OS

Sistema Operativo

Controlla l'esecuzione di programmi applicativi

Interfaccia tra le applicazioni e l'hardware del calcolatore

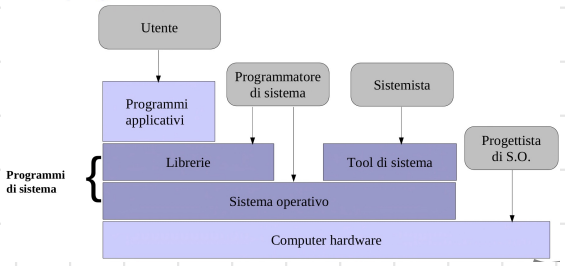
Come Gestore di Risorse

Problemi di un OS:

- Efficienza nell'uso delle risorse (Processore, memoria, dischi, ecc.)
- Protezione nell'uso delle risorse
- Il S.O. lascia il controllo alle applicazioni e affidarsi al processore per riottenere il controllo

Come Macchina Astratta:

- **Visione "a strati" delle componenti hardware/software che compongono un elaboratore:**



- Agisce come intermediario tra programmatore e hardware
- Indipendenza dall'hardware
- Comodità d'uso
- Programmabilità

Servizi Offerti:

- esecuzione di programmi
- accesso semplificato ai dispositivi I/O
- Accesso controllato a dispositivi, file system, etc.
- Accesso al sistema
- Rilevazione e risposta agli errori
- Accounting

Multi-Programmazione

(utilizzare il processore durante i periodi di I/O di un Job per eseguire altri Job)

Vantaggi:
- Il Processore non viene lasciato inattivo (Idle) durante operazioni di I/O molto lunghe.
- La memoria viene utilizzata al meglio, caricando il maggior numero di Job possibili.

Scheduler

(Componente del S.O. che si preoccupa di alternare i Job nell'uso della CPU.)

Quando un Job richiede un operazione di I/O, la CPU viene assegnata ad un altro Job.

Time-Sharing

(È l'estensione logica della multiprogrammazione)

- Nasce per realizzare un sistema multi-utente
- L'unità di elaborazione centrale del computer viene utilizzata per rispondere alle richieste dei singoli utenti, passando rapidamente da uno all'altro [context-switch (commutazione di contesto)].

Dando l'impressione di avere a disposizione il computer centrale interamente per sé.

- Dato il grande numero di programmi si rende necessario la gestione della Memoria virtuale.
- CPU scheduling: lo scheduling deve essere di tipo preemptive o time-sliced, cioè sospendere periodicamente l'esecuzione di un programma a favore di un altro.
- Meccanismi di Protezione: La presenza di più utenti rende necessari meccanismi di protezione (ex. Protezione nel file system, della memoria, etc.)

Sistemi

SIMD
(Single Instruction, Multiple Data)
- Le CPU eseguono all'unisono lo stesso programma su dati diversi.

MIMD
(Multiple Instruction, Multiple Data)
- Le CPU eseguono programmi differenti su dati differenti

Tassonomia basata sulla Struttura

Sistemi a basso Parallelismo
Pochi processori in genere molto potenti

Sistemi massicciamente Paralleli
Gran numero di processori, che possono avere anche potenza non elevata

Tassonomia basata sulla Dimensione

Symmetric multiprocessing (SMP)
- Ogni processore esegue una copia identica del S.O.
- Processi diversi possono essere eseguiti contemporaneamente

Asymmetric multiprocessing
- Ogni processore é assegnato ad un compito specifico; un processore master gestisce l'allocazione del lavoro ai processori slave
- Piú comune in sistemi estremamente grandi

Sistemi Paralleli (Sistemi tightly coupled)
(un singolo elaboratore che possiede più unità di elaborazione)
- Risorse dell'elaboratore condivise (es. memoria)
- Vi é un incremento delle prestazioni
- Incremento dell'affidabilitá (graceful degradation)

Sistemi Distribuiti (loosely coupled)
Sistemi composti da piú elaboratori indipendenti (con proprie risorse e proprio sistema operativo)
- Ogni processore possiede la propria memoria locale
- I processori sono collegati tramite linee di comunicazione (rete, linee telefoniche, linee wireless)

Sistemi Real-Time
Sono sistemi per i quali la correttezza del risultato non dipende solamente dal suo valore, ma anche dall'istante nel quale il risultato viene prodotto
- Hard real-time: il mancato rispetto dei vincoli temporali Puó avere effetti catastrofici (controllo assetto veivoli, centrali nucleari, ecc..)
- Soft real-time: Se si hanno solamente disagi o disservizi (programmi interattivi)

Interrupt

Meccanismo che permette l'interruzione del normale ciclo di esecuzione della CPU

- Possono essere sia Software che Hardware
- Possono essere mascherati (ritardati) se la CPU sta svolgendo compiti non interrompibili

Hardware

Eventi hardware asincroni non causati dal processo in esecuzione.

- Dispositivi I/O (notifica di eventi operazioni I/O)
- Clock

Software (Trap)

Causato dal programma

- Richiesta di servizi di sistema (system call)
- Eventi eccezionali come divisione per 0(zero) o problemi di indirizzamento

Gestione Interrupt - Panoramica

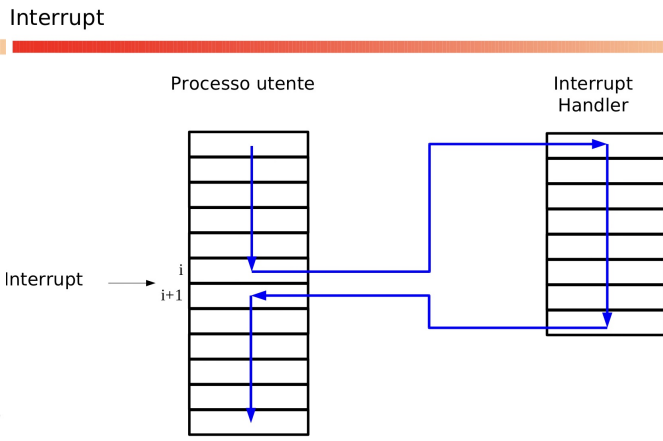
- **Cosa succede in seguito ad un interrupt**

Hardware

- Un segnale "interrupt request" viene spedito al processore
- Il processore
  - sospende le operazioni del processo corrente
  - salta ad un particolare indirizzo di memoria contenente la routine di gestione dell'interrupt (*interrupt handler*)

Software

- L'interrupt handler
  - gestisce nel modo opportuno l'interrupt
  - ritorna il controllo al processo interrotto (o a un altro processo, nel caso di scheduling)
- Il processore riprende l'esecuzione del processo interrotto come se nulla fosse successo



Interrupt Driven

I S.O. moderni sono detti Interrupt Driven, gran parte del nucleo di un S.O. viene eseguito come interrupt handler

Sono gli interrupt (o i trap) che guidano l'avvicendamento dei processi

Interrupt Multipli

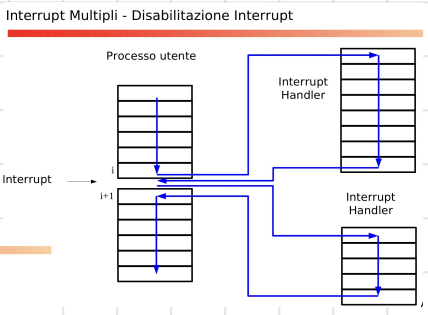
Interrupt generati da dispositivi diversi, può avvenire anche durante la gestione di un interrupt precedente

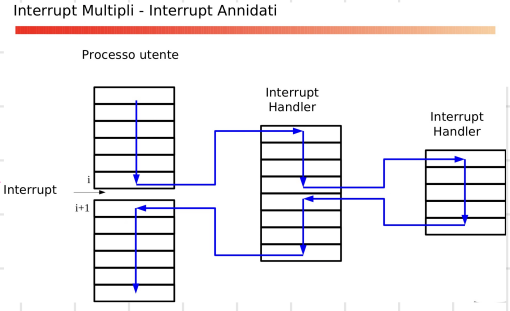
Approcci possibili:
- disabilitazione degli interrupt
- interrupt annidati

Interrupt Annidati

- Possibilità di dare diverse priorità agli Interrupt
- Necessario meccanismo di salvataggio e ripristino dell'esecuzione

Tutto ciò è un approccio più complesso, ma i dispositivi veloci possono essere serviti prima (es. schede di rete)





Comunicazione fra processore e dispositivi I/O

Programmed I/O (obsoleto)

1) La CPU carica i parametri della richiesta nei registri di comando
2) Il dispositivo esegue la richiesta
3) Il risultato viene registrato in un buffer locale
4) I registri di status segnalano il completamento dell'operazione
5) Il S.O. attende (busy waiting) che il comando sia stato completato periodicamente
6) La CPU copia i dati dal buffer locale del controller alla memoria

Interrupt - Driven I/O

2) A differenza del Programmed I/O successivamente al punto 1 il S.O. sospende l'esecuzione del processo che ha eseguito l'operazione di Input ed esegue un altro processo.
4) Il completamento dell'operazione viene segnalato attraverso un Interrupt
5) quindi: non vi è busy waiting
6) copia dei dati dal buffer locale alla memoria

Direct Memory Access (DMA)

Il S.O.:
- attiva l'operazione di I/O specificando l'indirizzo in memoria di destinazione (Input) o di provenienza (output)
- l'Interrupt specifica solamente la conclusione dell'operazione di I/O

VANTAGGI E SVANTAGGI:
- contesa nell'accesso al bus
- device driver più semplici
- Efficace perché la CPU non accede al bus ad ogni ciclo di clock

Sono 2 approcci similari

Svantaggi sono:
- Il Processore spreca gran parte del tempo nella gestione del trasferimento dei dati
- La velocità di trasferimento è limitata dalla velocità con cui il processore riesce a gestire il servizio

Programma

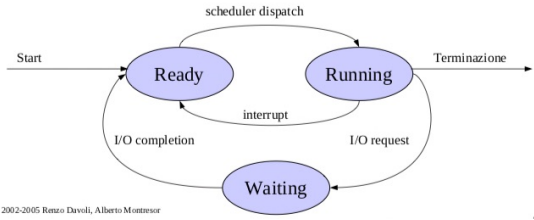
- Entità statica
- Specifica un'insieme di istruzioni e la loro sequenza di esecuzione

Processo

- Entità Dinamica
- Rappresenta il modo in cui un programma viene eseguito nel tempo
- Più processi possono eseguire lo stesso programma

Vita di un processo

- **Stati dei processi:**
  - *Running*: il processo è in esecuzione
  - *Waiting*: il processo è in attesa di qualche evento esterno (e.g., completamento operazione di I/O); non può essere eseguito
  - *Ready*: il processo può essere eseguito, ma attualmente il processore è impegnato in altre attività



Tema centrale nella progettazione dei S.O. riguarda la gestione di processi *multipli*

- *Multiprogramming* — Interleaving
  - più processi su un solo processore
  - parallelismo *apparente*
- *Multiprocessing* — Overlapping
  - più processi su una macchina con processori multipli
  - parallelismo *reale*
- *Distributed processing*
  - più processi su un insieme di computer distribuiti e indipendenti
  - parallelismo *reale*

Sia che il sistema sia a Singolo Processore sia che sia a multiProcessore si presenta lo stesso Problema: Non è possibile predire la velocità relativa dei processi.

Race condition

- **Definizione**
  - Si dice che un sistema di processi multipli presenta una *race condition* qualora il risultato finale dell'esecuzione dipenda dalla temporizzazione con cui vengono eseguiti i processi
    o dalla sequenza
- **Per scrivere un programma concorrente:**
  - è necessario eliminare le race condition

Tre Tipologie di ITERAZIONE tra Processi

Processi che sono uno dell'altro:

Ignari:

- Non progettati per lavorare insieme
- Competono per le stesse risorse
- devono sincronizzarsi nella loro utilizzazione
- Il S.O. arbitra la sincronizzazione

Indirettamente a conoscenza:

- Condividono risorse come ad esempio un buffer, per scambiarsi informazioni, non si conoscono tramite il loro ID
- Cooperano per uno scopo
- Devono sincronizzarsi nel utilizzo delle risorse
- Il S.O. facilita, fornendo meccanismi di sincronizzazione

Direttamente a conoscenza:

- Comunicano l'uno con l'altro sulla base dei loro ID
- La comunicazione è diretta, spesso basata sullo scambio di messaggi
- Cooperano per qualche scopo
- Comunicano informazioni ad altri Processi
- Il S.O. facilita, fornendo meccanismi di sincronizzazione

Proprietà di un Programma Concorrente

Safety

- Mostra che il programma (se avanza) "va nella direzione voluta", cioè non esegue azioni scorrette
- I processi non devono Interferire tra loro nel accesso alle risorse condivise

Liveness

- Il programma avanza, non si ferma.
- I meccanismi di sincro. utilizzati non devono prevenire

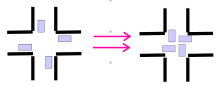
Mutua Esclusione

Accesso ad una risorsa, che ad ogni istante, al massimo un Processo può accedere a quella risorsa

Deadlock

Situazione di stallo dove due o più processi o azioni si bloccano a vicenda, aspettando che uno esegua una certa azione che serve all'altro e viceversa

Esempio: incrocio stradale



Starvation

Si intende l'impossibilità perpetua, da parte di un processo pronto all'esecuzione, di ottenere le risorse sia hardware che software di cui necessita per essere eseguito.

