KERNEL
Quattro categorie:
- Kernel monolitici
    Insieme completo e unico di procedure mutuamente correlate e coordinate (efficienza)
    Svantaggio -> aggiungere dispositivo significa aggiungere modulo (ricompilare)
- Microkernel
    Si porta fuori tutto ciò che può essere portato fuori dal kernel
    Solo due system call: send e receive -> comunicazione basata su message passing
    Semplici da realizzare; più espandibili (aggiungere un servizio significa aggiungere un processo a livello utente)
    e per modificare un servizio basta riscrivere solo il codice del servizio stesso.
    più stabili.
- Kernel ibridi
    Mantengono una parte di codice in kernel space per ragioni di efficienza
    Adottano il message passing tra i moduli in user space.
- Exokernel
    Mai usciti dal livello di ricerca
    Farebbe in modo che i processi abbiano la visione diretta dell'hardware.
    Quindi si occuperebbe di definire solo a quali settori di memoria può accedere un processo.

MACCHINE VIRTUALI
Sfruttano hypervisor per condividere e gestire l'hardware
Vantaggi:
    Consentono di avere sistemi operativi differenti che coesistono,
    Possono far funzionare sistemi monotask su sistemi multitask,
    Permettono di emulare architetture hardware diverse.
Svantaggi:
    Varie inefficienze, a cui si tenta di dare soluzioni tramite istruzioni hardware,
    Difficoltà a condividere le risorse, che richiederebbero una rete per comunicare

VM di processo -> supporta un solo processo e ha lo stesso lifespan del processo
    Fornisce un ambiente di programmazione indipendente dalla piattaforma

PROCESSI
Un processo è rappresentato da:
1. Il codice da eseguire (segmento codice).
2. dati su cui opera (segmento dati).
3. Lo stack di lavoro per la gestione di chiamate di funzione, passaggio di parametri e variabili locali.
4. Un insieme di attributi contenenti tutte le informazioni necessarie per la gestione del processo (PCB)

Nei PCB ci sono tre informazioni:
    identificazione di processo, informazioni di stato del processo, informazioni di controllo del processo.

    - Per identificare un processo si utilizza il PID, per evitare omonimie si usa un indice di reincarnazione
        Quando un processo termina il suo ID è riassegnato il contatore aumenta.

    - Lo stato del processo contenuto nel PCB è quello aggiornato all'ultima sospensione (es. se arriva interrupt)

    - Le informazioni di controllo sono varie:
        - Di scheduling (es. stato)

- Di gestione della memoria (es. per MMU)
- Di accounting (es. tempo di esecuzione)
- Relative alle risorse (es. file aperti)
- Per IPC (es. semafori)

MULTITHREADING
Un thread è l'unità base di utilizzazione della CPU.
In un processo multithreaded esistono molte "linee di controllo".
Esiste una tabella dei thread che tiene traccia di ogni thread a cui un processo "punta".
Quindi in un SO che supporta multithread a livello kernel, lo scheduler gestisce i singoli thread, non i processi
I thread condividono lo spazio di memoria e le risorse allocate degli altri thread dello stesso processo.

SCHEDULER
Schedule = sequenza temporale di assegnazioni delle risorse da gestire ai richiedenti.
Scheduling = l'azione di calcolare uno schedule.
Scheduler = componente software che calcola lo schedule.

Un processo potrebbe essere:
Running -> correntemente in esecuzione
Waiting -> ad esempio se avviene un interrupt
Ready -> nella coda di processi ready

Se accade che la coda dei processi ready è vuota, potrebbe essere perché tutti i processi sono in waiting.
Lo scheduler quindi potrebbe fare busy waiting (consuma energia) oppure il processore potrebbe fare nulla passando allo stato idle.

Tutte le volte che avviene un interrupt il processore è soggetto ad un mode switching (user / kernel).
Al termine della routine di gestione viene chiamato lo scheduler che potrebbe far partire un altro processo (context switch).

Gli eventi che possono causare context switching sono:
1. Quando un processo passa da uno stato running a uno stato waiting in seguito ad una system call bloccante.
Si sceglie per forza un altro processo.
2. Quando un processo un processo passa dallo stato running allo stato ready, ad esempio in seguito ad un interrupt
È possibile scegliere il processo corrente.
3. Quando un processo passa dallo stato waiting allo stato ready.
È possibile scegliere il processo corrente.
4. Quando un processo termina.
Lo scheduler dovrà per forza scegliere un altro processo.

Esistono due tipi di scheduler: preemptive e non-preemptive.
In uno scheduler non-preemptive il controllo della risorsa viene trasferito solo se l'assegnatario attuale lo cede volontariamente (casi 1 e 4)
Il principale vantaggio è l'assenza di un interval timer.
Negli scheduler preemptive invece gli switch possono avvenire in ogni condizione.
È quindi possibile utilizzare meglio le risorse.

Durante l'esecuzione di un processo si alternano attività svolte dalla CPU (CPU burst) a periodi di I/O (I/O burst).

Esistono processi che raramente fanno I/O, con CPU burst lunghi, detti CPU bound (es. database management system)

Invece processi caratterizzati da frequenti I/O burst si dicono I/O bound, ad esempio un programma di ray tracing.

