Sistemi Operativi Cheats-sheet

kocierik

September 14, 2024

1 Concorrenza

interliving = Quando al massimo abbiamo un processo in esecuzione
overlapping = Quando abbiamo più processi in esecuzione
Context Switch = Operazione del sistema operativo che conserva lo stato del processo
Race Condition = Quando il risultato dell'esecuzione dipende dalla temporalizzazione con cui vengono eseguiti i processi
Due proprietà di un programma concorrente sono:

- Proprietà: **Safety**, i processi non devono **interferire** tra loro nell'accesso alle risorse condivise
- Proprietà: **Liveness**, i meccanismi di sincronizzazione non devono prevenire l'avanzamento del programma (un processo non deve attendere **indefinitamente**

Mutua Esclusione = Ad ogni istante al massimo un processo può accedere a una risorsa
DeadLock = Indica una situazione in cui due o più processi o azioni si bloccano a vicenda, aspettando che uno esegua
Starvation = Si intende l'impossibilità continua, da parte di un processo pronto all'esecuzione, di ottenere le risorse sia hardware sia software di cui necessita per essere eseguito
Requisiti **Critical section:**

- Mutua esclusione
- Assenza di deadLock
- Assenza di Dalay (la **CS** non deve aspettare l'ingresso di un'altra CS)
- Starvation

Busy Waiting = Controllo iterativo su una condizione di accesso
SpinLock = Tecnica consiste nel verificare periodicamente se il lock è stato sbloccato, effettuando un test (**Test And Set**)
Semafori:

- **V** viene inviato un segnale (rilascio risorsa)
- **P** viene invocata per attendere un segnale

Semaforo Mutex = Utilizzato per garantire mutua esclusione
Monitor = Un monitor è un costrutto di sincronizzazione.
Un'istanza di un tipo monitor può essere utilizzato per rendere mutuamente esclusivo l'accesso a risorse condivise.
Un processo entra in un monitor invocando una delle sue procedure. Solo un processo alla volta puo' essere dentro un monitor **CV** = Variabili di condizione.
Nelle quali sono definite operazioni di:

- *c.wait* attende il verificarsi della condizione
- *c.signal* segnala che la condizione e' vera, ponendo il chiamante in attesa e causando la riattivazione di un processo

Message Passing = paradigma di comunicazione tra processi. Con operazioni di:

- **send**, spedisce un messaggio a un processo destinatario
- **receive**, riceve un messaggio da un processo mittente

2 Architettura sistemi operativi

Un **processo** è un programma in esecuzione che utilizza risorse fonrnite dal computer
Un **file** è l'astrazione di un archivio di dati
Un **file system** è composto da un insieme di file e responsabile di:

- Creazione e cancellazione di file
- Creazione e cancellazione di directory
- Manipolazione di file e directory
- Codifica del file system su una sequenza di blocchi

La progettazione di un S.O deve tener conto di:

- Efficienza
- Manutenibilità
- Espansibilità
- Modularità

E' possibile suddividere i S.O in due grandi famiglie a seconda della loro struttura:

- Sistemi con struttura semplice, descritti tramite un insieme di procedure. Un programma sbagliato può mandare in crash l'intero sistema
- Sistemi con struttura a strati

Possiamo individurare 4 categorie di kernel:

1. **Kernel monolitici**, tutte le procedure vengono raggruppate e gestite mutualmente. E' presente una astrazione dell'HW (**efficienza e modularità**). Se una procedura **fallisce** l'intero sistema **crasha** (**Linux, UNIX, FreeBSD**)
2. **Micro kernel**, esso si deve occupare di funzionalità minime di gestione dei processi e della memoria, e **meccanismi di comunicazione** per permettere ai processi clienti di chiedere servizi ai processi serventi. Semplice astrazione HW coordinate da un kernel minimale. Basato sulle primitive di Message Passing (**memoria condivisa**). Questa tipologia è **espandibile, portabile, robusta, sicura** ma **inefficiente**
3. **Kernel ibridi**, essi mantengono una parte di codice nello spazio utente per ragioni di maggiore efficienza di esecuzione. Anch'essi usano message passing
4. **ExoKernel**, non forniscono astrazione dell'HW, ma forniscono librerie che mettono a contatto diretto le applicaizoni con l'HW.