- A differenza del Deadlock, non è una condizione definitiva, basta adottare un'opportuna politica di assegnamento.

DA EVITARE

Riassunto

Tipo	Relazione	Meccanismo	Problemi di controllo
processi "ignari" uno dell'altro	competizione	sincronizzazione	mutua esclusione deadlock starvation
processi con conoscenza indiretta l'uno dell'altro	cooperazione (sharing)	sincronizzazione	mutua esclusione deadlock starvation
processi con conoscenza diretta l'uno dell'altro	cooperazione (comunicazione)	comunicazione	deadlock starvation

SEZIONE CRITICA

La parte di un programma che utilizza una o più risorse condivise viene detta sezione critica (CRITICAL SECTION = CS)

Abbiamo bisogno di costrutti specifici perché il S.O. non può capire da solo cosa è e cosa non è una sezione critica.

La responsabilità di garantire la mutua esclusione ricade sul S.O. o sul linguaggio.
Esempi: - Semafori
- Monitor
- Message passing

Sezioni critiche

- **Requisiti per le CS**
  1) *Mutua esclusione*
     - Solo un processo alla volta deve essere all'interno della CS, fra tutti quelli che hanno una CS per la stessa risorsa condivisa
  2) *Assenza di deadlock*
     - Uno scenario in cui tutti i processi restano bloccati definitivamente non è ammissibile
  3) *Assenza di delay non necessari*
     - Un processo fuori dalla CS non deve ritardare l'ingresso della CS da parte di un altro processo
  4) *Eventual entry (assenza di starvation)*
     - Ogni processo che lo richiede, prima o poi entra nella CS

Soluzioni Software

Permettono di risolvere il problema delle critical section.
Problemi: - sono tutte basate sul busy waiting.
- busy waiting spreca il tempo del processore
- È una tecnica che NON dovrebbe essere utilizzata!

Soluzioni:

- Test & Set
  Altri esempi:
- Fetch & set
- Compare & swap
- etc...

Test & Set

- `TS(x,y) := < y = x ; x = 1 >`
- spiegazione
  - ritorna in **y** il valore precedente di **x**
  - assegna 1 ad **x**

Test & Set

```
shared lock=0; cobegin P // Q coend.
process P {
  int vp;
  while (true) {
    do { f(); vp=0;
      TS(lock, vp);
    } while (vp);
    critical section
    lock=0;
    non-critical section
  }
}
process Q {
  int vp;
  while (true) {
    do {
      TS(lock, vp);
    } while (vp);
    critical section
    lock=0;
    non-critical section
  }
}
```

- **Mutua esclusione**
  - entra solo chi riesce a settare per primo il lock
- **No deadlock**
  - il primo che esegue TS entra senza problemi
- **No unnecessary delay**
  - un processo fuori dalla CS non blocca gli altri
- **No starvation**
  - No, se non assumiamo qualcosa di più

SEMAFORI BINARI

Semafori il cui valore può assumere solo valori 0 e 1.
Garantiscono mutua esclusione

Semafori

Paradigma per la sincronizzazione

Def:
Due o più processi possono cooperare attraverso semplici segnali, in modo tale che un processo possa essere bloccato in specifici punti del suo programma finché non riceve un segnale da un altro processo

Operazioni

V
- Invocata per inviare un segnale, quale il verificarsi di un evento o il rilascio di una risorsa
- Deve svegliare uno dei processi sospesi

P
- Viene invocata per attendere il segnale (ovvero per attendere un evento o il rilascio di una risorsa)
- Deve Sospendere il processo invocante

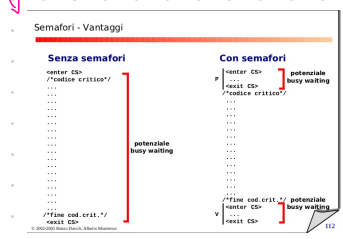
Semafori FIFO

- Il processo che è stato sospeso più a lungo viene svegliato per primo
- Politica fair, garantisce assenza di starvation
- La struttura dati è una coda

Se non viene specificato l'ordine in cui vengono rimossi i semafori possono dare origine a Starvation.

Vantaggi semafori

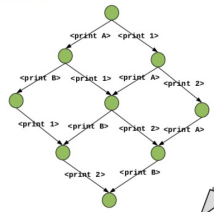
Utilizzando queste tecniche non abbiamo eliminato il busy waiting, lo abbiamo però limitato alle sezioni critiche di P e V e queste sezioni critiche sono molto brevi



Interleaving di azioni atomiche

- **Cosa stampa questo programma? (Vi ricordate?)**

```
process P {
  <print A>
  <print B>
}
process Q {
  <print 1>
  <print 2>
}
```

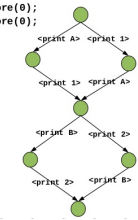


In questo caso non sappiamo chi fra i Processi verrà eseguito prima e chi dopo, prima A e poi B? prima A e poi 1?

Interleaving con semafori

- **Cosa stampa questo programma?**

```
Semaphore s1 = new Semaphore(0);
Semaphore s2 = new Semaphore(0);
process P {
  <print A>
  s1.V()
  s2.P()
  <print B>
}
process Q {
  <print 1>
  s2.V()
  s1.P()
  <print 2>
}
```



Con l'implementazione dei semafori possiamo sospendere un processo e successivamente svegliarlo

Difetti dei SEMAFORI

- Sono costrutti di basso livello
- Il programmatore non deve:
  - Omettere P e V
  - scambiare l'ordine delle operazioni P e V
  - Fare operazioni P e V su semafori sbagliati
- Necessario accedere ai dati condivisi in modo corretto
- Vi sono forti problemi di "leggibilità"

Monitor

Paradigma di programmazione concorrente che fornisce un approccio più strutturato alla programmazione concorrente

È un modulo software costituito da:

- dati locali
- una sequenza di inizializzazione
- una o più "procedure" (un processo entra in un monitor invocando una delle sue procedure)

Solo un processo alla volta può essere all'interno del monitor; gli altri processi che invocano il monitor sono sospesi, in attesa che il monitor diventi disponibile.

- Un monitor assomiglia un po' ad un "oggetto" nella programmazione O.O.

- Il Monitor fornisce un semplice meccanismo di mutua esclusione
- Strutture dati condivise possono essere messe all'interno di un monitor

Procedimento di sincronizzazione tra processi concorrenti che, con il quale si impedisce a task parallelli di accedere contemporaneamente ai dati in memoria o ad altre risorse soggette a RACE CONDITION

Abbiamo bisogno di meccanismi di sincronizzazione

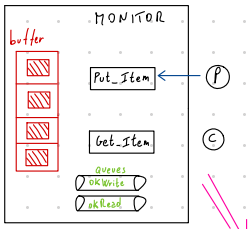
Monitor - Meccanismi di sincronizzazione

- **Dichiarazione di variabili di condizione (CV)**
  - `condition c;`
- **Le operazioni definite sulle CV sono:**
  - `c.wait()`
    attende il verificarsi della condizione
  - `c.signal()`
    segnala che la condizione è vera

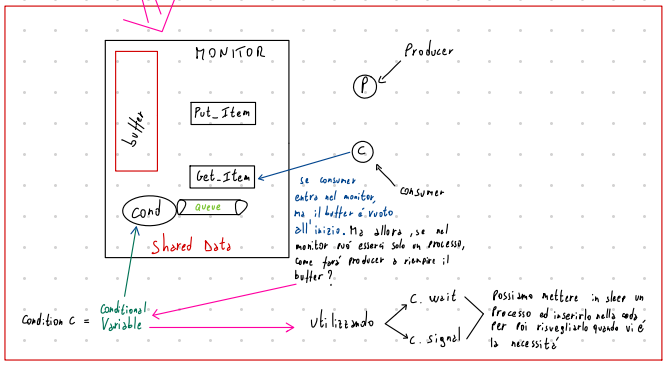
Monitor - Politica signal urgent

- `c.wait()`
  - viene rilasciata la mutua esclusione
  - il processo che chiama `c.wait()` viene sospeso in una coda di attesa della condizione c
- `c.signal()`
  - causa la riattivazione immediata di un processo (secondo una politica FIFO)
  - il chiamante viene posto in attesa
  - verrà riattivato quando il processo risvegliato avrà rilasciato la mutua esclusione (*urgent stack*)
  - se nessun processo sta attendando c la chiamata non avrà nessun effetto

Monitor e Buffer limitato



Simile è il caso in cui il buffer è limitato e pieno e producer tenta ancora di riempire il buffer oppure il buffer è vuoto e consumer tenta di leggere



Buffer limitato tramite Monitor

```
monitor PCController {
  Object[] buffer;
  condition okRead, okWrite;
  int count, rear, front;

  PCController(int size) {
    buffer = new Object[size];
    count = rear = front = 0;
  }

  procedure entry Object read() {}
    if (count == 0)
      okRead.wait();// inserisco nella coda Read
    int retval = buffer[rear];
    cont--;
    rear = (rear+1) % buffer.length;
    okWrite.signal(); //risveglio processo scrittura dalla coda write
    return retval;

  procedure entry void write(int val)
  {
    if (count == buffer.length)
      okWrite.wait(); // buffer pieno inserisco in coda write
    buffer[front] = val;
    count++;
    front = (front+1) %
  buffer.length;
    okRead.signal();//sblocco processo in coda di Read
  }
```

Monitor - wait/signal vs (P/V dei semafori)

- **A prima vista:**
  - **wait** e **signal** potrebbero sembrare simili alle operazioni sui semafori
    P e **V**
- **Non è vero!**
  - **signal** non ha alcun effetto se nessun processo sta attendendo la condizione
    V "memorizza" il verificarsi degli eventi
  - **wait** è sempre bloccante
    P (se il semaforo ha valore positivo) no
  - il processo risvegliato dalla **signal** viene eseguito per primo

Semafori e Monitor sono paradigmi di sincronizzazione, il Message Passing è un paradigma di comunicazione tra processi.
La comunicazione avviene tramite lo scambio di messaggi, e non semplici segnali

MESSAGE PASSING

Operazioni

SEND

"Spedisce" un messaggio ad un Processo destinatario
(Deve essere specificato il destinatario)

RECEIVE

Utilizzato dal processo destinatario per "ricevere" un messaggio da un processo mittente
(Il processo mittente può essere specificato, o può essere qualsiasi)

MP Sincrono

- **Operazione send sincrona**
  - sintassi: **ssend(m, q)**
  - il mittente **p** spedisce il messaggio **m** al processo **q**, restando bloccato fino a quando **q** non esegue l'operazione **sreceive(m, p)**
- **Operazione receive bloccante**
  - sintassi: **m = sreceive(p)**
  - il destinatario **q** riceve il messaggio **m** dal processo **p**; se il mittente non ha ancora spedito alcun messaggio, il destinatario si blocca in attesa di ricevere un messaggio
  - è possibile lasciare il mittente non specificato (utilizzando *)

MP Asincrono

- **Operazione send asincrona**
  - sintassi: **asend(m, q)**
  - il mittente **p** spedisce il messaggio **m** al processo **q**, senza bloccarsi in attesa che il destinatario esegua l'operazione **areceive(m, p)**
- **Operazione receive bloccante**
  - sintassi: **m = areceive(p)**
  - il destinatario **q** riceve il messaggio **m** dal processo **p**; se il mittente non ha ancora spedito alcun messaggio, il destinatario si blocca in attesa di ricevere un messaggio
  - è possibile lasciare il mittente non specificato (utilizzando *)

MP Totalmente Asincrono

- **Operazione send asincrona**
  - sintassi: **asend(m, q)**
  - il mittente **p** spedisce il messaggio **m** al processo **q**, senza bloccarsi in attesa che il destinatario esegua l'operazione **nb-receive(m, p)**
- **Operazione receive non bloccante**
  - sintassi: **m = nb-receive(p)**
  - il destinatario **q** riceve il messaggio **m** dal processo **p**; se il mittente non ha ancora spedito alcun messaggio, la **nb-receive** termina ritornando un messaggio "nullo"
  - è possibile lasciare il mittente non specificato (utilizzando *)

Possiamo implementare il MP sincrono dato quello assincrono e viceversa.