ALGORITMI DI SCHEDULING

- FCFS First come, first served
  - Implementato con coda FIFO
  - Elevati tempi di attesa
  - Svantaggia processi I/O bound
  - Se un processo lento si mette davanti a processi più veloci fa bottleneck (convoy effect)

- SJF Shortest job first
  - Prima il processo con CPU burst più breve
  - Ottimale rispetto a tempo di attesa
  - Soggetto a starvation
  - NON IMPLEMENTABILE (teorico) -> non si sa cosa farà un processo in futuro
    - Si usano approssimazioni guardando comportamento passato
      - $T_{n+1} = at_n + (1-a)T_n$, ovvero la previsione è uguale al tempo * per a + (1-a) * la precedente previsione
        - Il parametro a è compreso tra 0 e 1, quindi più a è grande più si adatta ai cambiamenti, più a è basso meno cambiamenti avvengono.

In Shortest-Remaining-Time First un processo può essere messo in pausa se ne arriva un altro con CPU burst più breve

- ROUND ROBIN
  - Time slice (hardware dedicato per interval timer)

La durata del quanto di tempo è un parametro critico del sistema, quindi va tarato.
Se è breve, il sistema è meno efficiente perché deve cambiare il processo attivo più spesso (il context switch costa).
Se è lungo, in presenza di numerosi processi pronti ci sono lunghi periodi di inattività di ogni singolo processo.

Round robin è troppo democratico
- I processi non sono tutti uguali:
  - Ad esempio usando round-robin puro la visualizzazione di un video MPEG potrebbe essere ritardata
    - da un processo che sta smistando la posta. Ma la lettera può aspettare mentre il frame video no.

- SCHEDULING A PRIORITÀ
  - Ogni processo ha una sua priorità
    - Può essere statica (possibile starvation) o dinamica (aging)
      - Aging: la priorità aumenta in base al tempo di attesa per poi ripristinare la priorità base una volta eseguito.

- SCHEDULING A CLASSI DI PRIORITÀ
  - Processi simili sono divisi in classi di priorità.
  - La ready queue è divisa in tante sotto-code
  - Il processo da eseguire è scelto tra quelli ready con classe di priorità massima

- SCHEDULING MULTILIVELLO
    Le classi di priorità hanno a loro volta una propria politica di scheduling.

- SCHEDULING REAL-TIME
    Processi periodici (riattivati con cadenza regolare) e aperiodici
    Due stati: critico e non critico
    Programmi scritti con linguaggi ad hoc (es. no cicli infiniti)

    - SCHEDULER RATE MONOTONIC
        Ogni processo ha una priorità statica
        Scheduling in base a frequenza (periodo più corto)
    - SCHEDULER EARLIEST DEADLINE FIRST
        Priorità dinamica
        Viene scelto chi ha la deadline più prossima

RISORSE
- Tipi:
    Fungibili: qualunque istanza va bene
    Infungibili: serve una determinata risorsa

    Singole: una sola risorsa di classe definita
    Multiple: più risorse di più classi

    Bloccanti: richiedente si sospende finché non ottiene l'assegnazione

    Condivisibili: utilizzabili da più processi contemporaneamente

- Assegnazione:
    Statica: risorsa assegnata a processo per tutta la sua vita (es. descrittore processo)
    Dinamica: risorse allocate e deallocate durante l'esecuzione

    Una risorsa è Pre-rilasciabile se la funzione di gestione può sottrarla ad un processo prima che questo l'abbia effettivamente rilasciata.
        La risorsa verrà restituita in un secondo momento.
        Il suo stato non si modifica durante l'utilizzo o il suo stato può essere facilmente salvato e ripristinato (es. processore durante interrupt)

Diciamo che un processo è in stato di starvation quando esso è pronto per l'esecuzione
ma è impossibilitato perpetuamente ad accedere alla risorsa di cui ha bisogno per l'esecuzione.

Si dice che due o più processi sono in deadlock quando ognuno di essi attende che venga eseguita una certa azione da un altro processo.

DEADLOCK
Il deadlock è una condizione patologica del sistema che impedisce ai processi di terminare correttamente.
Le risorse bloccate in deadlock non possono essere utilizzate da altri processi.
Le condizioni per avere deadlock, dette di Coffman, sono:
    - Le risorse devono essere non condivisibili.
    - Assenza di prerilascio: le risorse coinvolte non possono essere prerilasciate volontariamente dai processi che le controllano.
    - Le richieste devono essere bloccanti (hold and wait) e un processo che ha già ottenuto

risorse può chiederne ancora.

- Attesa circolare: esiste una sequenza di processi tali per cui ognuno vuole una risorsa che possiede il successivo e l'ultimo vuole quella del primo.

Per rappresentare l'allocazione delle risorse si usano i grafi di Holt che tengono traccia di assegnamenti e richieste

DEADLOCK DETECTION

Teorema

SE ogni classe contiene una sola istanza di ogni risorsa,
Se le risorse sono ad accesso mutualmente esclusivo, non condivisibili e non prerilasciabili lo stato è di deadlock sse il Grafo di Holt contiene un ciclo.
Se il Grafo di Holt contiene un ciclo, questo alternerà nodi processo a nodi risorsa.
Quindi questo ciclo significa che un processo ha qualcosa che sta aspettando qualcun'altro e quest'ultimo ha qualcosa che sta aspettando il primo.

Teorema

Se le risorse sono ad accesso mutualmente esclusivo, non condivisibili e non prerilasciabili NON c'è di deadlock sse il grafo di Holt è completamente riducibile

Ovvero esiste una sequenza di passi di riduzione che elimina tutti gli archi del grafo.