Nelle **macchine virtuali**, dove non si va a creare l'illusione di molteplici processi che posseggono la propria CPU e la propria memoria

3 Richiami hardware

Interrupt = meccanismo che permette l'interruzione del normale ciclo di esecuzione della CPU. Inseguito ad un interrupt abbiamo una gestione:

- Hardware, dove il processore sospende le operazioni del processo corrente, e salta ad un particolare indirizzo di memoria (**iterrupt handler**)
- Software, dove viene gestito l'interrupt handler e ritorna il controllo al processo interrotto

E' possibile avere interrupt multipli. Sono presenti due approcci:

- Disabilitazione degli interrupt, ulteriori segnali vengono ignorati ma restano pendenti. Viene attivato inseguito l'interrupt handler
- Interrupt annidati, dove è possibile interromperne uno di priorità inferiore

La comunicazione tra un processore e dispositivo può avvenire in due modalità:

- **Programmed I/O**, dove la CPU carica gli input nei registri attraversio i bus. Il dispositivo attendereà il risultato (**busy-waiting-polling**)

- **Interrupt-Driven I/O**, la CPU carica gli input nei registri ed inseguito il S.O sospende l'esecuzione del processo che ha eseguito l'operazione di input ed esegue un altro processo. Una volta terminato viene segnalato attraverso un interrupt e così la CPU copia i dati dal buffer locale. Output molto simile. In questo modo il processore spreca parte del suo tempo per trasferire dati e inoltre la velocità di traferimento è limitata

La **Direct Memory Access** (DMA), attiva l'operazione di I/O specificando l'indirizzo in memoria di destinazione o di provenienza dei dati
La **Random Access Memory** (RAM), assieme ai registrti è l'unico spazio di memorizzazione che può essere acceduto direttamente dal processore (**volatile**). Nei sistemi moderni l'accesso avviene tramite **MMU** (trasforma gli indirizzi logici in fisici)
La **Read Only Memory** (ROM), memoria accessibile solo in lettura
In un disco le operazioni di **seek** corrispondono allo spostamento del pettine di testa sul disco. E' una operazione costosa e deve essere limitata. A differenza delle SSD **non** hanno un numero di cicli di scrittura limitato
Un meccanismo di **caching** consiste nel memorizzare **parzialmente** i dati di una memoria in una seconda più costosa ma più efficiente. Maggiori richiesta aumentano prestazioni. Il caching può essere:

- Hardware (cache CPU) dove le politiche sono **non modificabili dal S.O**
- Software (cache disco) dove le politiche sono **sotto controllo dal S.O**

I sistemi multiprogrammati e multiutente richiedono la presenza di **meccanismi di protezione**. Sono presenti diverse modalità gestite dal **mode bit**:

- **Modalità Kernel**, dove i processi hanno accesso a tutte le istruzioni incluse quelle privilegiate
- **Modalità Utente**, dove i processi non hanno accesso alle istruzioni privilegiate

Il processo di caricamento del sistema operativo è chiamato **bootstrap**. Tutte le volte che avviene un interrupt, l'hardware passa in modalità kernel. E' necessario anche una protezione della memoria, altrimenti i processi potrebbero modificare codice del S.O o processi.
La protezione avviene tramite la **Memory Management Unit** (MMU). Poichè le istruzioni di I/O sono privilegiate possono essere eseguite solo dal S.O. I processi utente devono fare richieste al S.O tramite **syscall**.