MP sincrono dato quello asincrono

```
void ssend(Object msg, Process q) {
  asend(msg, q);
  ack = areceive(q);
}
Object sreceive(p) {
  Object msg = areceive(p);
  asend(ack, p);
  return msg;
}
```

MP asincrono dato quello sincrono

```
/* p is the calling process */
void asend(Object m, Process q) {
  ssend("SND(m,p,q)", server);
}
void areceive(Process q) {
  ssend("RCV(p,q)", server);
  Object m = sreceive(server);
  return m;
}
process server {
  /* One element x process pair
  */
  int[][] waiting;
  Queue[][] queue;
  while (true) {
    handleMessage();
  }
}
```

```
void handleMessage() {
  msg = sreceive(*);
  if (msg == <SND(m,p,q)>) {
    if (waiting[p,q]>0) {
      ssend(m, p);
      waiting[p,q]--;
    } else {
      queue[p,q].add(m);
    }
  } else if (msg == <RCV(q,p)>) {
    if (queue[p,q].isEmpty()) {
      waiting[p,q]++;
    } else {
      m = queue[p,q].remove();
      ssend(m, q);
    }
  }
}
```

Queue è la coda dei messaggi in attesa di essere prelevati dal processo ricevente.
Waiting è la coda dei processi in attesa che il processo mittente invii il messaggio al server.

- Nel primo caso utilizziamo un ack in modo da mettere in pausa il processo mittente e aspettare la risposta dal processo ricevente

- Nel secondo caso creiamo un processo server che fa da dispaccio.
  Il mittente manda un messaggio al server contenente (mittente, destinatario, messaggio) e contemporaneamente il destinatario per ricevere.
- Il processo vive in un ciclo infinito (while true)
- In ambiente unix spesso questi server vengono chiamati Demoni

Struttura dei S.O.

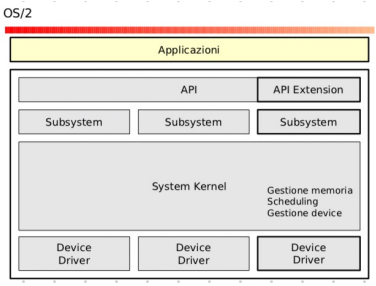
- Gestione dei processi
- Gestione memoria principale
- Gestione memoria secondaria
- Gestione file system
- Gestione dei dispositivi I/O
- Protezione
- Networking
- Interprete dei comandi

Sistemi con struttura a strati

- Ogni strato: – È basato sugli strati inferiori
  – Offre servizi agli strati superiori
- Il vantaggio principale è la modularità
  - Encapsulation e data hiding
  - Abstract data types
- Vengono semplificate le fasi di implementazione, debugging, ristrutturazione del sistema
- Tendono ad essere meno efficienti
  - Ogni strato tende ad aggiungere overhead

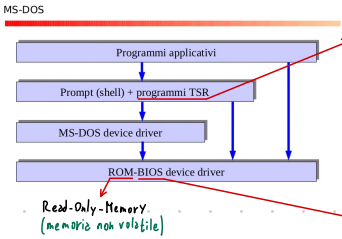
> **overhead** (letteralmente *in alto, che sta di sopra*) serve per definire le risorse accessorie, richieste in sovrappiù rispetto a quelle strettamente necessarie per ottenere un determinato scopo in seguito all'introduzione di un metodo o di un processo più evoluto o più generale.

- Le funzionalità al layer N devono essere implementate utilizzando esclusivamente i servizi dei livelli inferiori
- I moderni S.O. tendono ad avere meno strati



Sistemi con struttura semplice

- struttura non progettata a priori
- possono essere descritti come una collezione di procedure, ognuna delle quali può richiamare altre procedure
- tipicamente sono S.O. semplici e limitati che hanno subito un'evoluzione al di là dello scopo originario



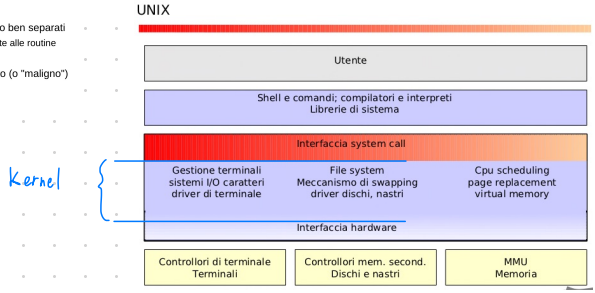
In informatica sono detti **Terminate and Stay Resident** (abbreviato **TSR**), quei programmi che, una volta terminata la loro esecuzione, restituiscono il controllo al sistema operativo restando però residenti in memoria. I programmi TSR sono stati usati sui sistemi MS-DOS per simulare il multitasking trasferendo il controllo al programma in memoria in modo automatico oppure tramite eventi generati esternamente, come la pressione di un determinato tasto: alcuni programmi TSR erano driver per dispositivi hardware non supportati direttamente dal sistema operativo mentre altri erano applicazioni che offrivano funzionalità aggiuntive.[1][2]

MS-DOS era un sistema operativo **single-tasking**, ossia era in grado di eseguire un solo programma alla volta

La **Basic Input-Output System** (in acronimo, BIOS, pronuncia inglese: ˈbaɪɒs), in informatica, è un insieme di routine software, generalmente scritte su memoria ROM, FLASH o altra memoria non volatile, che fornisce una serie di funzioni di base per l'accesso all'hardware del computer e alle periferiche integrate sulla scheda madre da parte del sistema operativo e dei programmi.

In MS-DOS

- le interfacce e i livelli di funzionalità non sono ben separati
  - le applicazioni possono accedere direttamente alle routine di base per fare I/O
- come conseguenza, un programma sbagliato (o "maligno") può mandare in crash l'intero sistema



Organizzazione del kernel

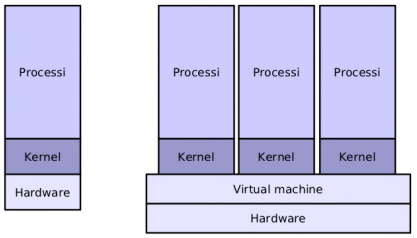
- **Esistono 4 categorie di Kernel**
  - Kernel Monolitici
    - Un aggregato unico (e ricco) di procedure di gestione mutuamente coordinate e astrazioni dell'HW
  - Micro Kernel
    - Semplici astrazioni dell'HW gestite e coordinate da un kernel minimale, basate un paradigma client/server, e primitive di message passing
  - Kernel Ibridi
    - Simili a Micro Kernel, ma hanno componenti eseguite in kernel space per questioni di maggiore efficienza
  - ExoKernel
    - Non forniscono livelli di astrazione dell'HW, ma forniscono librerie che mettono a contatto diretto le applicazioni con l'HW

Monolitico vs MicroKernel

Monolitico:
- Minor complessità nel gestire il codice di controllo in un'unica area di indirizzamento (kernel space).

MicroKernel:
- Più usato in contesti dove non si ammettono failure(braccio robotico space shuttle)

Macchine virtuali



Senza VM — Con VM

Macchine Virtuali

Pro:
- Consentono di far coesistere s.o. differenti.
- Possono emulare architetture hardware differenti.

Cont:
- Soluzione inefficiente.
- Difficile condividere risorse.

Java

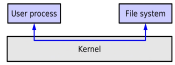
Gli eseguibili Java (detti bytecode) vengono eseguiti sulla Java Virtual Machine, questa macchina viene emulata in quasi tutte le architetture reali.
Il vantaggio è:
- Il codice è altamente portabile è relativamente veloce.
- Debugging facilitato.
- Controlli di sicurezza sul codice eseguibile.

Kernel Monolitici

- System Calls dei kernel Monolitici:
- Implementano servizi forniti dal kernel, tipicamente realizzati in moduli eseguiti in kernel mode.
- Esiste modularità, anche se l'integrazione del codice, e il fatto che tutti i moduli sono eseguiti nello stesso spazio, è tale da rendere tutto l'insieme un corpo unico in esecuzione.
- Alto grado di efficienza
- I più recenti kernel monolitici (Es. LINUX) permettono di effettuare il caricamento (load) di moduli eseguibili a runtime
  Possibile estendere le potenzialità del kernel, solo su richiesta

Microkernel o sistemi client/server

- Rimuovere dal kernel tutte le parti non essenziali e implementarle come processi a livello utente
  esempio: Per accedere ad un file, un processo interagisce con il processo gestore del file system
- La comunicazione è basata su message passing, è il microkernel che si occupa di smistare i messaggi fra i vari processi



- Kernel più semplice e facile da realizzare, più espandibile e modificabile(perchè basta aggiungere un processo a livello utente, senza dover ricompilare il kernel).
- Il S.O. è più facile da portare su altre architetture poiché molti servizi (es. il file system) possono essere semplicemente ricompilati.
- Il S.O. è più robusto perchè se un servizio cade, il resto del sistema può continuare ad eseguire.

Pro:
- Assegnamento al microkernel e processi di sistemi livelli di sicurezza diversi.
- Comunicazione tra processi nello stesso sistema o tra macchine differenti.

Cont:
- Inefficienza(overhead determinato dalla comunicazione tramite kernel del S.O.)
- parzialmente superata con i S.O. più recenti.

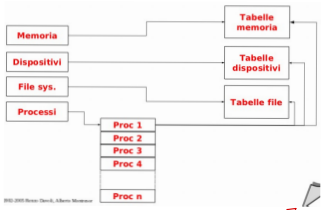
Kernel Ibridi

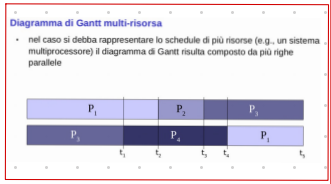
Si tratta di micro kernels che mantengono una parte di codice in "kernel space" per ragioni di maggiore efficienza di esecuzione, e adottano massage passing tra i moduli in "user space".
Non bisogna confonderli con i Kernel Monolitici che sono in grado di effettuare il caricamento (load) di moduli dopo la fase di boot.

ExoKernel(kernel di sistemi operativi a struttura verticale)

Il progettista dell'applicazione ha tutti gli elementi di controllo per decisioni riguardo alle prestazioni HW.
Dispone di Libreria di interfacce connesse all'ExoKernel(es. User vuole allocare area di memoria x o settore disco y)
LIMITI:
- Tipicamente non vanno oltre l'implementazione dei servizi di protezione e multiplazione delle risorse.
- Non forniscono astrazione concreta del HW

Overhead: serve per definire le risorse accessorie, richieste in sovrappiù rispetto a quelle strettamente necessarie per ottenere un determinato scopo in seguito all'introduzione di un metodo o di un processo più evoluto o più generale.

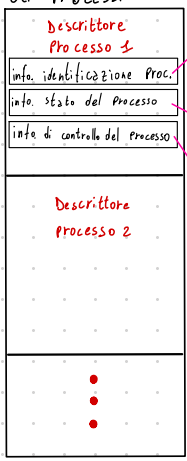




Process control block (PCB)

1. codice da eseguire (segmento codice)
2. I dati su cui operare (segment: dati)
3. stack di lavoro per la gestione di chiamate di funzione, passaggio di parametri e variabili locali
4. Insieme di attributi contenenti tutte le informazioni necessarie per la gestione del processo stesso. Incluse le info per descrivere i punti 1–3

Tab per la Gestione dei Processi

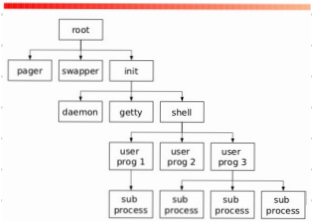


Informazioni di Identificazione del Processo:
- Process id, o PID.
- identificatori di altri processi collegati (esempio Pid del processo padre).
- id dell'utente che ha richiesto l'esecuzione del Processo.

Informazioni di stato del processo:
- registri generali del processore.
- registri speciali, come il program Counter e i registri di stato.

Informazioni di controllo del Processo:
- Informazioni di Scheduling,
  - stato del processo (in esecuzione, pronto, in attesa).
- Informazioni particolari necessarie dal particolare algoritmo di scheduling utilizzato.
  - Priorità, puntatori per la gestione delle code.
- Identificatore dell'evento per cui il Processo è in attesa.
- Informazioni di gestione della memoria
- Informazioni di accounting
  - Tempo di esecuzione del Processo
  - Tempo trascorso dall'attivazione di un Processo
- Info relative alle risorse
- Informazioni per InterProcess Communication (IPC)
  - stato di segnali, semafori, etc.

Gerarchia di processi: UNIX



Quando un processo crea un nuovo processo, il processo creante viene detto padre e il creato figlio. Si viene così a creare un albero di processi.
Semplificazione del procedimento di creazione di processi,
non occorre specificare esplicitamente tutti i parametri e le caratteristiche
Ciò che non viene specificato, viene ereditato dal padre.