KNOT

Dato un certo nodo n, possiamo costruire un insieme di raggiungibilità chiamato R(n) che rappresenta l'insieme di tutti i nodi raggiungibili da n.

Un knot del grafo G è il massimo sottoinsieme (non banale, non composto da un solo nodo) di nodi M tale che per ogni n in M, R(n)= M,

ovvero partendo da un qualunque nodo di M posso raggiungere tutti i nodi di M e nessun nodo all'infuori di esso.

Teorema

Dato un grafo di Holt con una sola richiesta sospesa per processo, se le risorse sono ad accesso mutualmente esclusivo, non condivisibili e non prerilasciabili,
allora il grafo rappresenta uno stato di deadlock se e solo se esiste un knot.

DEADLOCK RECOVERY

Una volta individuata la deadlock, la soluzione può essere:

Manuale: l'operatore viene informato ed eseguirà alcune azioni che permettano 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.

In entrambi i casi non se ne esce se non in maniera traumatica, perdendo esecuzioni e terminando processi.
I meccanismi di terminazione sono:

Terminazione totale: tutti i processi coinvolti vengono eliminati.

Terminazione parziale: viene eliminato un processo alla volta finché il deadlock non sparisce.

Un'altra soluzione è il checkpoint/rollback:

Lo stato dei processi viene periodicamente salvato su disco (checkpoint) e in caso di deadlock, si ripristinano (rollback) uno o più processi.

DEADLOCK PREVENTION

Si potrebbe pensare di:
Permettere la condivisione di risorse.
Problema: richieste bloccanti
Soluzione: allocazione totale:
È possibile richiedere che un processo richieda tutte le risorse all'inizio della computazione.
Facendo così il processo prima di partire sa già se potrà terminare.
Costo elevato -> un processo tiene la risorsa per tutto il tempo anche se gli servirà per poco.

Prevenire la condizione di pre-rilascio (non sempre possibile).

Risolvere il problema dell'attesa circolare:
Allocazione gerarchica:
Alle classi di risorse vengono associati valori di priorità e ogni processo può allocare solamente risorse di priorità superiore a quelle che già possiede.
Se un processo vuole allocare una risorsa a priorità inferiore, deve prima rilasciare tutte le risorse con priorità uguale o superiore a quella desiderata.

Sia allocazione gerarchica che allocazione totale sono altamente inefficienti, infatti:
- Nella prima l'indisponibilità di una risorsa ad alta priorità ritarda processi che già detengono risorse ad alta priorità
- Nella seconda anche se un processo ha necessità di risorse per poco tempo deve allocarle per tutta la propria esistenza.

DEADLOCK AVOIDANCE
ALGORITMO DEL BANCHIERE (vedi slides)

OSTRICH ALGORITHM
Algoritmo dello struzzo -> ignorare il problema del tutto lasciarlo risolvere all'amministratore di sistema quando se ne accorge.
Usato perché dal punto di vista ingegneristico, il costo di evitare i deadlock può essere troppo elevato.

GESTIONE DELLA MEMORIA
Effettuata da memory manager-> software che tiene traccia della memoria libera e occupata e alloca/dealloca la memoria per i processi.

Distinguiamo due spazi di indirizzamento: logici e fisici.
Ogni processo è associato ad uno spazio di indirizzamento logico e gli indirizzi usati in un processo sono indirizzi logici relativi a questo spazio.
La CPU genera quindi indirizzi logici.
In base a come è stata istruita la MMU, ad ogni indirizzo logico verrà fatto corrispondere un indirizzo fisico.
La funzione di trasformazione può essere parziale, in modo da non permettere di accedere a determinati indirizzi fisici garantendo un livello di protezione.

ALLOCAZIONE
Può essere contigua -> lo spazio fornito è costituito da celle di memoria consecutive o non contigua -> celle separate.
In quest'ultimo caso la MMU deve gestire un processo di traduzione più complesso.

Parliamo di allocazione statica quando viene assegnata un'area di memoria ad un processo e questa non cambia più
Parliamo di allocazione dinamica quando durante l'esecuzione un processo può essere spostato all'interno della memoria.

Nei sistemi embedded e a real-time viene usata una tecnica contigua e statica detta a partizioni fisse
(ogni processo viene caricato in una partizione sufficientemente grande)
Semplice ma limita il parallelismo ad un processo per partizione.

FRAMMENTAZIONE
La presenza di spazio inutilizzato all'interno di un'unità di allocazione è un fenomeno detto frammentazione interna.

La frammentazione interna dipende dall'uso di unità di allocazione di dimensione diversa da quella richiesta
La frammentazione esterna, invece, deriva dal susseguirsi di allocazioni e deallocazioni.
Per ovviare alla frammentazione esterna esiste la compattazione che fa uso del binding dinamico:
Si fermano i processi e si copiano le aree di memoria per non lasciare buchi.
Molto costosa e non utilizzabile in processi interattivi.