4 Scheduler

Un **Process Control Block** (PCB), è formato da:

- Codice da eseguire
- Dati su cui operare

- Uno stack di lavoro per la gestione di chiamate
- Un insieme di attributi contenenti le informazioni per gestire il processo

Per gestire i processi viene utilizzata una tabella contenente:

- I descrittori dei processi (PCB)
- Ogni processo ha un PCB associato

E' possibile suddividere le informazioni contenute nel descrittore in 3 aree:

- Informazione di identificazione di processo
- Informazione di stato del processo
- Informazioni di controllo del processo

Gli stati dei processi possono essere:

- **Running**, il processo è in esecuzione
- **Waiting**, il processo è in attesa di qualche evento esterno
- **Ready**, il processo può essere eseguito, ma attualmente il processore è impegnato in altre attività

Un singolo processo non può eseguire due differenti attività contemporaneamente. Utilizzando i thread possiamo invece eseguire un diverso insieme di istruzioni. Ogni thread possiede:

- La propria copia dello stato del processore
- Il proprio program counter
- Uno stack separato

I thread condividono lo spazio di memoria e le risorse allocate degli altri thread dello stesso processo. Un sistema operativo può implementare i thread in due modi:

- **User thread** (livello utente)
- **Kernel thread** (livello kernel)

Gli **User thread**, vengono supportati sopra il kernel e vengono implementati da una **thread library** a livello utente (implementazione efficiente ma se il kernel è single-threaded qualsiasi user thread che effettua una chiamata di sistema bloccante causa il blocco dell'intero processo)
I **Kernel thread**, vengono supportati direttamente dal sistema operativo. Essendo implementate a livello kernel se un thread esegue una operazione di I/O, esso può selezionare un altro thread. Questo causa un risultato più lento.

Si vengono a creare 3 differenti modelli di multithreading:

- Many-to-One, un certo numero di user thread vengono mappati su un solo kernel thread
- One-to-One, ogni user thread viene mappato su un kernel thread
- Many-to-Many, un certo numero di user thread vengono mappati su più kernel thread

Per rappresentare uno **schedule** si usano i diagrammi di Gantt. Gli eventi che possono causare un **context switch** sono:

- Quando un processo passa da stato running a stato waiting
- Quando un processo passa dallo stato running a ready
- Quando un processo passa dallo stato waiting allo stato ready
- Quando un processo termina

Uno scheduler si dice:

- **non-preemptive** se il controllo della risorsa viene trasferito solo se l'assegnatario attuale lo cede volontariamente
- **preemptive** se il controllo della risorsa venga tolto all'assegnatario attuale a causa di un evento

Il **throughput** è il numero di processo completati per unità di tempo. Il tempo di **turnaround** è il tempo che intercorre dalla entrata di un processo alla sua terminazione. La **CPU burst** sono le attività svolte dalla CPU. Mentre quelle **I/O burst** sono i periodi di attesa.

I principali algoritmi di scheduling sono:

- **FIFO** (First In First Out), il processo che arriva per primo, viene servito per primo
- **SJF** (Shortest Job First), questo algoritmo è **ottimale** rispetto al tempo di attesa, ma è **impossibile** da implementare perché non possiamo conoscere la lunghezza del prossimo CPU burst (processo). Esistono due versioni:
  1. **Non preemptive**, dove il processo corrente esegue fino al completamento del suo CPU burst
  2. **Preemptive**, dove il processo corrente può essere messo nella coda ready, se arriva un processo con CPU burst più breve
- **Round-Robin**, un processo non può rimanere in esecuzione per un tempo superiore alla durata del quanto di tempo
- **A priorità**, ogni processo è associato a una specifica priorità definita dal **SO** oppure **esternamente**. La priorità può essere anche **statica** dove non cambia durante la vita di un processo e **dinamica**

5 Risorse

Un sistema di elaborazione è composto da un insieme di risorse da assegnare ai processi presenti. Le risorse possono essere suddivise in classi. Se due risorse appartengono alla stessa classe sono equivalenti.
Le risorse di una classe vengono dette **istanze**. Il numero di risorse **molteplicità**.
Un processo può richiedere solo **una** risorsa di una specifica classe.
Una risorsa può essere assegnata **staticamente** e rimarrà valida fino alla terminazione (aree di memoria), oppure **dinamicamente** cambiando durante la loro esistenza.
Può avere anche una richiesta **singola** dove si riferisce a una risorsa di una classe, oppure **multipla**. A sua volta le richieste possono essere **bloccanti** o **non bloccanti**. Una singola risorsa **non può** essere assegnata a più processi.
Una risorsa può essere preemptable (rilascita prima della fine dell'utilizzo).
Queste condizioni deovno valere tutte contemporanteamnete affinché avvenga un deadlock:

- **Mutua esclusione** / non condivisibili, le risorse coinvolte devono essere non condivisibili
- **Assenza di prerilascio**, le risorse devono essere rilasciate volontariamente
- **Richieste bloccanti**, un processo che ha già ottenuto una risorsa può richiederne ancora
- **Attesa circolare**, quando ogni processo richiede una risorsa controllata da un altro processo a catena

Il grafo di **Holt** è un grafo **diretto** e bipartito tra:

- **risorse**
- **processi**

Questo grafo può essere utilizzato per verificare la presenza di un deadlock. Un grafo di Holt si dice **riducibile** se esiste almeno un nodo processo con solo archi entranti

Possiamo individuare diverse gestioni di deadlock:

- Deadlock **detection and recovery**, dato un nodo n, l'insieme dei nodi raggiungibili da n viene detto **insieme di raggiungibilità** di n. Se si verifica una deadlock potremmo terminare **tutti** i processi coinvolti, oppure eliminare un processo alla volta. Vengono salvati dei **checkpoint** per salvare lo stato precedente, ed eventualmente fare una **rollback**.
- Deadlock **prevention / avoidance**, per evitare il deadlock si elimina una delle 4 condizioni di deadlock, prima di assegnare una risorsa si controlla se l'operazione può portare al pericolo di deadlock (**avoidance**)

- **Ostrich** algorithm (struzzo), è la soluzione più adottata nei sistemi Unix. Consiste nell'ignorare i deadlock come se non si possano mai verificare.

Il costo per evitare i deadlock può essere troppo elevato. Per capire il concetto di stallo si è sviluppato l'algoritmo del **banchiere**.

6 Memoria

La parte del SO che gestisce la memoria principale si chiama **memory manager**. Essa si occupa di tenere traccia della memoria libera occupata e allocare/deallocare memoria ai processi. La memory manager è **software** mentre la Memory Management Unit (MMU) è **hardware**.
Con il termine **binding** si indica l'associazione di indirizzi logici di memoria agli indirizzi fisici. Esso può avvenire:

- Durante la compilazione, dove gli indirizzi rimarranno gli stessi ad ogni esecuzione del programma ma non supporta (kernel) la multiprogrammazione
- Durante il caricamento, dove gli indirizzi potrebbero cambiare ad ogni allocazione, il **loader** si preoccupa di aggiornare i riferimenti agli indirizzi di memoria (**rilocabile**)
- Durante l'esecuzione, gli indirizzi di memoria effettivi vengono gestiti dalla **MMU**

Gli indirizzi possono essere:

- **Logici**, dove si utilizzano riferimenti a uno spazio di indirizzamento
- **Fisico**, dove la MMU traduce gli indirizzi logici in fisici

Per effettuare la traduzione degli indirizzi la MMU utilizza un **registro limite** (≥ 1000) che andra a tradurre l'indirizzo in uno fisico (24234).
E' possibile caricare alcuni routine di libreria solo quando vengono richiamate attraverso il **linking dinamico**.
La allocazione può essere **contigua** quando tutto lo spazio assegnato ad un processo è formato da celle consecutive. La MMU basata su rilocazione gestisce solo allocazioni contigue. L'allocazione di memoria oltre a essere **statica** o **dinamica**, può avere:

- Partizioni **fisse**, dove ogni processo viene caricato in una delle partizioni libere sufficientemente grande. Questo tipo di gestione causa **frammentazione interna** causando spreco di memoria per spazio non utilizzato
- Partizioni **dinamiche**, dove la memoria disponibile viene assegnata ai processi che ne fanno richiesta. Problema di questa implementazione è la **frammentazione esterna**, poiché lo spazio libero apparerebbe suddiviso in piccole aree