Scheduler:

- Gestisce l'avvicendamento dei processi (decide quale processo deve essere in esecuzione ad ogni istante)
- Interviene quando viene richiesta un'operazione I/O e quando un'operazione I/O termina, ma anche periodicamente.

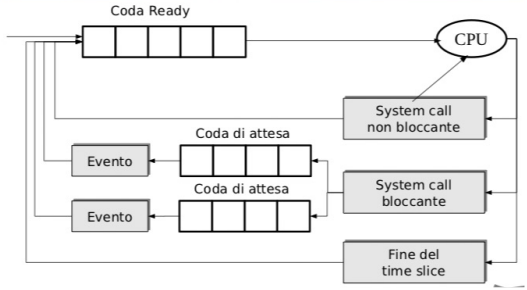
Mode switching e Context switching

- Tutte le volte che avviene un Interrupt (software o hardware) il processore è soggetto a un mode switching.
Modalità utente -> modalità supervisore
- Viene richiamato lo Scheduler
- Se lo Scheduler decide di eseguire un altro processo, il sistema è soggetto ad un Context Switching.

Context Switching

- Lo stato del processo viene salvato nel PCB.
- Lo stato del processo selezionato per l'esecuzione viene caricato dal PCB nel processore.

Vita di un processo nello scheduler



Eventi che possono causare un context switch

1. quando un processo passa da stato running a stato waiting (system call bloccante, operazione di I/O)
2. quando un processo passa dallo stato running allo stato ready (a causa di un interrupt)
3. quando un processo passa dallo stato waiting allo stato ready
4. quando un processo termina

Nota:

- nelle condizioni 1 e 4, l'unica scelta possibile è quella di selezionare un altro processo per l'esecuzione
- nelle condizioni 2 e 3, è possibile continuare ad eseguire il processo corrente

Tipologie Scheduler

Long-Term:

- Seleziona quali processi creare fra quelli che non hanno ancora iniziato la loro esecuzione
- Utilizzato per i programmi batch.
-nei sistemi interattivi (UNIX), non appena un programma viene lanciato il processo relativo viene automaticamente creato

Short-term:

- Seleziona quale dei processi pronti all'esecuzione deve essere eseguito, ovvero a quale assegnare il controllo della CPU

Mid-Term:

- E' anche possibile definire uno scheduler a medio termine che gestisca i processi bloccati per lunghe attese.
- Per questi processi l'immagine della memoria può venire copiata su disco al fine di ottimizzare l'uso della memoria centrale.

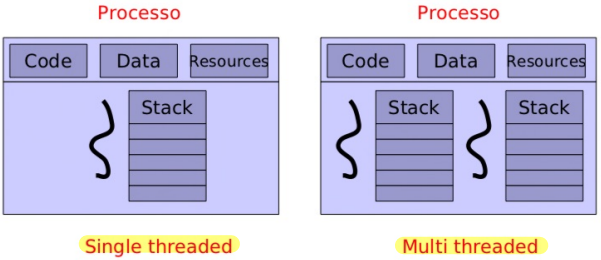
Processi e thread

Thread:

È una “linea di controllo” all’interno del processo (esecuzione di una singola sequenza di istruzioni).

Ogni thread possiede:
- la propria copia dello stato del processore
- il proprio program counter.
- uno stack separato.

I thread appartenenti allo stesso processo condividono:
- codice
- dati
- risorse di I/O



- i thread condividono lo spazio di memoria e le risorse allocate degli altri thread dello stesso processo
- condividere informazioni tra thread logicamente correlati rende più semplice l’implementazione di certe applicazioni

- allocare memoria e risorse per creare nuovi processi è costoso.
- fare context switching fra diversi processi è costoso.
- Gestire i thread è in generale più economico, quindi preferibile.
- creare thread all’interno di un processo è meno costoso.
- fare context switching fra thread è meno costoso.

Il S.O. Implementa i Thread in 2 modi

User thread
(A livello utente)

Gli user thread vengono supportati sopra il kernel e vengono implementati da una "thread library" a livello utente.
La thread library fornisce supporto per la creazione, lo scheduling e la gestione dei thread senza alcun intervento del kernel.

Pro:
l’implementazione risultante è molto efficiente
Cont:
se il kernel è single-threaded, qualsiasi user thread che effettua una chiamata di sistema bloccante (che si pone in attesa di I/O) causa il blocco dell’intero processo

Kernel thread
(A livello kernel)

I kernel thread vengono supportati direttamente dal sistema operativo.
La creazione, lo scheduling e la gestione dei thread sono implementati a livello kernel.

Pro:
poichè è il kernel a gestire lo scheduling dei thread, se un thread esegue una operazione di I/O, il kernel può selezionare un altro thread in attesa di essere eseguito.
Cont:
L’implementazione risultante è più lenta, perché richiede un passaggio da livello utente a livello supervisore.

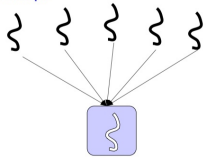
Molti sistemi supportano sia kernel thread che user thread, si vengono così a creare 3 differenti modelli di multithreading

Multithreading

Many-to-One

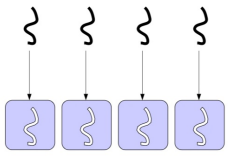
Un certo numero di user thread vengono mappati su un solo kernel thread

Modello generalmente adottato da s.o. che non supportano kernel thread multipli



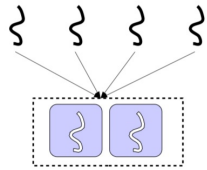
One-to-One

- Ogni user thread viene mappato su un kernel thread
- Può creare problemi di scalabilità per il kernel
- Supportato da Windows 95, OS/2



Many-to-Many

- Riassume i benefici di entrambe le architetture
- Supportato da Solaris, IRIX, Digital Unix



Tipi di Scheduler

Non-Preemptive o Cooperativo

- I Context switch avvengono solo quando un processo passa da stato running a stato waiting(system call bloccante, operazione I/O).
- Quando un processo termina.

In altre parole: il controllo della risorsa viene trasferito solo se l'assegnatario attuale lo cede volontariamente.
Non richiede alcuni meccanismi hardware come ad esempio timer programmabili.

Preemptive

- I Context switch possono avvenire in ogni momento.

In altre parole: è possibile che il controllo della risorsa venga tolto all'assegnatario attuale a causa di un evento.
Permette di utilizzare al meglio le risorse.

Criteri di scelta di uno Scheduler

Utilizzo della risorsa (CPU)

- Percentuale di tempo in cui la CPU è impegnata ad eseguire processi
- Deve essere massimizzato

Tempo di tounaround

- Tempo che intercorre dalla sottomissione di un processo alla sua terminazione.
- Deve essere minimizzato.

Tempo di attesa

- Tempo che intercorre fra la sottomissione di un processo e il tempo di prima risposta.
- Particolarmente significativo nei programmi interattivi, deve essere minimizzato.

Throughput

- Numero di processi completati per unità di tempo
- Dipende dalla lunghezza dei processi
- Deve essere massimizzato.

Tempo di risposta

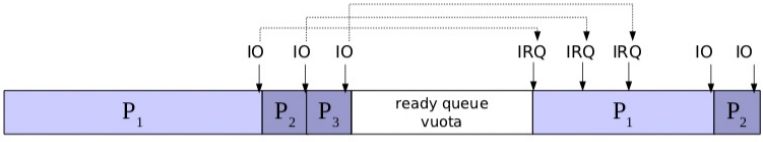
- Il tempo trascorso da un processo nella coda ready.
- Deve essere minimizzato.

Durante l'esecuzione di un processo si alternano periodi di attività svolti dalla CPU (CPU burst) e periodi di attività di I/O (I/O burst).
I processi caratterizzati da CPU burst molto lunghi si dicono (CPU bound), mentre se caratterizzati da I/O burst molto lunghi si dicono (I/O bound).

Burst = periodo di attività.
Bound = periodo di attività molto lungo.

First Come, First Served (FCFS)

- **Supponiamo di avere**
  - un processo CPU bound
  - un certo numero di processi I/O bound
  - i processi I/O bound si "mettono in coda" dietro al processo CPU bound, e in alcuni casi la ready queue si puo svuotare
- *Convoy effect*



Convoy Effect
FCFS è un algoritmo non-preemtive per natura, una volta che del tempo di CPU è assegnato ad un processo, gli altri processi devono attendere la sua terminazione prima di ricevere a loro volta CPU time.

1. First Come, First Served (FCFS)

- **Algoritmo**
  - il processo che arriva per primo, viene servito per primo
  - politica senza preemption
- **Implementazione**
  - semplice, tramite una coda (politica FIFO)
- **Problemi**
  - elevati tempi medi di attesa e di turnaround
  - processi CPU bound ritardano i processi I/O bound

Scheduling

- **Algoritmi:**
  1. First Come, First Served
  2. Shortest-Job First
     - 2.1 Shortest-Next-CPU-Burst First
     - 2.2 Shortest-Remaining-Time-First
  3. Round-Robin

2.1 Shortest Job First (SJF)

- **Algoritmo (Shortest Next CPU Burst First)**
  - la CPU viene assegnata al processo ready che ha la minima durata del CPU burst successivo
  - politica senza preemption

Shortest Job First (SJF)

- **L'algoritmo SJF**
  - è *ottimale* rispetto al tempo di attesa, in quanto è possibile dimostrare che produce il minor tempo di attesa possibile
  - ma è *impossibile da implementare* in pratica!
  - è possibile solo fornire delle *approssimazioni*
- **Negli scheduler long-term**
  - possiamo chiedere a chi sottomette un job di predire la durata del job
- **Negli scheduler short-term**
  - non possiamo conoscere la lunghezza del *prossimo* CPU burst.... ma conosciamo la lunghezza di quelli *precedenti*

2.2 Shortest Job First (SJF) Shortest-Remaining-Time First

- **Shortest Job First "approssimato" esiste in due versioni:**
  - *non preemptive*
    - il processo corrente esegue fino al completamento del suo CPU burst
  - *preemptive*
    - il processo corrente può essere messo nella coda ready, se arriva un processo con un CPU burst più breve di quanto rimane da eseguire al processo corrente
    - "Shortest-Remaining-Time First"

3 Scheduling Round-Robin

- **E' basato sul concetto di quanto di tempo (o time slice)**
  - un processo non può rimanere in esecuzione per un tempo superiore alla durata del quanto di tempo
- **Implementazione (1)**
  - l'insieme dei processi pronti è organizzato come una coda
  - due possibilità:
    - un processo può lasciare il processore *volontariamente*, in seguito ad un'operazione di I/O
    - un processo può *esaurire il suo quanto di tempo* senza completare il suo CPU burst, nel qual caso viene aggiunto in fondo alla coda dei processi pronti
  - in entrambi i casi, il prossimo processo da esegure è il primo della coda dei processi pronti

La durata del quanto di tempo è un parametro critico del sistema

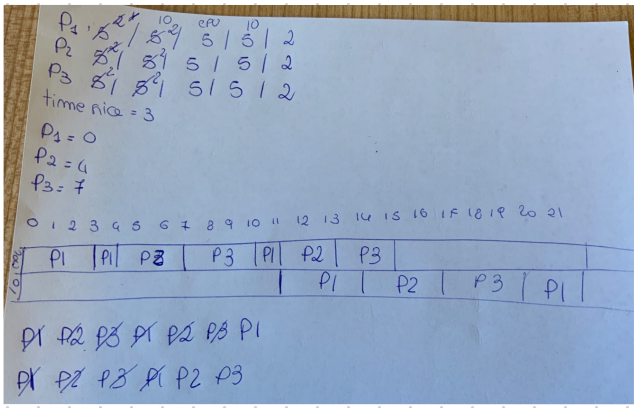
- se il quanto di tempo è breve, il sistema è meno efficiente perchè deve cambiare il processo attivo più spesso
- se il quanto è lungo, in presenza di numerosi processi pronti ci sono lunghi periodi di inattività di ogni singolo processo,
  - in sistemi interattivi, questo può essere fastidioso per gli utenti

Implementazione (2)

- è necessario che l'hardware fornisca un timer (interval timer) che agisca come "sveglia" del processore
- il timer è un dispositivo che, attivito con un preciso valore di tempo, è in grado di fornire un interrupt allo scadere del tempo prefissato
- il timer viene interfacciato come se fosse un'unita di I/O

- **Round-robin**
  - fornisce le stesse possibilità di esecuzione a tutti i processi
- **Ma i processi non sono tutti uguali:** !!
  - usando round-robin puro la visualizzazione dei un video MPEG potrebbe essere ritardata da un processo che sta smistando la posta
  - la lettera può aspettare ½ sec, il frame video NO

Per questo motivo si usa lo scheduling a priorità.



Esercizio esempio sullo scheduling.

Scheduling a priorità

- Ogni processo è associato ad una specifica priorità.
- Lo scheduler sceglie il processo pronto con priorità più alta.