ALLOCAZIONE DINAMICA
Strutture dati per indicare zone libere e zone occupate:
- MAPPE DI BIT
Ad ogni unità di allocazione viene assegnato un bit (0 se libera o 1 se occupata)
I processori CISC hanno un'istruzione per trovare sequenze di bit uguali
La mappa deve avere una dimensione consona
Potrebbe essere calcolabile a priori
Scorrerla tutta costa O(m) dove m è il numero di unità di allocazione

- LISTE DI PUNTATORI
Si mantiene una lista dei blocchi allocati e liberi di memoria
Ogni elemento della lista specifica se si tratta di un processo (P) o di un blocco libero (hole, H) e specifica la dimensione (inizio/fine) del segmento.
Le operazioni avvengono in tempo costante

Quando il Memory Manager deve decidere quale area tra quelle libere utilizzare lo fa attraverso due tecniche:
First fit: primo blocco libero di ampiezza sufficiente
Next fit: come first fit ma riparte dall'ultimo punto in cui si era fermato

Esistono anche best fit (miglior spazio libero, tecnica molto lenta) e worst fit (blocco più grande tra quelli presenti).

PAGINAZIONE
Riduce frammentazione interna ed esterna, anche se non le elimina
Richiede hardware dedicato

Ogni processo ha uno spazio di indirizzamento logico, che viene suddiviso in un insieme di blocchi di dimensione fissa detti pagine di ampiezza una potenza di 2.
Dividiamo tutta la memoria fisica dei processi in frame della stessa dimensione delle pagine.
Grandezze standard delle pagine potrebbero essere 1, 2, 4 KB
Quando dobbiamo allocare un processo in memoria cerchiamo i frame disponibili e li assegniamo alle pagine.

Questi frame possono essere ovunque in memoria.

Ogni processo utilizza la tabella delle pagine per memorizzare la mappatura tra indirizzi virtuali e fisici.

TLB
Il TLB è una cache della tabella delle pagine, cioè solamente un sottoinsieme del suo contenuto viene memorizzato.

Ogni registro del TLB è diviso in chiave (indirizzo virtuale) e valore (indirizzo reale)
Se la chiave è presente (TLB hit) si ritorna il valore corrispondente, altrimenti (TLB miss) si usa la tabella in memoria
In questo caso si copia il valore nel TLB.

Grazie a questo sistema la frammentazione interna è al massimo di una pagina.

SEGMENTAZIONE
Serve per dividere la memoria di un processo in segmenti, ovvero aree distinte che vengono gestite diversamente.

In un sistema basato su segmentazione, un segmento è un'area di memoria (logicamente continua) contenente elementi tra loro affini.
Ogni segmento è caratterizzato da un nome (indice) e da una lunghezza.
La divisione in segmenti la fa in automatico il compilatore, o secondo le esigenze, la può fare il programmatore.

es:
Se abbiamo due processi che eseguono lo stesso codice (ad esempio un editor condiviso),
il segmento codice è comune e viene mappato in memoria allo stesso indirizzo, mentre i dati e lo stack sono separati.

La segmentazione NON risolve la frammentazione.

MEMORIA VIRTUALE
Si usa una parte della memoria secondaria, in modo che le pagine siano recuperabili quando servono.

Ogni processo ha accesso ad uno spazio di indirizzamento virtuale che può essere più grande di quello fisico.
Gli indirizzi virtuali (particolari indirizzi logici) possono essere mappati su indirizzi fisici della memoria primaria o della memoria secondaria.
Quando il programma tenta di accedere ad un indirizzo in memoria secondaria i dati associati vengono trasferiti in memoria principale.
Se la memoria è piena, si spostano in memoria secondaria i dati contenuti in memoria principale che sono considerati meno utili.

Demand paging -> tecnica di paginazione che ammette la presenza di pagine in memoria secondaria
Si utilizza un bit valid che se a 1 indica che la pagina è in memoria centrale
Se un processo tenta di accedere ad una pagina in memoria secondaria avviene un page fault e il pager carica la pagina mancante in RAM

ALGORITMI DI RIMPIAZZAMENTO
ANOMALIA DI BELADY
Ci si aspetta che con un algoritmo di rimpiazzamento il numero di page fault decresca in funzione del numero di frame, ma non è sempre così

- FIFO
    - Viene scelto il frame più vecchio in memoria
    - Non richiede hardware
    - Causa anomalia di Belady

- MIN
    - Viene scelta la pagina a cui si accederà più tardi nel futuro
    - È solo teorico -> non si possono conoscere i riferimenti futuri

- LFU (least frequently used)
    - Viene scelta la pagina meno acceduta
    - Teorico -> Mantiene un contatore di accessi e si basa sul rapporto accessi / permanenza in memoria
    - Poco affidabile, in quanto un processo potrebbe essere usato per tanto tempo frequentemente e poi non essere più utilizzato.

ALGORITMO A STACK

Un algoritmo si definisce a stack se per ogni stringa di riferimento e per ogni istante di tempo,
l'insieme delle pagine con memoria m sono un sottoinsieme dell'insieme delle pagine con m + 1 frame di memoria.

Gli algoritmi a stack non generano anomalia di Belady
Dimostrazione:
Poniamoci ad un certo tempo t:
Se p1 ... pn sono le pagine in memoria, aggiungendo un frame le pagine saranno p1 ... pn, p_n+1
Per avere l'anomalia di Belady dovrebbe avvenire un page fault nel caso della memoria con più frame.
(All'aumentare dei frame dovrebbero aumentare i page fault)
Ora ci sono tre casi:
- La pagina si trova in p1 ... pn -> non c'è page fault in nessun caso
- La pagina è in p_n+1 -> c'è page fault solo con memoria m
- Nessuno dei precedenti -> page fault in entrambi i casi

QED.