La frammentazione interna dipende dall'uso di unità di allocazione di dimensione diversa da quella richiesta, mentre la frammentazione esterna deriva dal susseguirsi di allocazioni e deallocazioni. Se è possibile riallocare i processi durante l'esecuzione è possibile procedere alla **compattazione** della memoria dove andiamo a spostare i processi per risolvere il problema della frammentazione esterna.
E' una operazione onerosa e i processi devono essere fermi durante la compattazione.
Quando la memoria è assegnata dinamicamente abbiamo bisogno di una struttura dati per mantenere informazioni sulle zone libere e occupate. E' possibile usare **mappe di bit, liste con puntatori, ecc**. L'operazione di selezione di un blocco può essere:

- **First fit**, dove si scorre la lista dei blocchi liberi fino a quando non trova il primo segmento vuoto grande abbastanza da contenere il processo
- **Next fit**, come First fit, ma si parte dal punto dove si era fermato all'ultima allocazione. L'operazione di selezione di un blocco può essere (peggiore di First fit)
- **Best fit**, seleziona il più piccolo fra i blocchi liberi presenti in memoria
- **Worst fit**, seleziona il più grande fra i blocchi presenti in memoria

Ad oggi viene utilizzato l'approccio della **Paginazione**. Ogni spazio di indirizzamento logico di un processo vinee suddiviso in un insieme di blocchi di dimensione fissa chiamati **pagine**. La memoria fisica in egual modo (**frame**).
La dimensione delle pagine deve essere una potenza di due. Con pagine **piccole**, la tabella cresce di dimensioni, con pagine **grandi** la frammentazione può essere considerevole. Valori tipici (1KB,2KB,4KB).
La tabella delle pagine può essere contenuta in:

- **Registri dedicati**, troppo costosa
- **Totalmente in memoria**, troppi accessi in memoria

Un **Translation lookaside buffer** (TLB) è costituito da un insieme di registri associativi ad alta velocità, ogni registro è diviso in due parti, chiave e valore, essa funge da **cache**.
Nella operazione di **lookup** viene richiesta la ricerca di una chiave che se trovata ritorna un TLB **hit** altrimenti un **miss**.
Nella **segmentazione** (alternativa alla paginazione) la memoria associata ad un processo è suddivisa in aree differenti dal punto di vista funzionale (aree eseguibili, aree dati, area stack) ed ha dimensione variabile. Spetta al programmatore o al compilatore la suddivisione di un programma in segmenti. Ogni segmento ha un nome con dimensione maggiore (64KB).
E' possibile adottare **entrambe le tecniche** a patto che la MMU supporti sia la segmentazione sia la paginazione

La **memoria virtuale**, è la tecnica che permette l'esecuzione di processi che non sono completamente in memoria. Ogni processo ha accesso ad uno **spazio di indirizzamento virtuale**. Se la memoria è piena, si sposta in memoria secondaria i dati contenuti in memoria principale.
La paginazione a richiesta (**demand paging**) utilizza la tecnica della paginazione ammettendo che alcune pagine si trovino in memoria secondaria. Quando un processo tenta di accedere ad una pagina non in memoria il processore genera un **page fault**. Il **pager** si occuperà di caricare la pagina mancante.
Con **swap** si intende l'operazione di copiare l'intera area di memoria usata da un processo:

- Dalla memoria secondaria alla memoria principale (**swap-in**)
- Dalla memoria principale alla memoria secondaria (**swap-out**)

Utilizziamo il termine **swap area** per indicare l'area del disco utilizzata per ospitare le pagine in memoria secondaria. Vengono utilizzati algoritmi di rimpiazzamento per tenere pagine in memoria ad esempio **FIFO**.
Non è detto che aumentando il numero di frame, il numero di page fault diminuisca questo fenomeno si chiama **Anomalia di Belady**. Un algoritmo ottimale dovrebbe selezionare come pagina vittima una pagina che non sarà più acceduta o la pagina che verrà acceduta nel futuro più lontano. Esso però è un algoritmo teorico perché richiederebbe la conoscenza a priori dei futuri programmi.
Algoritmo **LRU**, seleziona come pagina vittima la pagina che è stata usata meno recentemente nel passato. Algoritmo **LFU**, mantiene un contatore del numero di accessi ad una pagina. Viene messa in cima la pagina più usata.
Un processo si dice che è in **trashing** quando spende più tempo per la paginazione che per l'esecuzione. Si definisce **working set di finestra** Δ l'insieme delle pagine accedute nei più recenti riferimenti Δ.