Priorità definite da:

Definite dal sistema operativo

- Vengono utilizzate una o più quantità misurabili per calcolare la priorità di un processo.
- Ad esempio SJF è un sistema basato su priorità.

Definite esternamente

- Le priorità non vengono definite dal sistema operativo, ma vengono imposte dal livello utente.

Tipologie:

Priorità statica

- La priorità non cambia durante la vita di un processo.
- Problema: processi a bassa priorità possono essere messi in starvation da processi ad alta priorità.

Priorità dinamica

- La priorità può variare durante la vita di un processo.
- È possibile utilizzare metodologie di priorità dinamica per evitare starvation.

Priorità basata su aging

- Tecnica che consiste nell'incrementare gradualmente la priorità dei processi in attesa
- Nessun processo rimarrà in attesa per un tempo indefinito perchè prima o poi raggiungerà la priorità massima.

Scheduling a classi di priorità

- È possibile creare diverse classi di processi con caratteristiche simili e assegnare ad ogni classe specifiche priorità.
- La coda ready viene scomposta in molteplici "sottocode", una per ogni classe di processi.

Uno scheduler a classi di priorità seleziona il processo da eseguire fra quelli pronti della classe a priorità massima che contiene processi.

All'interno di ogni classe di processi, è possibile utilizzare una politica specifica adatta alle caratteristiche della classe.

Esempio, di SCHEDULING MULTILIVELLO, di quattro classi di processi(priorità decrescente):
- Processi server (priorità statica)
- Processi interattivi (Round Robin)
- Altri processi utente (FIFO)
- Il processo vuoto (FIFO banale)

Scheduling Real-Time

- La correttezza dell'esecuzione non dipende solamente dal valore del risultato, ma anche dall'istante temporale nel quale il risultato viene emesso.

Hard Real-Time

- le deadline di esecuzione dei programmi non devono essere superate in nessun caso.
Es. sistemi di controllo di veicoli, centrali nucleari

Soft Real-Time

- Errori occasionali sono tollerabili
Es. Ricostruzioni di segnali audio-video

Tipologia processi

Processi periodici

- Sono periodici i processi che vengono riattivati con una cadenza regolare(periodo)
Es. controllo assetto dei velivoli, basato sulla rilevazione periodica dei parametri di volo.

Processi aperiodici

- I processi che vengono scatenati da un evento sporadico, ad esempio l'allarme di un rilevatore di pericolo.

Le risorse possono essere divise in classi, Due risorse appartenenti alla stessa classe sono equivalenti per il processo.

Risorse
Possono essere suddivise in classi

Istanze
Risorse di una classe

Molteplicità
Numero di risorse di una classe

Risorse ad assegnazione:

Statica
Avviene al momento della creazione del processo e rimane valida fino alla terminazione

Dinamica
I processi:
- Richiedono le risorse durante la loro esistenza
- Le utilizzano una volta ottenute
- Le rilasciano quando non più necessarie(eventualmente alla terminazione del processo)

Richiesta:

Singola
Si riferisce a una singola risorsa di una classe definita.
È il caso normale.

Multipla
Si riferisce ad una o più classi, e per ogni classe, ad una o più risorse.
Deve essere soddisfatta integralmente.

Bloccante
Il processo richiedente si sospende se non ottiene immediatamente l'assegnazione.

Non Bloccante
La mancata assegnazione viene notificata al processo richiedente, senza provocare la sospensione.

Tipi di risorse:

Risorse seriali ("non condivisibili")
Una singola risorsa non può essere assegnata a più processi contemporaneamente.
Es. stampanti, sezioni critiche, i processori.

Risorse non seriali ("condivisibili")
Es. file di sola lettura

Risorse Prerilasciabili ("preemptable")
Una risorsa si dice prerilasciabile se la funzione di gestione può sottrarla ad un processo prima che questo l'abbia effettivamente rilasciata.
- Il processo che subisce il pre rilascio deve sospendersi.
- La risorsa prerilasciata sarà successivamente restituita al processo.

Una risorsa è prerilasciabile se il suo stato non si modifica durante l'utilizzo oppure il suo stato può essere facilmente salvato e ripristinato.
Es. processore.

Risorse Non Prerilasciabili ("non-preemptable")
La funzione di gestione non può sottrarle al processo al quale sono assegnate.
Sono non Prerilasciabili le risorse il cui stato non può essere salvato e ripristinato.
Es. stampanti, classi di sezioni critiche, partizioni di memoria(nel caso di gestione statica)

Deadlock nelle risorse

Deadlock quando:
- risorse non condivisibili.
- risorse non prerilasciabili
- richiesta bloccante
- attesa circolare

Algoritmo di "Ostrich"
L'algoritmo opera come se il deadlock non esistesse.
(Mettiamo la testa sotto la sabbia).

Gestione del deadlock tramite:

Deadlock detective:
- Utilizzare il grafo di Holt al fine di riconoscere gli stati Deadlock.
- Mantenere aggiornato il grafo di Holt, registrando su di esso tutte le assegnazioni e le richieste di risorse.

Deadlock prevention
Per evitare Deadlock si eliminano una delle 4 condizioni del Deadlock.
Il Deadlock viene eliminato strutturalmente.

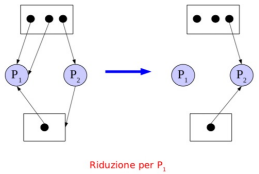
Deadlock avoidance
Mantiene l'attribuzione di risorse in stato SAFE, nel quale abbiamo garanzia che in futuro non possa avvenire deadlock.
Es. questa operazione che sto per eseguire potrà portare ad un Deadlock?

Utilizziamo l'algoritmo del banchiere

Avviene Deadlock quando si presentano:
- Mutua esclusione
- Accumulo incrementale (i processi che possiedono almeno una risorsa devono attendere prima di richiederne altre, già allocate ad altri processi)
- Impossibilità di prelazione (solo il processo che detiene la risorsa può rilasciarla)
- Attesa circolare (processo P0 aspetta una risorsa da P1 che aspetta una risorsa da P2 che a sua volta aspetta una risorsa da P0)

Diagramma di Holt
Il diagramma delle attese(diagramma di Holt) è un grafo orientato diretto, utilizzato per rappresentare gli stati di allocazione tra risorse e processi.

Esempio di riduzione



Riduzione per P_1

Un grafo di Holt è riducibile se esiste almeno un nodo processo con solo archi entranti.

Lo stato non è di Deadlock se e solo se il grafo di Holt è completamente riducibile.

Deadlock Recovery

Checkpoint
Lo stato dei dischi viene periodicamente salvato su disco.

Rollback
In caso di Deadlock, si ripristina uno o più processi ad uno stato precedente, fino a quando il deadlock non scompare

Soluzione

Manuale
L'operatore viene informato ed eseguirà alcune azioni che permettono al sistema di proseguire

Automatica
Il sistema operativo è dotato di meccanismi che permettono di risolvere in modo automatico la situazione, in base ad alcune politiche

Terminazione Totale
(tutti i processi coinvolti vengono terminati)

Terminazione Parziale
(Viene terminato un processo alla volta, fino a quando il Deadlock non scompare)

Preemption
(Una risorsa (o più di una, se necessario) viene sottratta ad uno dei processi coinvolti nel Deadlock)

Algoritmo del Banchiere

C = quello che il processo potrebbe chiedere al massimo
L = quello che il processo effettivamente richiede
N = quello che il processo potrebbe ancora chiedere (C - L = N)

$Avaible_{[P1]} = COH = 0,2$
$Avaible_{[P2]} = Avaible_{[P1]} + L_{[P1]} = 0;2 + 5,3 = 5;5$

$Avaible_{[P3]} = Avaible_{[P2]} + L_{[P2]} = 5;5 + 6;2 = 11;7$

COH = 0,2 // esempio 0 euro e 2 dollari

	C	L	N	AVAIL
p1	10,5	5,3	5,2	0;2
p2	8,4	6,2	2,2	5;5
p3	4,4	4,2	0,2	11;7

5;2 > 0;2

Deve!
N < Avaible

Controllo se AVAIL (a quel passo) >= N(p1)|N(p2)|N(p3)

	C	L	N	AVAIL	
p3	4,4	4,2	0,2	0,2	
p2	8,4	6,2	2,2	4,4	//AVAIL(p2) = (L(p3)+AVAIL(p3))
p1	10,5	5,3	5,2	10,6	

//SAFE

se ordiniamo dal più piccolo al più grande le richieste massime dei processi generalmente lo stato del sistema torna SAFE

Memory Manager

Gestisce la memoria principale del OS.
Tiene traccia della memoria libera e occupata.
Alloca memoria ai processi e dealloca quando non più necessaria.

Binding

Associazione di indirizzi di memoria ai dati e alle istruzioni di un programma.

Può avvenire durante:

Durante la compilazione:

Pro:
- Non richiede hardware speciale.
- Semplice e veloce.

Cont:
- Non funziona con la multiprogrammazione

Durante il caricamento:

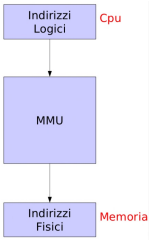
Il codice generato dal compilatore non contiene indirizzi assoluti, ma relativi (al PC oppure ad un indirizzo base).
Questo tipo di codice viene detto Rilocabile.
Durante il caricamento il Loader si preoccupa di aggiornare tutti i riferimenti agli indirizzi di memoria coerentemente al punto iniziale di caricamento.

Pro:
- No richiede hardware speciale.
- Si multiprogrammazione

Cont:
- Richiede una traduzione degli indirizzi da parte del loader, e quindi formati particolari dei file eseguibili.

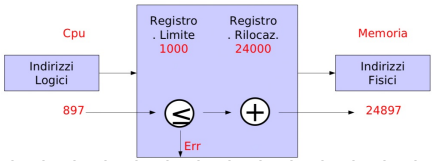
Durante l'esecuzione:

L'individuazione dell'indirizzo di memoria effettivo viene effettuata durante l'esecuzione da un componente hardware specifico: la MEMORY MANAGEMENT UNIT(MMU), da non confondere con il memori manager(MM)



la MMU opera come una funzione di traduzione da indirizzi logici a indirizzi fisici

Il registro limite viene utilizzato per implementare meccanismi di protezione della memoria.



Linking

Statico

Il linker risolve tutti i riferimenti dei programmi

Dinamico

Posticipare il linking delle routine di libreria al momento del primo riferimento durante l'esecuzione.
Risparmio di memoria.

Allocazione

consiste nel reperire ed assegnare uno spazio di memoria fisica a un programma che viene attivato oppure soddisfare ulteriori richieste di uno già attivo

Allocazione Contigua

Celle di memoria consecutive

Allocazione NON Contigua

Celle di mem. non consecutive

Statica

Un programma deve mantenere la propria area di memoria dal caricamento alla terminazione

Dinamica

Durante l'esecuzione un programma può essere spostato all'interno della memoria

A Partizioni Fisse

- È statica e contigua
- Semplice
- Spreco di memoria, grado di parallelismo limitato dal numero di partizioni

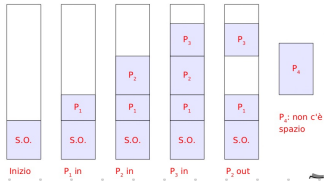
Se un processo occupa una dimensione inferiore a quella della partizione che lo contiene, lo spazio non utilizzato è sprecato. Questo spazio inutilizzato si chiama:

Frammentazione interna

A Partizioni Dinamiche

la memoria disponibile (nella quantità richiesta) viene assegnata ai processi che ne fanno richiesta. Nella memoria possono essere presenti diverse zone inutilizzate

- Statica e Contigua



Dopo un certo numero di allocazioni e deallocazioni di memoria dovute all'attivazione e alla terminazione dei processi lo spazio libero appare suddiviso, è il fenomeno della

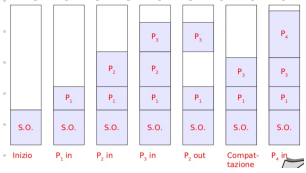
Frammentazione Esterna

Compattazione

Compattare la memoria significa spostare in memoria tutti i programmi in modo da riunire tutte le aree inutilizzate

Operazione volta a risolvere la Frammentazione Esterna, ma è molto onerosa (data fisicamente in memoria grandi quantità di dati).

I processi devono essere fermi durante la compattazione, NON può essere utilizzata in sistemi Interattivi



Allocazione Dinamica

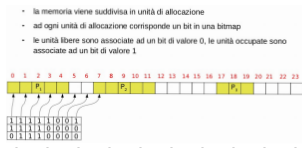
È necessaria una struttura dati che mantenga le informazioni sulle zone libere e sulle zone occupate in memoria

MAPPE DI BIT

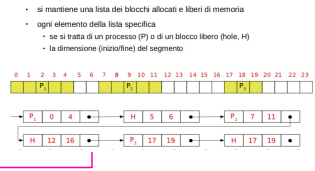
Pro: - struttura dati ha una dimensione fissa e calcolabile a priori

Cont: - Per individuare uno spazio di memoria di dimensione k unità, è necessario cercare una sequenza di k bit 0 consecutivi
- in generale, tale operazione è $O(m)$, dove m rappresenta il numero di unità di allocazione.