- LRU (least recently used)
    - Viene scelta la pagina utilizzata meno recentemente nel passato
    - Algoritmo a stack (non causa anomalia di Belady)
    - Potrebbe essere implementato con uno stack

Dimostrazione: LRU è a stack
Nello stato iniziale, quando in memoria ci sono un numero di pagine minore o uguale a m, allora gli stati della memoria con m ed m + 1 frame sono identici.
Quando si aggiunge il frame m + 1-esimo, nella memoria con più frame questo verrà normalmente aggiunto, in quella con m frame uno dovrà essere scelto come vittima.
Per ipotesi induttiva poniamo che LRU rispetti la definizione di algoritmo a stack.
Vogliamo dimostrare che resta a stack anche se confrontiamo i tempi t e t+1, m e m+1 frame.
Denotiamo con s[t+1] la pagina che entra al tempo t+1, abbiamo tre casi:
- s[t+1] appartiene agli m frame al tempo t -> no page fault
- s[t+1] appartiene agli m+1 frame al tempo t -> s[t+1] è l'elemento in più quindi un frame di m viene sostituito
- s[t+1] non appartiene agli m+1 frame al tempo t -> page fault, due casi:

- Sostituire l'elemento in più: in questo caso dagli m+1 frame viene eliminato l'elemento che non è in m frame
mentre dagli m frame viene eliminato un elemento scelto dall'algoritmo (resta sottoinsieme)
- Sostituire un elemento nell'intersezione -> dipende dall'algoritmo:
Dobbiamo mostrare che la scelta fra m pagine dipende solo dalla stringa di riferimenti, ovvero scelgano entrambi la stessa pagina.
LRU dato il tempo, sceglie deterministicamente la pagina usata meno di recente, mentre FIFO è influenzato anche dall'ampiezza della memoria.
QED.

LRU è molto costoso e supportato da poche MMU
Si utilizza un reference bit che viene posto a 1 quando una pagina è acceduta
Se si salvano periodicamente gli ultimi n reference bit si può avere una storia degli accessi di n bit
Si shiftano i bit di una posizione e si guarda il valore degli n bit di storia, la pagina vittima è quella con valore minore

Second-chance algorithm (algoritmo dell'orologio)
Passa in rassegna le pagine mettendo a 0 il reference bit se è a 1 o scegliendo come vittima la prima pagina con il bit a 0
In caso di pagine tutte a 0 degenera nell'algoritmo FIFO.
Facile da implementare, non richiede grandi sforzi da parte della MMU.

ALLOCAZIONE
Locale: ogni processo ha un insieme proprio di frame
Poco flessibile e statica (un processo potrebbe avere bisogno di più frame in un determinato momento e non in un altro)

Globale: tutti i processi possono allocare tutti i frame presenti nel sistema, quindi avviene competizione.
Possibile TRASHING
Quando un sistema spende più tempo per la paginazione che per l'esecuzione.
Avviene se si caricano troppi processi in memoria virtuale con pochi frame di memoria
Bisogna sospendere uno dei processi, in quanto è meglio sospendere un processo che mandare il sistema in trashing

WORKING SET
(indipendente dagli algoritmi di rimpiazzamento)
Modo per stimare, data una stringa di riferimento, di quante pagine avrà bisogno il processo.
Si definisce working set di finestra X l'insieme delle pagine accedute nei più recenti X riferimenti.
Se una pagina non compare in X riferimenti successivi in memoria, allora esce dal working set: non è più una pagina su cui si lavora attivamente
La somma dell'ampiezza di tutti i working set dei processi attivi deve essere sempre minore del numero di frame disponibili, altrimenti il sistema è in trashing.
X deve essere adeguato, altrimenti si potrebbero avere falsi positivi o negativi di trashing se, rispettivamente, troppo grande o troppo piccolo

GESTIONE MEMORIA SECONDARIA
I driver sono procedure software che permettono al SO di utilizzare l'hardware senza sapere come

esso funzioni, ma dialogandoci attraverso un'interfaccia standard.

I sistemi di I/O possono essere caratterizzati in diversi aspetti:
- Modalità di trasferimento
    A caratteri: quando l'unità di scambio è il singolo carattere come nel caso del terminale.
    A blocchi: quando si inviano i dati a blocchi di byte, come i dischi.
        Dati letti e scritti a blocchi di 512 - 1024 Bytes
- Modalità di accesso:
    Sequenziali: un insieme di elementi sono acceduti in una sequenza ordinata predeterminata (es. musicassetta).
    Random: può accedere ad un elemento arbitrario di una sequenza in tempo costante e indipendente dalla dimensione della sequenza stessa (es. CD)
- Trasferimento
    Sincrono: si invia un flusso continuo di dati al ricevitore utilizzando segnali di temporizzazione regolari che garantiscono
        che sia il trasmettitore che il ricevitore siano sincronizzati tra loro. Un esempio sono i nastri.
    Asincrono: invia i dati dal trasmettitore al ricevitore con bit di parità (bit di inizio e fine) in intervalli irregolari, come ad esempio il mouse.
- Condivisione
    Condivisibili
    Dedicati
- Velocità
    Pochi vs tanti Gb/s.
- Direzione di I/O
    Sola lettura.
    Sola scrittura.
    Entrambe.

