **Round-robin (Time Slicing)**: Il round-robin è un meccanismo di scheduling basato su un clock generato ad intervalli regolari. Ad ogni interrupt (scatto del clock) il processo correntemente in esecuzione viene rimesso in coda e viene caricato un nuovo processo in stato ready. Anche questa scelta favorisce i processi CPU bound in quanto questi ultimi sono in grado di sfruttare tutto il quanto di tempo assegnatogli.
- **Shortest process next (SPN)**: Un metodo di scheduling che cerca di favorire l'esecuzione dei processi che si "pensa" richiedano meno tempo di processamento. E' presente anche una variante preemptive di questo meccanismo chiamata shortest remaining time in cui è appunto possibile interrompere un processo per farne eseguire un altro. In questo modo naturalmente vengono pesantemente penalizzati i processi molto lunghi.
- Feedback: Il meccanismo di feedback è basato su un insieme di code di priorità in cui i processi vengono collocati a seconda del loro comportamento durante l'ultima esecuzione, solitamente i processi appartenenti ad una certa coda di priorità allo scadere del quanto di tempo assegnatogli vengono accodati alla coda di priorità più bassa.

File Management
- File system
Il file system è la componente del SO dedita alla gestione dei file, è sostanzialmente costituita da una o più strutture dati che contengono informazioni riguardanti i blocchi liberi, quelli occupati e così via... I file system più utilizzati sono FAT, NTFS, ext2/3. I file system odierni sono "**journaled**", registrano cioè ogni spostamento o modifica, permettendo, in caso di malfunzionamenti, di evitare la perdita di dati in fase di trasferimento, questo però comporta un lavoro maggiore e quindi una perdita di efficienza.

- iNode
In unix un file è una sequenza di byte puntata da un inode (**Index Node**). Un inode è una struttura che contiene le informazioni principali di un file, quali: permessi, data di modifica, numero di link, lunghezza e soprattutto, posizione. Gli inode vengono utilizzati anche per la rappresentazione delle directory. Le locazioni puntate da un inode si dividono in dirette e indirette (di vario livello), l'utilizzo di locazioni indirizzate indirettamente comporta naturalmente una perdita di prestazioni dovuta all'inserimento di passaggi intermedi, vengono quindi utilizzate prima tutte le locazioni direttamente indirizzate, e poi se necessario (file di grandi dimenzioni) le indirette.

- Link
I link permettono l'accesso a un file o una directory da diverse posizioni, possono essere di due tipologie, hard e soft. Un **hard link** è un nome che fa riferimento ad un inode, se un inode non ha hardlink associati (e solo in questo caso) viene eliminato, mentre un **soft link** è un file che contiene un path (che fà riferimento ad un hard link) e possono essere validi o

non validi.

- Virtual filesystem
Il virtual file system di linux è un filesystem astratto con il quale si interfacciano, da un lato i filesystem reali, dall'altro il kernel, in questo modo il kernel non sa nulla dei file system a lui sottostanti.

Disk scheduling

- Lettura da disco
I dischi rigidi sono molto spesso i colli di bottiglia dei sistemi, infatti, essendo dispositivi in parte meccanici non rispondono alla velocità dei componenti elettronici, per effettuare una lettura infatti è necessario aspettare il tempo di posizionamento della testina ed il tempo di rotazione del disco per leggere la posizione corretta, ne risulta che il tempo di posizionamento è molto maggiore rispetto all'effettivo tempo di lettura del dato. Proprio per cercare di ottimizzare lo sfruttamento di questa risorsa, anche l'accesso a disco è controllato da uno scheduler.

- Disk scheduling
Lo scheduling (decisione della prossima richiesta da servire) delle operazioni richieste ad un disco (letture e scritture) deve quindi avere come obbiettivo la massimizzazione dell'output e allo stesso tempo il garantire che nessuna richiesta sia eccessivamente penalizzata, può essere realizzato in diversi modi:

- **First-In First-Out** (**FIFO**): la politica più semplice, serve le richieste in ordine di arrivo, senza alcun controllo.
- **Last-In Last-Out** (**LIFO**): l'ultima richiesta arrivata è la prima ad essere servita, politica molto semplice ma assolutamente inefficiente. Ha solamente interesse teorico.
- **Shortest Service Time** (**SST**): viene scelta la richiesta che richiede il minor movimento della testina, garantisce il througput maggiore ma purtroppo è caratterizzata da un alto livello di discriminazione tra i processi.
- **Elevator**: così chiamato perchè utilizzato dagli ascensori dei grattacieli, è anche noto come scan o look, consiste nello scorrere tutto il disco, prima in un verso poi in un altro, servendo tutte le richieste che corrispondono alla posizione attuale della testina, questo metodo offre un troughput abbastanza elevato ma genera problemi di unfairness, in quanto i processi agli estremi della traccia possono avere un tempo di attesa massimo molto superiore a quello dei processi centrali.
- **C Scan** (**One Way Elevator**): variante dell'elevator in cui la testina legge solo mentre si muove in una direzione (ritornando poi direttamente alla posizione iniziale quando giunge alla fine della traccia) riducendo i problemi di unfairness ma allo stesso tempo riducendo anche il troughput, che rimane comunque abbastanza elevato da rendere conveniente questo metodo.
- **Request merging**: nel request merging le richieste per blocchi adiacenti sono trattate come fossero una, risparmiando il tempo di posizionamento della testina per queste ultime. Questa tecnica viene applicata ai metodi di disk caching sopraelencati per migliorarne le prestazioni.

- Linux disk scheduling
Linux permette di scegliere fra differenti scheduler per il disco, in particolare i più conosciuti sono:

- **Noop**: implementazione del metodo FIFO a cui è applicato il request merging.
- **Deadline**: implementazione del C scan modificata in modo da risolvere problemi legati alla differenza di trattamento tra letture e scritture (le scritture vengono solitamente effettuate sequenzialmente, al contrario delle letture, sono quindi favorite da questo metodo) impostando un tempo massimo di attesa pari a 5s per le scritture e 500ms per le letture.
- **Anticipatory**: una versione evoluta della deadline che applica il request merging e che per risolvere il problema dell'unfairness lettura/scrittura dopo una read si mette in attesa fino a 6ms cercando si servire un altra read. E' da notare però come questa attesa non sia sempre effettuata, l'anticipatory implementa infatti degli algoritmi per "prevedere" il comportamento futuro del processo e stabilire come agire per ottimizzare i tempi.
- **Complete Fair Queueing (CFQ)**: il CFQ è un altra versione del C scan con request merging che introduce però supporto alle priorità e soprattutto un approccio del tipo round robin che divide il tempo a disposizione in slice assegnati ai processi, in ogni slice si eseguono richieste del processo corrispondente. Questo metodo è adatto a sistemi multiutente ed offre prestazioni simili all'anticipatory, inoltre garantisce, come indicato dal nome, una totale assenza di indiscriminazioni.

Lo switch tra i vari scheduler è possibile anche a runtime (a patto che il volume sia smontato) ed è inoltre possibile specificare scheduler diversi per diversi dischi.

RAID

- RAID
RAID (Redundant Array of Indipendent Disks), tecnologia che ha lo scopo di aumentare la performance e la tolleranza ai guasti dei dischi rigidi facendo in modo che il sistema operativo veda più dischi differenti come un unico disco. Quando un disco si guasta,la struttura continua a funzionare ma entra nello stato **degraded**, quando il disco guasto viene sostituito, viene avviato il processo di **rebuilding** per inserire nel nuovo disco i dati necessari per il corretto funzionamento. Alcuni sistemi possono essere dotati di uno o più dischi addizionali (**hot spare disks**), non utilizzati durante il normale funzionamento che entrano in funzione in caso di fault in modo da non generare disservizi.

- Tecniche e tipi di RAID

Le tecnologie applicate al RAID sono diverse, in particolare:

- **mirroring**: gli stessi dati sono duplicati su ogni disco, questo consente un elevato grado di sicurezza (numerose copie) e un grande vantaggio in fatto di letture, in quanto possono essere realizzate in parallelo, le richieste di scrittura invece non subiscono nessun miglioramento.
- **parity**: dei bit ridondanti vengono aggiunti ai dati a scopo di controllo e correzione di eventuali errori.
- **striping**: lo stripping consiste nella scrittura di dati logicamente consecutivi su dischi diversi.