MAPPE di Bit



Liste di Puntatori



LISTE DI PUNTATORI

Pro: - Costo di deallocazione $O(1)$, è possibile ottimizzare il costo di allocazione:
- mantenendo una lista di blocchi liberi separata.
- ordinando tale lista per dimensione.

Cont: - Dove mantenere queste informazioni?:
- Per i blocchi occupati: ad esempio nella tabella dei processi
- Per i blocchi liberi: nei blocchi stessi
- è richiesta un unità minima di allocazione

- **First Fit**: 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 invece di partire dall'inizio, parte dal punto dove si era fermato all'ultima allocazione. Prestazioni peggiori di First Fit.
- **Best Fit**: Seleziona il blocco più piccolo fra i blocchi liberi presenti in memoria.
- **Worst fit**: Seleziona il più grande fra i blocchi presenti in memoria

Paginazione

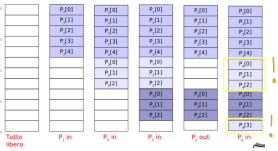
Memoria fisica suddivisa in un insieme di blocchi della stessa dimensione delle pagine, chiamati *frame*

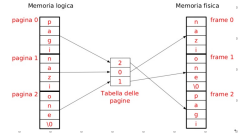
Approccio contemporaneo
- Riduce il fenomeno di frammentazione interna
- Minimizza (elimina il fenomeno della frammentazione esterna)
- NECESSITÀ DI HARDWARE ADEGUATO

Spazio di indirizzamento logico di un processo suddiviso in blocchi fissi chiamate *pagine*

Dimensione delle pagine?
- Potenza di 2 per facilitare trasformazione da indirizzo logico a fisico
- No troppo piccole, tabella delle pagine cresce di dimensioni
- No troppo grande, spazio di memoria per frammentazione interna è considerevole

Allocazione di un processo in memoria: vengono reperiti ovunque in memoria un numero sufficiente di frame per contenere le pagine del processo



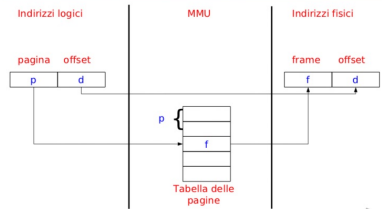


Dove mettere la tabella delle pagine?

Soluzione 1:
Registri dedicati
- Sono contenuti in registri ad alta velocità nella MMU (o della CPU)
- Problema: troppo costoso

Soluzione 2:
Totalmente in memoria
- Problema: il numero di accessi in memoria verrebbe raddoppiato; ad ogni riferimento bisognerebbe prima accedere alla tabella delle pagine, poi al dato

Supporto hardware (MMU) per paginazione

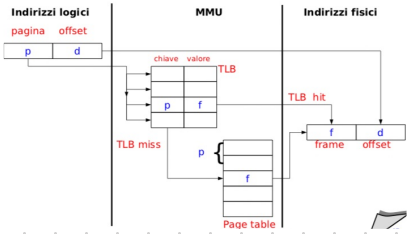


TLB (Translation lookaside buffer)

- insieme di registri associativi ad alta velocità
- ogni registro è suddiviso in 2 parti: una chiave ed un valore

OPERAZIONE DI LOOKUP:
- Viene richiesta la ricerca di una chiave
- La chiave viene confrontata simultaneamente con tutte le chiavi presenti nel buffer
- Se la chiave è presente (TLB Hit), si ritorna il valore corrispondente
- Se la chiave non è presente (TLB Miss), si utilizza la tabella in memoria

Translation lookaside buffer (TLB)



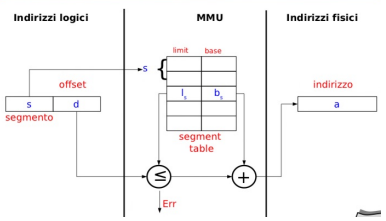
L'hardware per il TLB è costoso.

Il Translation Lookaside Buffer (TLB) è un buffer, cioè una memoria tampone (o, nelle implementazioni più sofisticate, una cache nella CPU), che l'MMU (Memory Management Unit) usa per velocizzare la traduzione degli Indirizzi Virtuali. Il TLB possiede un numero fisso di elementi della Page Table, la quale viene usata per mappare gli Indirizzi Virtuali in Indirizzi Fisici. La Memoria virtuale è lo spazio visto da un processo che può essere più grande della memoria fisica (reale). Questo spazio è catalogato in pagine di dimensioni prefissate. Generalmente solo alcune pagine vengono caricate nella memoria fisica in zone dipendenti dalla politica di Page Replacement. La Page Table (generalmente caricata in memoria) tiene traccia di dove le pagine virtuali sono caricate nella memoria fisica. Il TLB è una cache della Page Table, cioè solamente un sottoinsieme del suo contenuto viene memorizzato.

Segmentazione

- Lo spazio di indirizzamento logico è dato da un insieme di segmenti
- Un segmento è un'area di memoria (logicamente continua) contenente elementi tra loro affini
- Ogni segmento è caratterizzato da un nome (normalmente un indice) e da una lunghezza.

Supporto hardware per segmentazione



Segmentazione vs Paginazione

Paginazione	Segmentazione
la divisione in pagine è automatica.	la divisione in segmenti spetta al programmatore.
le pagine hanno dimensione fissa	i segmenti hanno dimensione variabile
le pagine possono contenere informazioni disomogenee (ad es. sia codice sia dati)	un segmento contiene informazioni omogenee per tipo di accesso e permessi di condivisione
una pagina ha un indirizzo	un segmento ha un nome.
dimensione tipica della pagina: 1-4 KB	dimensione tipica di un segmento: 64KB - 1MB

Problemi:

- Allocare segmenti di dimensione variabile è del tutto equivalente al problema di allocare in modo contiguo la memoria dei processi
- È possibile utilizzare tecniche di allocazione dinamica (es. First fit), compattazione, ma così torniamo ai problemi precedenti.

Segmentazione + Paginazione

Ogni segmento viene suddiviso in pagine che vengono allocate in frame liberi della memoria (non necessariamente contigui).

La MMU deve avere il supporto hardware sia per la segmentazione che per la paginazione

Benefici di entrambi i metodi.

Memoria Virtuale

Definizione

- Tecnica per eseguire processi che non sono completamente in memoria
- Permette di eseguire in concorrenza processi che nel loro complesso (o anche singolarmente) hanno necessità di memoria maggiore di quella disponibile.
- Può diminuire le prestazioni di un sistema se implementata male.

Implementazione

- Ogni processo ha accesso ad uno spazio di indirizzamento virtuale che può essere più grande di quello fisico.
- Gli indirizzi virtuali:
  - mappati su indirizzi fisici della memoria principale
  - mappati su memoria secondaria
- in caso di accesso ad indirizzi virtuali mappati in mem. second.:
  - dati associati trasferiti in mem. principale
  - se la memoria è piena, si sposta in mem. sec. i dati contenuti in mem. princ. che sono considerati meno utili

Pager: Componente che si occupa di caricare la pagina mancante in memoria e di aggiornare la tabella delle pagine.

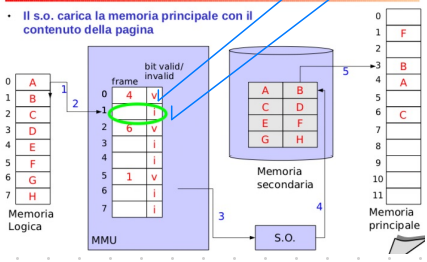
Demand Paging

(v, per valid) che indica se la pagina è presente in memoria.

Processo tenta di accedere ad una pagina non in memoria, il processore genera un trap (page fault)

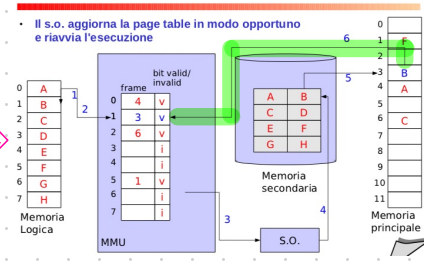
Gestione dei page fault

- Il s.o. carica la memoria principale con il contenuto della pagina



Gestione dei page fault

- Il s.o. aggiorna la page table in modo opportuno e riavvia l'esecuzione



SWAP:

Azione di copia dell'intera area di memoria usata da un processo

Tecnica usata prima del demand paging

- dalla mem. sec. alla mem. princ. (swap-in)
- dalla mem. princ. alla mem. sec. (swap-out)

Algoritmi di Gestione del Page Fault

La pagina vittima deve essere la meno utile

Algoritmi di rimpiazzamento

Vengono valutati applicando una stringa di riferimento in memoria

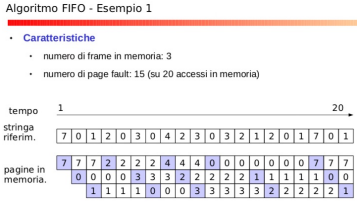
Anomalia di Belady

In alcuni algoritmi non è detto che aumentando il numero di frame il numero di page fault diminuisca.

FIFO

Quando c'è necessità di liberare un frame viene individuato come "vittima" il frame che per primo fu caricato in memoria

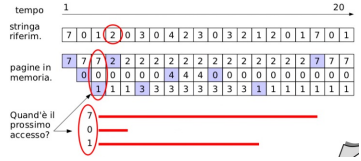
Vantaggi:
semplice, non richiede particolari supporti hardware
Svantaggi:
vengono talvolta scaricate pagine che sono sempre utilizzate



MIN - ottimale

- Seleziona come pagina vittima una pagina che non sarà più acceduta o la pagina che verrà acceduta nel futuro più lontano

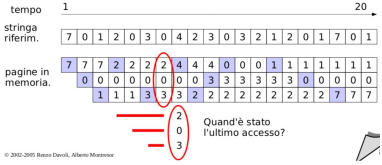
- È ottimale perché fornisce il minimo numero di Page Fault, ma è un algoritmo teorico perché richiederebbe la conoscenza a priori della stringa dei riferimenti futuri del programma.
- Viene utilizzato a posteriori come paragone per verificare le performance degli algoritmi di rimpiazzamento reali



la più vecchia

LRU (Last Recently Used)

- Seleziona come pagina vittima la pagina che è stata usata meno recentemente nel passato.
- È basato sul presupposto che la distanza tra due riferimenti successivi alla stessa pagina non vari eccessivamente.
- Stima la distanza nel futuro utilizzando la distanza nel passato.



Implementazione basata su stack

Si mantiene uno stack di pagine, tutte le volte che una pagina viene acceduta viene rimossa dallo stack (se presente) e posta in cima allo stack stesso.

- In cima la pagina utilizzata più recentemente.
- In fondo la pagina utilizzata meno recentemente.

Problema:

- È necessario uno specifico supporto hardware, la MMU deve registrare nella tabella delle pagine un time-stamp quando accede ad una pagina e può essere implementato come un contatore che viene incrementato ad ogni accesso in memoria.
- Questo comporta accessi addizionali

Approssimare LRU (Reference bit)

- L'implementazione a stack e a contatori è troppo costosa.
- Supporto sotto forma di Reference bit
- Tutte le volte che una pagina è acceduta, il bit associato alla pagina viene aggiornato ad 1.
  Però in questo modo non conosciamo l'ordine in cui sono state usate
- Possiamo utilizzare queste informazioni per approssimare LRU
- Vengono salvati i reference bit ad intervalli regolari, il nuovo valore del Ref. bit viene salvato tramite shift a destra (presupponiamo 8 bit di "storia" per ogni pagina) della storia ed inserimento del bit come most significant bit.
- La pagina vittima è quella con valore minore; in caso di parità si usa una disciplina FIFO

Approssimare LRU (Second-chance)

- Le pagine in memoria vengono gestite come una lista circolare, è un caso particolare del Reference bit dove la "storia" è uguale ad 1.
- A partire dalla posizione successiva all'ultima pagina caricata, si scandisce la lista con la seguente regola:
  - pagina già acceduta (reference bit a 1) → il reference bit viene messo a 0 (zero)
  - la pagina non è stata acceduta (reference bit a 0) → la pagina selezionata è la vittima.

Altri Algoritmi di Rimpiazzamento

LFU (Least Frequently Used)

- Si mantiene un contatore del numero di accessi ad una pagina.
- La frequenza è il valore del contatore diviso per il "tempo" di permanenza in memoria.
- La pagina con il valore minore viene scelta come vittima
- Una pagina utilizzata spesso dovrebbe avere un valore molto alto
- Implementazione: tramite reference bit.
- Problemi: una pagina utilizzata frequentemente all'inizio, e non viene più utilizzata, non viene rimossa per lunghi periodi di tempo.