7 Secondaria

Tecniche di gestione dei dispositivi di I/O:

- **Buffering**, gestire una differenza di velocità tra il produttore e il consumatore
- **Caching**, mantiene una copia in memoria primaria di informazioni che si trovano in memoria secondaria
- **Spooling**, è un buffer che mantiene in output per un dispositivo che non può accettare flussi di dati distinti (stampanti)
- **I/O scheduling**

Un disco è composto da un insieme di piatti, suddivisi in tracce, le quali sono suddivise in settori. I dischi sono caratterizzati da 3 parametri:

- La velocità di rotazione
- Il tempo di seek, che indica il tempo medio affinché la testina si sposti sulla traccia desiderata
- La velocità di trasferimento

Anche il gestore del disco utilizza politiche di gestione:

- **FCFS**, FIFO
- **SSTF**, seleziona la richiesta che prevede il minor spostamento della testina
- **LOOK**, ad ogni istante, la testina è associata ad una direzione che si sposta
- **C-LOOK**, ha lo stesso principio del LOOK ma la scansione avviene in una sola direzione

Per aumentare la velocità di gestione del disco si utilizzano tecniche di **RAID**. **Redundant Array of Independent Disks** è uno standard industriale per l'utilizzo di più dischi in parallelo (da 0-6).
L'utilizzo di più dischi in parallelo aumenta la probabilità di guasti nel sistema. Possiamo individuare le seguenti tipologie di RAID:

1. **RAID 0**, può essere utilizzato per applicazioni in cui l'affidabilità non è un grosso problema, ma lo sono la velocità e il basso costo. I dati vengono distribuiti su più dischi. Le richieste possono essere eseguite in parallelo. I dati nel disco logico vengono suddivisi in **strip** (settori). **Non** presenta **ridondanze**.
2. **RAID 1**, la ridondanza è ottenuta duplicando tutti i dati su due insiemi indipendenti di dischi. In questo modo salvando su due strip il costo **raddoppia**. Una richiesta può essere servita da uno qualsiasi dei dischi. Il **recovery** è semplice, se un disco si guasta, i dati sono accessibili dall'altro disco
3. **RAID 4**, si utilizza il meccanismo di **data striping** con strip grandi. Viene usato uno **strip di parita** calcolato bit per bit che andrà ad occupare un intero settore del disco.
   Ad esempio il primo strip da $0-3$ il secondo da $4-7$, ecc. In questo modo se il disco corrispondente è guasto si individua lo strip e si effettua la lettura di tutti gli strip rimasti, e tramite il disco di parità si ottiene lo strip mancante
4. **RAID 5**, come RAID 4 ma i blocchi di parità sono sparsi fra i vari dischi. In questo modo non esiste un disco di parità che diventa un bottleneck
5. **RAID 6**, come RAID 5 ma si utilizzano **due** strip di parità invece di uno. In questo modo viene aumentata l'affidabilità perché è necessario il guasto di 3 dischi affinchè i dati non siano più utilizzabili

8 File System

Il **file system** ha il compito di astrarre la complessità di utilizzo dei diversi media proponendo una interfaccia per i sistemi di memorizzazione. Dal punto di vista dell'utente un file system è compost da due elementi:

- **file**
- **directory**

Esistono 3 tecniche principali per identificare il tipo di un file:

1. Estensioni
2. Utilizzo di un attributo
3. Magic number, sequenza di bit posta prima della sequenza di dati, utile per definire il formato in cui i dati sono memorizzati

Abbiamo diversi metodi di accesso ad un file:

- **Sequenziale** (read, write)
- **Ad accesso diretto** (read pos, write pos)
- **Indicizzato** (read key, write key)