Queste tecnologie vengono utilizzate e combinate in modi diversi dai diversi tipi di RAID:

- **RAID 0**, è una semplice applicazione dello striping, i dati vengono scritti su diversi dischi e possono essere spezzati in byte o blocchi, permette un incremento di prestazioni pari a N in lettura (dove N è il numero di dischi utilizzati), è estremamente semplice ma poco tollerante ai guasti, infatti in caso di fault di un disco i dati sono irrecuperabili e il servizio interrotto (non è possibile il rebuilding).
- **RAID 1**, applicazione del mirroring, i dati vengono duplicati su tutti i dischi utilizzati (solitamente non più di due), il tempo di lettura non subisce miglioramenti ma le prestazioni in lettura vengono migliorate di un fattore N (numero di dischi). In caso di dischi di dimensioni differenti la dimensione viene limitata a quella del disco più piccolo.
- **RAID 2**, applicazione dello striping a livello di bit, i dischi devono essere sincronizzati, alcuni dei dischi servono per l'immagazzinamento di dati necessari a processi di correzione dell'errore.
- **RAID 3**, striping a livello di byte, utilizza inoltre un disco per i bit di parità. Offre prestazioni in lettura pari al RAID 0, e grazie al controllo di parità una maggiore fault tollerance, d'altra parte le prestazioni in scrittura sono abbastanza scadenti in quanto il disco per i bit di parità è un collo di bottiglia.
- **RAID 4**, identico al raid 3 ma utilizza lo stripping a livello di blocco.
- **RAID 5**, identico al RAID 4 ma i bit di parità sono divisi equamente sui dischi, in questo modo abbiamo un disco in più per le letture e soprattutto miglioriamo le prestazioni in scrittura in quanto la parità è divisa tra diversi dischi.
- **RAID 6**, identico al RAID 5 ma con doppia parità su ogni disco, in questo modo aumenta la fault tolelrance.
- **JBOD** (**Just a Bunch Of Disks**), utilizza i dischi come se fossero una sequenza di blocchi consecutivi, non è propriamente un RAID.

Tra queste tipologie i più utilizzati sono 0,1 e 5, sono inoltre possibili combinazioni di RAID (10, 01, 51, ...) ma richiedono controller appositi molto costosi e sono quindi abbastanza rare.

#Pratica#

21-10-2008

- Terminale (tty)

Una postazione costituita da uno schermo e da una tastiera (nel caso fisico). Dal terminale è possibile lanciare dei processi, che a questo punto prenderà il controllo di quest'ultimo direzionandovi i suoi canali di output e raccogliendo i suoi canali di input. Solitamente un solo processo per volta può essere attaccato ad un terminale (in foreground), è possibile lanciare più processi da un terminale il modalità background, ma in questo modo ne l'output, ne l'input del programma saranno collegati al terminale.

- Canali dei Processi

Ogni processo ha diversi canali:

- Standard Input: Canale che convoglia l'input al processo.
- Signal: Un insieme di "bottoni" che controllano il processo:
  - Stop: il processo è congelato (Ctrl+z)
  - Term: il processo viene terminato "gentilmente" (Ctrl+c)
- Standard Output: Canale che convoglia l'output dal processo a destinazione.
- Standard Error: Canale che convoglia eventuali errori generati durante l'esecuzione del processo a destinazione.

NB: Ogni processo può essere attaccato o staccato da un terminale, se il processo è attaccato ad un terminale Standard input e Standard output sono inviati al terminale in questione.

- Organizzazione dei processi

I processi in esecuzione sono organizzati secondo relazioni di parentela, in particolare un processo viene obbligatoriamente lanciato da un altro processo (ad eccezione del primo processo) a questo punto il processo lanciato è figlio del processo che lo ha lanciato detto a sua volta padre. Se due processi sono stati lanciati dallo stesso processo padre, sono detti fratelli.

- Comandi utili:

- jobs: elenca i processi lanciati dalla shell visualizzandone lo stato.
- fg (foreground): ripristina l'esecuzione di un programma in foreground (non permette interazione con la shell).
- bg %NUMPROCESSO: ripristina l'esecuzione di un programma in background.
- ps aux: elenca tutti i processi in esecuzione.