MFU (Most Frequently used)

- Si mantiene un contatore del numero di accessi ad una pagina.
- La pagina con il valore maggiore viene scelta come vittima
- Pagine appena caricate hanno un valore molto basso, e non dovrebbero essere rimosse.
- Implementazione: tramite Reference bit.
- Problemi: di performance

Allocazione

Algoritmo di Allocazione (per memoria virtuale)
Si intende l'algoritmo utilizzato per scegliere quanti frame assegnare ad ogni singolo processo.

Allocazione Locale

- Ogni processo ha un insieme proprio di frame
- Poco flessibile

Allocazione Globale

- Tutti i processi possono allocare tutti i frame presenti nel sistema (sono in competizione)
- Può portare Trashing

Trashing

Un processo (o un sistema) si dice che è in trashing quando spende più tempo per la paginazione che per l'esecuzione.

Cause:
- In un sistema con allocazione globale, si ha trashing se i processi tendono a "rubarsi i frame a vicenda"
- Non riescono a tenere in memoria i frame utili a breve termine (perché altri processi chiedono frame liberi) e quindi generano page fault ogni pochi passi di avanzamento.

Esempio

- esaminiamo un sistema che accetti nuovi processi quando il grado di utilizzazione della CPU è basso
- se per qualche motivo gran parte dei processi entrano in page fault:
  - la ready queue si riduce
  - il sistema sarebbe indotto ad accettare nuovi processi....
  - E' UN ERRORE!
- statisticamente, il sistema:
  - genererà un maggior numero di page fault
  - di conseguenza diminuirà il livello della multiprogrammazione

Working Set

Definizione

Working Set definisce la quantità di memoria che un processo richiede in un certo intervallo di tempo.

Si definisce **working set di finestra Δ** l'insieme delle pagine accedute nei più recenti Δ riferimenti

Considerazioni

- È una rappresentazione approssimata del concetto di località
- Se una pagina non compare in Δ riferimenti successivi in memoria, allora esce dal working set; non è più una pagina su cui si lavora attivamente

A cosa serve

- Se l'ampiezza della finestra è ben calcolata, il working set è una buona approssimazione dell'insieme delle pagine "utili".
- Sommando quindi l'ampiezza di tutti i working set dei processi attivi, questo valore deve essere sempre minore del numero dei frame disponibili.
- Altrimenti il sistema è in Trashing.

Se si sceglie Δ troppo piccolo

- Si considera non più utile ciò che in realtà serve
- Si sottovaluta il numero di pagine necessarie per il processo
- Falsi negativi di trashing

Se si sceglie Δ troppo grande

- Si considera utile anche ciò che non serve più.
- Si sopravvaluta il numero di pagine necessarie
- Falsi positivi di trashing

Come si usa

- Serve per controllare l'allocazione dei frame ai singoli processi.
- Quando ci sono sufficienti frame disponibili non occupati dai working set dei processi attivi, allora si può attivare un nuovo processo.
- Se al contrario la somma totale dei working set supera il numero totale dei frame, si può decidere di sospendere l'esecuzione di un processo.

Esempio: Δ = 5

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
{ 0,1,2,7 }												{ 0,1,2 }							

Progettazione del sistema di I/O

Tecniche di gestione dei dispositivi I/O

Buffering

- Per gestire una differenza di velocità tra il produttore ed il consumatore di un certo flusso di dati.
- Per gestire la differenza di dimensioni nell'unità di trasferimento.
- Per implementare la "semantica di copia" delle operazioni di I/O.

Caching

- Mantiene una copia in memoria primaria di informazioni che si trovano in memoria secondaria.
- È differente da buffering:
  - nel buffer si trova l'unica istanza di un'informazione.
  - la cache mantiene la copia di un'informazione.

Spooling

- È un buffer che mantiene output per un dispositivo che non può accettare flussi di dati distinti.
- Ad esempio stampanti.

I/O scheduling

Struttura di un disco

r: velocità di rotazione, espressa in rpm.

T_s: tempo di seek, ovvero il tempo medio necessario affinché la testina si sposti sulla traccia desiderata.

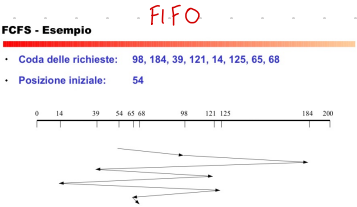
V_r: velocità di trasferimento, espressa in byte al secondo.

Ritardo Rotazionale:
- È il tempo medio necessario affinché il settore desiderato arrivi sotto la testina.
- È uguale a $1/2r$.

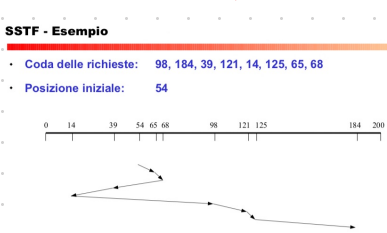
Transfert time:
- Dipende dalla quantità di dati b da leggere (supponendo che siano contigui sulla stessa traccia)
- È uguale a b/V_r

Algoritmi di lettura/scrittura

FIFO



Shortest Seek Time First

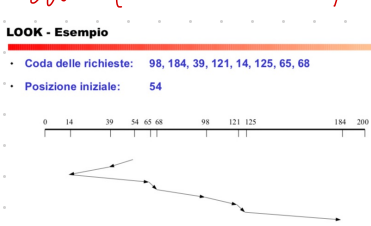


- Può provocare Starvation

Però la testina non si muove per lunghi periodi di tempo, ad esempio un certo numero di processi legge dallo stesso cilindo.

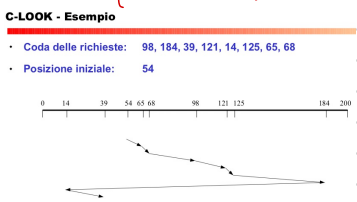
Soluzione: suddividere la coda delle richieste in 2 sottocode separate, quando vengono risolte le richieste di una, quelle che arrivano vengono inserite nell'altra, dopo che sono state risolte tutte, si passa a risolvere quest'ultima.

LOOK (o anche chiamato SCAN)

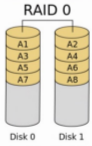
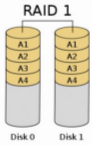
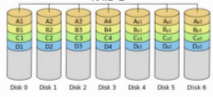
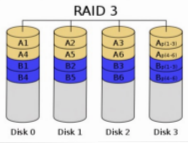
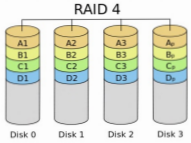
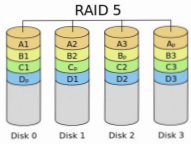
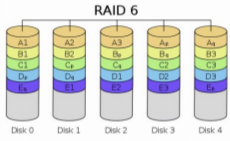


- la testina si muove in una direzione, quando viene raggiunta l'ultima richiesta nella direzione scelta, la direzione si inverte.
- Sono privilegiate le tracce centrali

C-LOOK (o anche chiamato C-SCAN)



- Uguale a LOOK, ma la scansione del disco avviene in una sola direzione
- Quando si raggiunge l'ultima richiesta, la testina si sposta direttamente alla prima richiesta

Livello	Numero minimo di dischi	Capacità	Numero massimo consentito di dischi difettosi	Schema	Vantaggi	Svantaggi
RAID 0	2	$C^1+C^2+..+C^n$	0		Costo di implementazione basso; alte prestazioni in scrittura e lettura grazie al parallelismo delle operazioni I/O dei dischi concatenati.	Impossibilità d'utilizzo di dischi hot-spare; affidabilità drasticamente ridotta, anche rispetto a quella di un disco singolo.
RAID 1	2	C	N / 2		Affidabilità, cioè resistenza ai guasti, che aumenta linearmente con il numero di copie; velocità di lettura (in certe implementazioni e sotto certe condizioni).	Bassa scalabilità; costi aumentati linearmente con il numero di copie; velocità di scrittura ridotta a quella del disco più lento dell'insieme.
RAID 2	7	C × 7				
RAID 3	3	C × (N - 1)	1			
RAID 4	3	C × (N - 1)	1		Resistenza ai guasti; letture veloci grazie al parallelismo della struttura; possibilità di inserire dischi hot-spare.	Il disco utilizzato per la parità può costituire il collo di bottiglia del sistema; scrittura lenta a causa della modifica e del calcolo della parità (4 accessi a disco per ogni operazione I/O).
RAID 5	3	C × (N – 1)	1		La parità è distribuita e quindi non esiste il problema del disco collo di bottiglia come nel RAID 4; Le letture e le scritture sono più veloci rispetto allo stesso RAID 4 (perché il disco che nel RAID 4 è dedicato alla parità ora può essere utilizzato per le letture parallele).	Scritture lente a causa della modifica e del calcolo della parità (4 accessi a disco per ogni operazione I/O), ma è comunque più veloce del singolo disco ed aumenta all'aumentare del numero di dischi. Su un controller P410, un raid 5 composto da 5 dischi da 10.000 RPM, la lettura/scrittura sequenziale è paragonabile a quella di un SSD; con 10 dischi è più del doppio.
RAID 6	4	C × (N – 2)	2		Altissima fault tolerance grazie alla doppia ridondanza.	Scritture molto lente a causa della modifica e del calcolo della parità (6 accessi a disco per ogni operazione I/O), necessari N+2 dischi, molto costoso economicamente a causa della ridondanza e della complessità del controller della struttura. Problema del Write Hole. Le write sui diversi dispositivi non sono atomiche nell'insieme: questo vuol dire che la mancanza di alimentazione durante una scrittura può portare alla perdita di dati. Ad esempio, con un sistema con 3 dischi in raid 5, se si volesse modificare il blocco n si dovrebbero fare queste operazioni: lettura del blocco n-1, lettura della parità, calcolo della nuova parità, scrittura del blocco n, e scrittura della parità. Il problema si ha nel caso in cui venga a mancare l'alimentazione durante queste due ultime operazioni.

RAID

File System

Il compito del FILE SYSTEM è quello di astrarre la complessità di utilizzo dei diversi media programando una interfaccia per i sistemi di memorizzazione.

Più semplicemente: indica informalmente un meccanismo con il quale i file sono posizionati e organizzati su dispositivi di archiviazione o dispositivi remoti tramite protocolli di rete.

→ organizzazione dal punto di vista utente:
- directory: Un insieme di informazioni per organizzare e fornire informazioni sui file che compongono un file system
- file: unità logica di memorizzazione

Attributi di un file

- nome
- tipo: necessario in alcuni sistemi per identificare il tipo di file
- Locazione e dimensione
- Data e ora: info. relative al tempo di creazione ed ultima modifica del file
- Info sulla proprietà: utenti, gruppi
- Attributi di protezione: chi è autorizzato ad eseguire operazioni sui file
- Altri Attributi:
  - flag (sistema, archivio, hidden, ecc.)
  - informazioni di locking
  - etc.

Tipi di file:

- A seconda della struttura interna:
  - senza formato (stringa di byte): file di testo
  - con formato: file di record, file di database, a.out, ...
- A seconda del contenuto:
  - ASCII/binario (visualizzabile o no, 7/8 bit)
  - sorgente, oggetto, ...
  - eseguibile (oggetto attivo)

Alcuni S.O. supportano e riconoscono diversi tipi di file, conoscendo il tipo del file il s.o. può evitare alcuni errori comuni, quali ad esempio eseguire un file eseguibile

Tipi di file: ulteriori distinzioni

- In un file system, si distingue fra:
  - *file regolari*
  - *directory*
    - file di sistema per mantenere la struttura del file system
  - *file speciali a blocchi*
    - utilizzati per modellare dispositivi di I/O come i dischi
  - *file speciali a caratteri*
    - utilizzati per modellare device di I/O seriali come terminali, stampanti e reti
  - altri file speciali
    - ad es., pipe

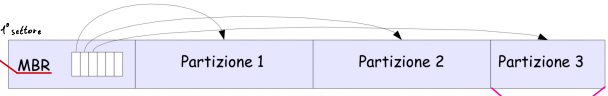
Semantica della coerenza

- In un sistema operativo multitasking, i processi accedono ai file indipendentemente
- Come vengono viste le modifiche ai file da parte dei vari processi?
- In UNIX
  - le modifiche al contenuto di un file aperto vengono rese visibili agli altri processi immediatamente.
  - esistono due tipi di condivisione del file:
    - condivisione del puntatore alla posizione corrente nel file (condivisione ottenibile con fork)
    - condivisione con distinti puntatori alla posizione corrente.