In un SO multitasking, i processi accedono ai file indipendentemente. Le modifiche al contenuto di un file vengono rese visibili agli altri processi immediatamente. Esistono due tipi di condivisione del file:

- Condivisione del puntatore alla posizione corrente nel file (**fork**)
- Condivisione con distinti puntatori alla posizione corrente

Il primo settore dei dischi è il **Master Boot Record** (MBR) utilizzato per:

- fare il **boot** del sistema
- Contiene la **partition table**
- Contiene l'indicazione della partizione attiva

Una partizione caricata dal **Boot block** è formata da:

- **Superblock**, contiene informazioni sul tipo di file system e la sua organizzazione
- **Tabella per la gestione dello spazio libero**, contiene informazioni sui blocchi liberi
- **Tabella per la gestione dello spazio occupato**, contiene informazioni sui file presenti nel sistema

- **Root dir**, directory radice del file system

La **FAT** in sé mantiene la traccia delle aree del disco disponibili e di quelle già usate dai file e dalle directory.
Il problema dell'**allocazione** è il problema del come scegliere i blocchi dati da utilizzare per un file. Possiamo individuare diverse tecniche di allocazione:

1. **Allocazione contigua**, i file sono memorizzati in sequenze contigue di blocchi di dischi. Problema della frammentazione

2. **Allocazione concatenata**, ogni file è costituito da una lista concatenata di blocchi. Ogni blocco contiene un puntatore al blocco successivo. Il descrittore del file contiene i puntatori al primo e all'ultimo elemento della lista. Vengono effettuate molte opererazioni di **seek**

3. **Allocazione basata su fat**, invece di utilizzare parte del blocco dati per contenere il puntatore al blocco successivo si crea una tabella unica con un elemento per blocco (**cluster**). La scansione richiede anche la lettura della FAT, aumentando il numero di accessi al disco (chiavette)

4. **Allocazione indicizzata**, l'elenco dei blocchi che compongono un file viene memorizzato in un blocco indice. Per accedere ad un file, si carica in memoria la sua area indice e si utilizzano i puntatori contenuti (**puntatori raccolti in un blocco**).
   L'ultimo elemento del blocco indice non punta al blocco dati ma al blocco indice successivo. La dimensione del blocco indice determina l'ampiezza massima del file

In **UNIX** ogni file è associato ad un **i-node**. Un i-node è una struttura dati contenente gli attributi del file, e un indice di blocchi diretti e indiretti.
Per gestire lo spazio libero vengono usate le **mappe di bit**. Ogni blocco corrisponde ad un **bit** nella bitmap. I blocchi libero sono indicati da 0 e i blocchi occupati a 1. Questa tecnica può richiedere molto spazio.
I blocchi liberi invece vengono mantenuti in una **lista concatenata**. L'allocazione di un'area di ampie dimensioni è costosa.
Le liste concatenate possono anche essere gestite a **blocchi**, in questo modo è sufficiente mantenere in memoria solo un blocco contenente elementi liberi.

- Blocchi $1KB$
- Dimensione partizione $16GB$
- Mappa di bit $2MB$
- Lista concatenata (FAT) $48MB$
- Lista concatenata (blocchi) spazio blocchi liberi

Scegliere la dimensione di un cluster:

- **Cluster grandi**: velocità alta, frammentaizone interna

- **Cluster piccoli**: velocità bassa, minore frammentazione

Una directory è un **file speciale** contenente informazioni sui file contenuti nella directory. Una directory è suddivisa in un certo numero di **directory entry**. Gli attributi possono essere inseriti nelle directory entry oppure nell'**i-node** (UNIX).
Una directory può essere implementata in:

- **Lista lineare**
- **Tabella hash**

Due implementazioni possibili:

- **Link simbolici**, dove viene creato un riferimento al file in questione. Quando si scopre che si tratta di un link esso viene **risolto**
- **Hard link**, dove le informazionial file sono presenti in entrambe le directory. Se viene modificato il file, esso cambierà in entrambi. E' necessario utilizzare la tecnica degli i-node i quali conteggiano il **numero di riferimenti**.