- pstree: elenca tutti i processi in esecuzione strutturati in modo da evidenziare le relazioni di parentela.
- kill PID: uccide un processo in esecuzione, con l'opzione -9 si effettua un uccisione non gentile del processo.
- echo STRINGA: stampa STRINGA su standard output, utile per gli script.
- touch NOMEFILE: "tocca" il file indicato, se non esiste lo crea, se esiste ne modifica la data di accesso.

24-10-2008

- Redirigere i canali di I/O di un processo

I canali di un processo possono essere rediretti, ad esempio su un file tramite '<' e '>' (es. ls > z invierà l'output di ls ad un file di nome z). Questo consente anche la concatenazione di processi (pipelining) inviando l'output di uno all'input dell'altro, esistono svariati programmi che elaborano l'output di altri programmi e vengono detti fltri (es. less, tr, head, ...).

- Filtri utili:

- head: visualizza le prime 10 righe di un output, -n visualizza le prime n righe
- tail: visualizza le ultime 10 righe di un output, -n visualizza le ultime n righe
- less: visualizza un output una pagina alla volta
- wc: (Word Count) visualizza numero di righe, numero di parole e numero di caratteri di un output

28-10-2008

- Comandi utili per la concatenazione:

- cat FILE: legge un file da disco e invia in standard output il contenuto.
- COMANDO | COMANDO: pipe, congiunge output di un processo con input di un altro.
- tr C1 C2: traduce o elimina caratteri al'interno di un testo.
- FILE: mette in ordine alfabetico le righe di un testo, con l'opzione -n effettua un sort numerico, con l'opzione -r effettua il sort in ordine contrario.
- uniq: contrae tutte le righe consecutive uguali in una sola, con l'opzione -c le conta.
- cut -f NUMCAMPO -d SEPARATORECAMPI: effettua la proiezione di un output (eliminazione di una colonna).

4-11-2008/7-11-2008

- Awk

Un linguaggio di "micro"-programmazione utilizzabile da riga di comando. Utilizzato tramite il comando awk 'PATTERN {AZIONI}', analizza il testo una riga alla volta e, se corrisponde al pattern indicato, esegue le azioni indicate sulla riga in questione, se non viene specificato un pattern (indicata tramite un espressione regolare) awk esegue il codice per tutte le righe. NF (Number of fields), NR (Number of records), FS (Field Separator), OFS (Output Field Separator).

Opzioni:

- v : Consente di modificare una variabile che influenza il funzionamento del programma (ad esempio FS o OFS) o di crearne una da passare in input al programma.

Comandi:

- print $N: Stampa i valori del campo N, senza parametri stampa il valore del primo campo, con più parametri (separati da virgola) stampa tutti i campi indicati.
- if (CONDIZIONE) {ISTRUZIONI} else {ISTRUZIONI}: Costrutto if, in sintassi C/Java.
- printf "TIPO",$N: Print analoga al C.
- for (CONT;COND;INCR) {ISTRUZIONI}: Costrutto for.
- gsub (ESPRREG,"STRINGA",$N): Sostituisce l'espressione regolare nel campo N con la stringa indicata, se non viene indicato il campo agisce su tutta la riga.

es. cat /etc/passwd | awk -v FS=: '{print $3}'

- Grep

Comando che cerca in uno o più file tutte le righe che corrispondono a un pattern ('a', "abc",...) e le visualizza.

NB: per matchare i caratteri speciali dobbiamo accodarli a una backslash.

- Espressioni regolari

- Espansioni

L'esecuzione di un comando da parte della shell è preceduta da una serie di trasformazioni, o meglio espansioni, della stringa data in input, le indichiamo di seguito in sequenza ordinata:

1- Brace Expansion a{1,2,3} --> a1 a2 a3

2- Tilde Expansion ~kimo --> /home/kimo

3- Variable Expansion x=ciao, $x --> ciao

Alle variabili viene sostituito il loro effettivo valore.

3b- Aritmetic Expansion $(1+2) --> 3

Vengono valutate le espressioni aritmetiche e sostituite con il loro valore.

3c- Parameter Expansion

4- Command Substitution `` --> hostname

Viene elaborato il comando contenuto tra `` e sostituito dal suo otuput.