Organizzazione del disco

Il disco può essere diviso in una o più partizioni, partizioni indipendenti del disco che possono ospitare file system distinti.

master boot record (MBR)
- utilizzato per fare il boot di un sistema.
- Contiene la partition table (tabella delle partizioni)
- Contiene l'indicazione della partizione attiva
- Al boot, il MBR viene letto ed eseguito
- È il primo blocco



GUID Partition Table (GPT)
- Quando si partiziona, se si perde la tabella delle partizioni, c'è una replica della tabella stessa in fondo.
- Esistono delle Magic Number per riconoscere che tipo di partizioni sono state caricate.

- *superblock*
  - contiene informazioni sul tipo di file system e sui parametri fondamentali della sua organizzazione
- *tabelle per la gestione dello spazio libero*
  - struttura dati contenente informazioni sui blocchi liberi
- *tabelle per la gestione dello spazio occupato*
  - contiene informazioni sui file presenti nel sistema
  - non presente in tutti i file system
- *root dir*
  - directory radice (del file system)
- file e directory

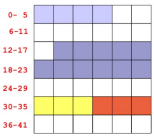
Struttura di una partizione
- ogni partizione inizia con un boot block
- il MBR carica il boot block della partizione attiva e lo esegue
- il boot block carica il sistema operativo e lo esegue
- l'organizzazione del resto della partizione dipende dal file system

Allocazione

Contigua

Pro: - comodo
- consente accesso diretto molto rapido

Contro: - scomodo per file system dinamico



directory

Name	Start	Size
a	0	4
b	13	11
c	30	3
d	33	3

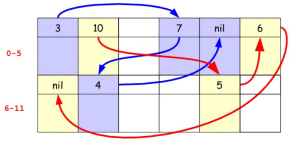
Concatenata

Il blocco punta al blocco successivo

Pro: - non c'è limite al file
- si risolve il problema della frammentazione esterna

Contro: - Scandire per la lettura finché non si arriva al punto desiderato.

Accesso diretto inefficiente.



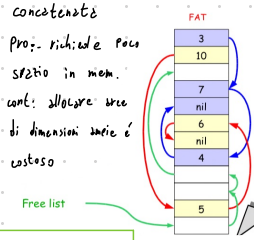
directory

Name	Start	End
a	0	4
b	1	6

Nella lista concatenata i blocchi liberi vengono mantenuti in una lista concatenata

Pro: - richiede poco spazio in mem.

cont: allocare uno di dimensioni varie è costoso



FAT (File Allocation Table)

Invece di usare parte del blocco dati per contenere il puntatore al blocco successivo si crea una tabella unica con un elemento per blocco (o per cluster)

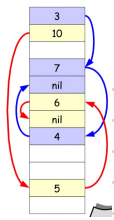
Pro: - i blocchi sono interamente dedicati di dati
- È possibile fare caching in memoria dei blocchi FAT

Cont.: la scansione richiede anche la lettura della FAT

Metodo usato da DOS

directory

Name	Start	End
a	1	6
b		



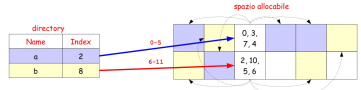
Indicizzata

L'elenco dei blocchi che compongono un file viene memorizzato in un blocco (o area) indice.

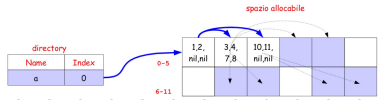
Per accedere ad un file, si carica in memoria la sua area indice e si utilizzano i puntatori contenuti.

Pro: - risolve il problema della framm. esterna.
- accesso diretto efficiente
- blocco indice caricato in memoria solo quando il file è aperto.

Cont.: - La dimensione del blocco indice determina l'ampiezza massima del file.
- Utilizzare blocchi indice troppo grandi comporta un notevole spreco di spazio



Indice multilivello



In UNIX ogni file è associato ad un I-NODE (Index Node)

Necessari in questa implementazione

Gestione spazio libero: un po' di conti

- **Premesse**
  - blocchi: 1K
  - dimensione partizione: 16GB
- **Mappa di bit**
  - 2^{24} bit = 2^{21} byte = 2048 blocchi = 2MB nella bitmap
- **Lista concatenata (FAT)**
  - 2^{24} puntatori = 2^{24}*3 byte = 48MB nella FAT
- **Lista concatenata (blocchi)**
  - spazio occupato nei blocchi liberi

Implementazione delle directory

- **Lista lineare**
  - semplice da implementare
  - inefficiente nel caso di directory di grandi dimensioni
- **Tabella hash**
  - occorre stabilire la dimensione della tabella di hash e il metodo di gestione delle collisioni
  - la gestione può essere inefficiente se ci sono numerose collisioni

Directory strutturata a grafo aciclico

- **Due implementazioni possibili**
  - link simbolici
  - hard link
- **Link simbolici**
  - viene creato un tipo speciale di directory entry, che contiene un riferimento (sotto forma di cammino assoluto) al file in questione
  - quando viene fatto un riferimento al file
    - si cerca nella directory
    - si scopre che si tratta di un link
    - viene *risolto* il link
      (ovvero, viene utilizzato il cammino assoluto registrato nel file

Quando il File system trova un I-node valido, ma non raggiungibile, lo posiziona nella cartella /lost+found, contrassegnandolo con il relativo valore dell'I-node

Sicurezza

Sicurezza: Concetto, obbiettivi teorici
Protezione: meccanismi
Trust: Percezione di sicurezza
} Sono indipendenti tra loro

Garanzie di Sicurezza	Possibili Errori	Problemi Pratici
1. Confidenzialità/riservatezza	→ Errore di disclosure (rivelate cose che non devono essere rivelate)	→ Sniffing traffico di rete che viaggia in chiaro
2. Integrità	→ DOS (Denial of Service)	→ man in the middle, Kernel Panic (DOS)
3. Continuità di servizio	→ Alterazione	→

CRITTOGRAFIA

É sicura la chiave non l'algoritmo, se perdiamo solo una chiave verrà decifrato solo ciò che viene criptato con quella chiave, se perdiamo la sicurezza dell'algoritmo abbiamo perso tutto il sistema. Per quello l'algoritmo é in chiaro per tutti ed é la chiave ad essere segreta.

Crittografia

chiave privata

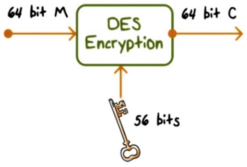
La chiave usata per criptare i messaggi é la stessa usata per decriptarli.
Pro: Algoritmi veloci.
Cons: La distribuzione delle chiavi private é un problema di sicurezza; può essere necessario uno scambio "fisico".

chiave pubblica

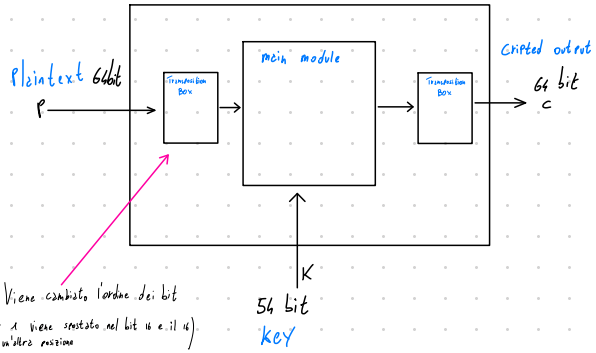
Esistono 2 chiavi distinte:
La chiave pubblica é utilizzata per criptare i mess. in chiaro
La chiave privata é utilizzata per decriptare i mess cifrati

DES (Data Encription Standard)

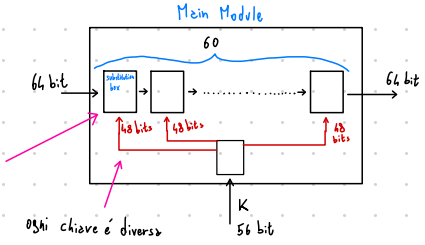
Data Encryption Standard



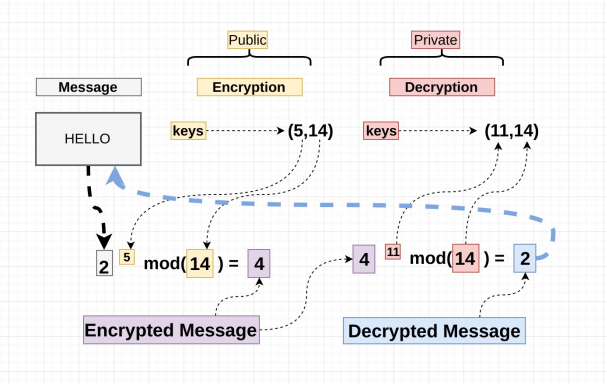
- Published in 1977, standardized in 1979
- Key: 64 bit quantity=8-bit parity+56-bit key
  - Every 8th bit is a parity bit
- 64 bit input, 64 bit output



alcuni bit sono sostituiti con altri: secondo una tabella, avviene uno split e i bit diventano 32 bit, dopo una serie di operazioni abbiamo 48 bits che vengono messi in XOR con la chiave a 48 bit in entrata.



RSA



Funzionamento di RSA (schema)

- vengono scelti due numeri primi molto grandi **p** e **q** (almeno 100 cifre)
- si chiami **n=pq**.
- si sceglie un valore **d** in modo tale che sia primo con **(p-1)(q-1)**
  - cioè: **MCD(d,(p-1)(q-1))=1.**
- sia **e** l'inverso moltiplicativo di **d mod (p-1)(q-1)**
  - cioé: **de mod (p-1)(q-1)=1**
- allora
  - **E(m) = C = m^e mod n**
  - **D(C) = m = C^d mod n**

Tipologie di attacco

Root kit: Programmi che servono per fare, o mantenere, Privilage escalation.
Per fare ciò sfrutta: codice con bug, amministratore poco accorto, cattive abitudini di uso.

Logics Bomb: porzione di codice inserito in un programma apparentemente innocuo e che resta latente fino al verificarsi di particolari condizioni che attivano "la bomba", ad esempio in un programma di gestione di un database una bomba logica può attivarsi al raggiungimento di un certo numero di record salvati oppure quando viene cancellato un preciso dato.

Backdoor: metodo, normalmente segreto, per passare oltre (aggirare, bypassare) la normale autenticazione. Hanno la funzione principale di superare le difese imposte da un sistema, al fine di averne accesso da remoto.

Worm: programma malevolo che riproduce sé stesso attraverso la rete

Password

Memorizzazione
Nei sistemi obsoleti venivano salvate in chiaro, Palese problema di sicurezza.

Codifica
Con una funzione one-way vengono codificate e salvate, quando viene eseguito il login la password viene codificata e confrontata con il risultato. Se si codifica anche il dizionario per l'attacco il problema permane.

Aggiunte di maiuscole, minuscole, cifre e caratteri speciali. Però ancora non basta.

Meccanismo del Sale
Prima di essere criptate e memorizzate le password vengono concatenate con un numero casuale (Salt). In questo modo per crackare le password con un dizionario bisogna tentare ogni parola con tutti i possibili numeri casuali e richiederebbe molto più tempo.

Non viene più utilizzato /etc/passwd, ma ora si usa /etc/shadow poiché quest'ultimo può essere letto solo da root

Controllo degli Accessi

ACL (Access Control List)

- Lista dei permessi dei processi dei relativi utenti

Revoca:

È sufficiente aggiornare in modo corrispondente le strutture dati dei diritti di accesso

Capability

Sono una sorta di "chiave" per l'accesso alla "serratura" che protegge l'oggetto.

Revoca:

L'informazione relativa ai permessi è memorizzata presso i processi, quindi?:

- Capability a validità temporale limitata.
- Doppia memorizzazione (ogni capability viene controllata prima di essere utilizzata, perdita di alcuni benefici delle capability).
- Capability indirette (vengono concessi diritti non agli oggetti, ma a elementi di una tabella globale che puntano agli oggetti; è possibile revocare diritti cancellando elementi della tabella intermedia)
- Cambiamento dell'identità dell'oggetto (contatore nel nome): (I processi devono chiedere nuovamente l'autorizzazione di accesso; può essere pesante se ci sono frequenti variazioni)

In sintesi si può revocare una capability:

1) cancellando l'utente
2) creando una nuova login per quell'utente e creando nel nuovo account solo gli oggetti a cui l'utente deve poter avere accesso
3) revocare l'accesso dell'utente ad un oggetto in particolare

Il problema delle capability è che non si può affermare di aver rimosso tutte le possibilità di accesso ad un utente se non recidendo l'oggetto stesso.

POSIX Working Group 1003.1e - 1003.2c

Controllo accessi

- **Il gruppo POSIX ha tentato di standardizzare numerose problematiche relative alla sicurezza:**
  - Access Control Lists (ACL) * (in Linux)
  - Capability * (in Linux)
  - Mandatory Access Control (MAC)
  - Information Labeling
  - Audit
- **Sfortunatamente**
  - Standardizzare aree così diverse e variegate era un progetto troppo ambizioso
  - Tuttavia, il lavoro svolto dal Working Group non è andato perso