I meccanismi di caching possono causare inconsistenza nel file system. Nei file system basati su **log** ogni aggiornamente è trattato come una **transazione**. Essa consente di ripristinare uno stato corrente.

9 Sicurezza

L'insieme dei meccanismi utilizzati per trasformare messaggi in chiaro (**plaintext**) in un messaggio cifrato (**ciphertext**). Nella crittografia a **chiave privata** la chiave per criptare i messagi è la stessa usata per decriptarli. Gli algoritmi utilizzati sono molto veloci. La crittografia a **chiave pubblica** ha due chiavi distinte:

- **chiave pubblica**, utilizzata per criptare i messaggi in chiaro
- **chiave privata**, utilizzata per decriptare i messaggi cifrati

Per criptare le password viene utilizzato il meccanismo del **salt** dove prima di essere criptate e memorizzate nel file di password, le password vengono concatenate con un numero casuale. Una volta le password venivano salvate in chiaro nel file `/etc/passwd`. **Pluggable Authentication Module** (PAM) è un servizio di autenticazioni, basato su file di configurazione.
L'insieme di meccanismi che separano il **gestore** (nucleo SO) dalle **risorse gestite** (processi, risorse) è realizzato mediante:

- **Mode bit**, meccanismo degli interrupt
- **Protezione**, memoria dispositivi

Le **autorizzazioni** sono l'insieme di meccanismi e politiche con cui il SO decide se un soggetto ha il permesso di eseguire una determinata azione. Vengono realizzati tramite meccanismi software:

- Trusted Computing Base, Reference Monitor
- **ACL**, matrice di accesso

Molti SO controllano i diritti solo al momento dell'apertura. Non vengono controllate le operazioni successive (principio di **accesso mediato**). Un sistema non dovrebbe concedere permessi in base ad una singola condizione, (principio di **accesso mediato**) (**UNIX**). Nessun soggetto ha diritti per default (principio di **failsafe default**).
Nelle **ACL** ad ogni oggetto viene associata una lista di elementi *<dominio, diritti di accesso>*. L'associazione soggetto/dominio può essere **statica** o **dinamica**. UNIX ha liste di 3 elementi: owning user, owning group, others. Nelle **capability** ad ogni dominio viene associata una **lista** di capability, ovvero coppie `<oggetti, diritti di accesso>`. I processi mantengono le capability e le presentano quali credenziali per accedere all'oggetto. Sono una sorta di chiave per l'accesso alla serratura che protegge l'oggetto. Le capability possono essere mantenute:

- nello spazio kernel
- nello spazio utente

Ogni processo possiede 6 o più ID associate ad esso. Le **umask** sono maschere utilizzate per la creazione dei file (triple). In un file system i permessi di ogni oggetto vengono rappresentati come una **ACL**.
Una ACL consiste in un insieme di **ACL entry**. Ogni user class è rappresentata da una **AC entry**. Ogni file è associato ad una **access ACL**, esse determinano i diritti di accesso. Se una directory non ha default ACL, si utilizza il meccanismo tradizionale UNIX (umask, etc). Le ACL sono implementate come **EA** (Extended attribute) con coppia `name = value`.
File system diversi hanno supporti per ACL diversi. Tutti supportano le POSIX ACL. Ogni processo ha 3 **bitmap** che rappresentano capability:

1. **Effective Set**, contiene le capability che il processo possiede ad un certo istante
2. **Permitted Set**, contiene il massimo insieme di capability che un processo possiede
3. **Inheritable Set**, continee il sottoinsieme di capability che un processo può lasciare in eredità ai suoi sottoprocessi

Nelle **Discrectionary Access Control** (DAC) i controlli di accesso sono basati sull'identità dei soggetti e sui permessi di accesso assegnati agli oggetti (sufficiente quando il sistema ha **dati** che devono essere protetti da **abusi del proprietario)**. Nei modelli non discrezionali **Mandatory, MAC** il controllo degli accessi è bassato su regole e informazioni associate ai soggetti ed agli oggetti.