Per la gestione dei SO esistono varie tecniche:
- Buffering: serve per disaccoppiare la produzione e il consumo dei dati.
    Se un dispositivo è più lento si usa un buffer intermedio.
- Caching: mantenere nella memoria più veloce il contenuto utile delle Memorie più lente.
- Spooling: serve per rendere condivisibili delle unità non condivisibili per loro natura.
    È un buffer che mantiene output per un dispositivo che non può accettare flussi dati distinti (es. stampante)

MEMORIA SECONDARIA
SSD
    Non hanno fragilità meccaniche.
    Consumano meno energia dei dischi rotazionali.
    Hanno un numero massimo di cicli di scrittura, questo implica scelte di file system più adatte di altre.
    Sono più veloci a leggere che a scrivere.
    Si legge a blocchi, si scrive a “banchi” (molti blocchi insieme)
    Accesso uniforme su tutto lo spazio di memoria (impiega sempre lo stesso tempo indipendentemente dalla posizione).

DISCHI ROTAZIONALI
    Un disco è composto da un insieme di piatti, suddivisi in tracce, le quali sono suddivise in settori.
    I dischi sono caratterizzati da tre parametri fondamentali:
        r: la velocità di rotazione, espressa in rpm (revolutions per minute).

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

Vr: la velocità di trasferimento, espressa in byte al secondo.

Il ritardo rotazionale è il tempo medio necessario affinché un settore arrivi sotto la testina ed è uguale alla metà del tempo che impiega a fare un giro

Il tempo di accesso è il tempo necessario per leggere un settore del disco, composto da tempo di seek, ritardo rotazionale e tempo di trasferimento.

DISK SCHEDULING

- FCFS First come, first served
  - FIFO
  - Non genera starvation

- SSTF Shortest seek time first
  - Seleziona la richiesta che richiede lo spostamento minimo della testina
    - In caso di parità sceglie a caso
  - È solo teorico perché può generare starvation

- LOOK (algoritmo dell'ascensore)
  - Ad ogni istante la testina è associata ad una direzione.
  - La testina si sposta di richiesta in richiesta, seguendo la direzione scelta.
  - Quando si raggiunge l'ultima richiesta nella direzione scelta, la direzione viene invertita e si eseguono le richieste nella direzione opposta.
  - Efficiente, no starvation

- C-LOOK
  - Come LOOK ma al termine delle richieste in una direzione si sposta alla richiesta più lontana nell'altra direzione

RAID

Serve per aumentare la velocità di accesso ai dischi grazie al parallelismo.
L'idea è quella di utilizzare un array di dischi indipendenti, che possano gestire più richieste di I/O in parallelo,
assicurandoci però di garantire che i dati letti in parallelo risiedano su dischi indipendenti.

Striping: il sistema RAID viene visto come un disco logico.
I dati nel disco logico vengono suddivisi in strip (settori, blocchi o altri multipli).
Strip consecutivi sono distribuiti su dischi diversi, aumentando le performance di lettura dei dati sequenziali.

- RAID 0
  - Non ha ridondanza
  - Velocità senza affidabilità
  - Dati distribuiti su più dischi
  - es: disco 1 -> Strip 0, 2, 4, 6; disco 2 -> Strip 1, 3, 5 , 7

- RAID 1
  - Usa mirroring
  - es: disco 1 -> Strip 0, 1, 2, 3; disco 2 -> Strip 0, 1, 2, 3

- RAID 5
  - I dischi memorizzano i dati come se avessero un disco in meno.

Per ogni “riga di strip” nei vari dischi viene memorizzato uno strip di parità che contiene l’or esclusivo di tutti gli strip corrispondenti.

es: con 5 dischi avremo gli strip da 0 a 3 nei primi 4 e lo strip di parità (0-3) nel quinto, ogni disco contiene uno strip di parità

Grazie alle propietà dello xor è facile aggiornare dati e strip

- RAID 4

Come RAID 5 ma gli strip di parità sono tutti in un solo disco

- RAID 6

Come RAID 5, ma si utilizzano due strip di parità, aumentando l'affidabilità (è necessario il guasto di tre dischi affinché i dati non siano utilizzabili).

FILE SYSTEM

Il compito del file system è quello di astrarre la complessità di utilizzo dei diversi media

Propone un'interfaccia comune, efficiente e conveniente da usare per i sistemi di memorizzazione.

Dal punto di vista dell'utente, il file system è composto da:

- File: unità logica di memorizzazione.
- Directory: servono per organizzare e fornire informazioni sui file che compongono un file system.

File

Il file è l’entità atomica del file system.

Gli attributi di un file sono: Nome Tipo, Locazione e dimensione, Data e ora, proprietà, Attributi di protezione, Flag, informazioni di locking, ecc.

I tipi di file possiamo catalogarli a seconda della struttura (formato) o del contenuto (ASCII, Binario, Sorgente, Oggetto, Eseguibile)

Per identificare i tipi dei file esistono tre tecniche:

- Meccanismi delle estensioni: sono delle convenzioni (.pdf, .png, .exe).
- Attributi nelle directory: quando le directory elencano i file, questi avranno delle informazioni di corredo, tra i quali il tipo.
- Magic number: in punti specifici dei file vengono inseriti dei numeri che identificano il tipo (es: #! per gli script nei sistemi Unix e Unix-like).

Struttura dei file

I file possono essere strutturati in molti modi:

- Sequenze di byte,
- Sequenze di record logici,
- File indicizzati (struttura ad albero).

Il metodo 1 è il più semplice.

I metodi 2 e 3 vengono usati per grossi archivi di formato standard o per database.