5- Word Splitting ciao sono io --> ciao sono io

Si occupa di separare (utilizzando lo spazio come separatore) la stringa in un array di stringhe.

6- Pathname Expansion x? --> x1 xa x+

Vengono inseriti in sequenza i nomi di tutti i file (nella dir.) che cominciano per il valore che precede '?'.

- Script

Un file di testo costituito da un insieme di comandi di terminale. Gli script, per convenzione, hanno come estensione '.sh' ed inoltre devono essere settati come eseguibili (vedi Chmod). Ad uno script è possibile passare dei parametri utilizzando la funzione $n che individua il parametro n-esimo, in particolare:

- $0 fornisce il nome con cui è stato eseguito il file (comprensivo di path).
- $# fornisce il numero dei parametri.
- $? fornisce indicazioni sullo status del processo in foreground lanciato più recentemente (0 indica corretto funzionamento), è utile per verificare se il comando precedentemente lanciato ha riscontrato problemi.

- Eseguibili

Esistono due tipi di file eseguibili:

- Binari: generati da un compilatore per essere eseguiti direttamente dal processore o da una macchina vituale.
- ASCII: file testuali che vengono interpretati da un interprete. Un file testuale viene riconosciuto come eseguibile se inizia con '#!' seguito dall'interprete da utilizzare (ad es /bin/sh). Il sistema operativo quindi esegue l'interprete con parametro il file stesso (contenente a questo punto i comandi da eseguire in sequenza). Ad esempio è possibile scrivere un programma awk su un file che ha come intestazione "#!/usr/bin/awk -f".

NB: Tutti i linguaggi interpretabili individuano i commenti con il carattere '#', questo è necessario, in quanto altrimenti gli interpreti non saprebbero come interpretare la prima riga che compone il file eseguibile.

- Comando chmod

Il comando chmod modifica le proprietà di un file, in particolare:

+x Rende eseguibile un file da tutti gli utenti.

-x Rende un file eseguibile non più eseguibile.

- Directory Proc

La directory /proc/ contiene un inseme di file che forniscono informazioni sul sistema, la cosa interessante e che questi file non sono fisici (letti da disco), ma sono generati al momento dal kernel. Possiamo trovare ad esempio informazioni sull hardware, sul buddy system, sulle system call o sui processi, infatti ogni processo ha una propria directory (in /proc/) che contiene una grandissima quantità di informazioni.

- Sistemi operativi 18/12/2008

#include

#define, permette di definire dei simboli, utilizzato per definire le costanti (PI = 3.14)

#ifdef

#endif, utilizzate dal precompilatore per effettuare selezioni sulle parti di codice da compilare (ad esempio nel kernel che si deve adattare a diverse architetture) questa tecnica è detta compilazione condizionale.

- Formato ELF

Formato per i file oggetto (.o) generati dall'assemlazione di un programma C, è un simil-linguaggio macchina che non include però tutte le librerie richieste dal programma.

- Evoluzione di un sorgente

Quando viene richiesta la compilazione di un file sorgente, solitamente indicato con l'estensione .c, viene richiamato un precompilatore (CPP) che genera appunto un precompilato, solo in seguito viene richiamato il compilatore (GCC) che trasforma il precompilato in un file in assembly, a questo punto l'assemblatore (AS) lo converte nel formato ELF generando un file oggetto (.o), infine interviene il linker (LD) che completa il file oggetto con tutte le librerie necessarie rendendolo effettivamente eseguibile.

- Makefile

Il makefile indica quali sorgenti compilare e come compilarli, nonchè il nome che vi deve assegnare, è composto da una serie di definizioni del tipo:

target: sorgente

relazione_target\sorgente

- GDB

Debugger C molto semplice che lavora a livello macchina e mette a disposizione i seguenti comandi:

- l, list: visualizza il sorgente del programma
- r,run: esegue il programma
- b NUMRIGA: imposta un punto di break
- n, next: continua l'esecuzione in caso di break
- p, print PARAMETRO: stampa una serie di informazioni utili quali ad esempio i parametri
- s, stack: restituisce l'istruzione affiorante dallo stack
- bt: visualizza tutto lo stack
- info br: visualizza tutti i break point
- dis NUMBRK: disabilita un break point
- cond NUMBRK: abilita un break point