I sistemi operativi possono attuare diverse scelte nella gestione della struttura dei file:

- Scelta minimale: i file sono considerati semplici stringhe di byte, a parte i file eseguibili il cui formato è dettato dal SO.

Qualsiasi programma / applicativo deve contenere il proprio codice per interpretare in modo appropriato la struttura di un file.

- Parte strutturata / parte a scelta dell'utente: ad esempio i file Macintosh hanno due parti: resource fork e data fork.

La prima contiene le informazioni che interessano all'utente, mentre la seconda il codice e i dati del programma.

- Diversi tipi di file predefiniti.

Lo scopo è quello di rendere più leggero il kernel quindi la gestione viene sempre più delegata ai programmi applicativi.

- Più formati : implicano un codice di sistema molto più complesso e la necessità di dover accedere a file di formato diverso

Potrebbe causare incompatibilità tra i programmi, a fronte di una maggiore efficienza per la gestione dei formati speciali (Kernel based).

- Meno formati: implicano invece un codice più snello (Application based).

Metodi di accesso

I file memorizzano informazioni;

Al momento dell'uso è necessario accedere a queste informazioni e trasferirle in memoria.

Esistono molti metodi per accedere alle informazioni dei file:

- Sequenziale: le informazioni del file si elaborano ordinatamente, le operazioni di lettura e scrittura fanno avanzare il puntatore del file.

È il metodo più comune, usato da compilatori e editor.

- Accesso diretto: Il file si considera come una sequenza numerata di blocchi o record che si possono leggere o scrivere in modo arbitrario.

Le operazioni di lettura e scrittura diventano parametrizzate al blocco da leggere e scrivere

Molto utili quando è necessario accedere immediatamente a grandi quantità di informazioni.

- Accesso indicizzato: Read e write avvengono mediante una chiave che dà accesso all'elemento

La chiave diventa l'identificativo del record all'interno del file.

Esiste una tabella di corrispondenza chiave-posizione (memorizzabile in memoria o paginabile su disco)

Operazioni sui file

Le operazioni fondamentali sui file sono:

- Creazione;
- Apertura/chiusura;
- Lettura/scrittura;
- Posizionamento;
- Cancellazione;
- Troncamento;
- Lettura/scrittura attributi.

L'API (interfaccia per la programmazione) relativa alle operazioni su file è basata sulle operazioni open/close.

I file devono essere aperti prima di effettuare operazioni e chiusi al termine.

Questo perché la maggior parte delle operazioni sopra citate richiede una ricerca dell'elemento associato al file specificato nella directory.

Per evitare questa ricerca, quindi, molti sistemi richiedono l'impiego di una chiamata di sistema open() la prima volta che si adopera un file.

Il sistema operativo mantiene una piccola tabella contenente

informazioni riguardanti tutti i file aperti (tabella dei file aperti).

Quando si richiede un'operazione su un file, questo viene individuato tramite un indice in tale tabella, in questo modo si evita qualsiasi ricerca.

Quando il file non è più attivamente usato viene chiuso dal processo e il sistema operativo rimuove l'elemento a esso associato dalla tabella.

L'astrazione relativa all'apertura/chiusura dei file è utile per

- Mantenere le strutture dati di accesso al file,
- Controllare le modalità di accesso,
- Gestire gli accessi concorrenti,
- Definire un descrittore per le operazioni di accesso ai dati.

Directory

Il file system si basa sul concetto di directory, la quale fornisce un'astrazione per un insieme di file.

Le operazioni su directory sono:

- Creazione;
- Cancellazione;
- Apertura/chiusura di una directory;
- Lettura di una directory;
- Rinominazione;
- Link/unlink;

Le directory possono avere diverse strutture: livello singolo, due livelli, albero, grafo aciclico (DAG) e grafo.

Nella directory a livello singolo tutti i file sono contenuti nella stessa directory facilitando il supporto e la comprensione.

Tuttavia, una directory a livello singolo è limitata quando il numero di file aumenta o quando il sistema ha più di un utente

Ad esempio non possiamo avere più file con lo stesso nome

Nella struttura a due livelli, ogni utente ha la sua directory separata, in ciascuna sono elencati solo i file del proprietario.

Ogni volta che comincia una nuova elaborazione si fa una ricerca nella directory principale (master file directory, MFD).

Directory a grafo aciclico (DAG)

Nella struttura ad albero ogni file è contenuto in una directory univocamente.

Quindi in quest'ottica, i file possono essere contenuti in due o più directory, anche se il file è unico

Le modifiche sono visibili in tutte le directory.

Semantica della coerenza

Le scritture in un file aperto da parte di un utente sono immediatamente visibili ad altri utenti che hanno aperto contemporaneamente lo stesso file.

Esistono due tipi di condivisione del file in questo caso:

- Una con puntatore alla posizione corrente
- Una con condivisione di distinti puntatori alla posizione corrente

Usata in UNIX

Semantica delle sessioni

Fa in modo che le scritture su un file aperto non vengano viste

immediatamente da altri utenti.

Una volta chiuso il file le modifiche apportate sono visibili.

In questo modo un file può essere associato ad immagini diverse, così da rendere l'accesso ai file più veloce.

Usata in AFS.

IMPLEMENTAZIONE FILE SYSTEM

- Organizzazione di un disco

Un disco può essere diviso in una o più partizioni, Le partizioni sono ottenute prendendo i blocchi fisici e dandone una visione logica.

Le informazioni su come è strutturato il disco e sul partizionamento sono memorizzate nel master boot record (MBR).

Contiene la partition table e l'indicazione della partizione attiva. Al boot il MBR viene letto ed eseguito.

GPT è un partizionamento più moderno e versatile di MBR.

Il motivo della partizione è voler tenere file system o SO diversi in partizioni diverse

Oppure per non tenere il file system (aree di swap gestite dal pager).

Le partizioni hanno una struttura standard:

- Il boot block è l'inizio di ogni partizione e viene caricato dal MBR che lo attiva ed esegue permettendogli di caricare il SO e lo esegue.
- Il superblock contiene l'indice del file system, informazioni sul tipo file system e sui parametri fondamentali della sua organizzazione.
- Le tabelle per la gestione dello spazio libero contengono la struttura dati contenente le informazioni sui blocchi liberi.
- Le tabelle per la gestione dello spazio occupato contengono le informazioni sui file presenti nel sistema

  Non è presente in tutti i file system.
- La root dir è la radice del file system.
- File e directory grazie alla root dir.

- Allocazione dello spazio in blocchi

Una volta fornito dall'hardware e dal driver l'accesso al disco, bisogna occuparsi del posizionamento dei file nei blocchi attraverso l'allocazione.

Ci sono diversi tipi di allocazione:

- Allocazione contigua

  I file sono memorizzati in sequenze contigue di blocchi di dischi.

  Vantaggi: non si utilizzano strutture per collegare i blocchi,

  L'accesso sequenziale risulta efficiente in quanto blocchi contigui non necessitano di seek e l'accesso diretto è efficiente.

  Svantaggi: Se volessimo ampliare un file dovremmo riscriverlo da un'altra parte nel disco,

  L'allocazione contigua può portare alla frammentazione esterna.

  Soluzione: Per prevenire la frammentazione esterna si copia un intero file system su un altro disco o nastro.

  Svantaggio soluzione: Alcuni sistemi

richiedono l'esecuzione di questa funzionalità con il file system non montato.

Durante questo “periodo morto” (down time) il normale funzionamento del sistema non è possibile.

Utilizzo: Utile per file system di dispositivi come i CD ROM in cui si accede in sola lettura.

- Allocazione concatenata

Ogni file è costituito da una lista concatenata di blocchi:

ogni blocco contiene un puntatore al blocco successivo

il descrittore del file contiene i puntatori al primo e all'ultimo elemento della lista,

così da poter appendere velocemente i dati in fondo alla lista di blocchi senza scandire tutta la sequenza.

Vantaggi: In questo modo risolviamo frammentazione esterna e possiamo allargare i file,

L'accesso sequenziale in “append mode” è efficiente.

Svantaggi: L’accesso diretto è inefficiente (in quanto bisogna seguire l’intera lista di puntatori),

Progressivamente l’efficienza globale del file system degrada (aumenta il numero di seek),

Inoltre la dimensione utile di un blocco non è una potenza di due (overhead alto).

Soluzione: Per minimizzare l’overhead dei puntatori i blocchi vengono riuniti in cluster, e vengono allocati in modo indivisibile.

In questo modo la percentuale di spazio utilizzata dai puntatori diminuisce.

Svantaggio soluzione: Aumenta lo spazio sprecato per la frammentazione interna.

- Allocazione basata su FAT

Si crea una tabella unica con un elemento per blocco (o per cluster).

Vantaggi: In questo modo i blocchi dati sono interamente dedicati ai dati, risolvendo il problema di overhead,

Ottimizzazione del tempo d'accesso diretto

La testina del disco può trovare la locazione di ogni blocco leggendo le informazioni contenute nella FAT,

È possibile fare caching in memoria dei blocchi FAT così che l'accesso diretto diventa più efficiente

(in quanto la lista di puntatori può essere seguita in memoria).

Svantaggi: L’accesso sequenziale richiede comunque la lettura della FAT, quindi aumenta il numero di accessi al disco.

Utilizzo: Questo metodo è usato da DOS, macchine fotografiche e chiavette.

- Allocazione indicizzata

L'elenco dei blocchi che compongono un file viene

memorizzato in un blocco indice.

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

Vantaggi: Risolve il problema della frammentazione esterna.

Svantaggi: Come sempre, bisogna scegliere bene l'ampiezza dei blocchi per non sprecare memoria.

CONCATENAZIONE BLOCCHI INDICE
L'ultimo elemento del blocco indice non punta al blocco dati ma al blocco indice successivo.

Rimane il problema per l'accesso diretto

INDICE MULTILIVELLO
Si utilizza un blocco indice dei blocchi indice.
Degradano le prestazioni, in quanto richiede un maggior numero di accessi.

Se non bastano gli indici basta aggiungere un nuovo livello di indici.

In UNIX si utilizza l'allocazione indicizzata.
Ogni file ha associato un inode, ovvero un nodo indice.
Un inode è una struttura dati contenente gli attributi del file, e un indice di blocchi diretti e indiretti, secondo uno schema misto:

Puntatori diretti: vanno direttamente alle aree dati
Puntatori indiretti: vanno verso uno o più livelli di indici

es: Doppio indiretto -> Indice -> Indice -> Data

|--> Indice -> ...

Lo schema UNIX garantisce buone performance nel caso di accesso sequenziale

Si possono ottenere miglioramenti grazie al pre-caricamento delle pagine

- Gestione dello spazio libero

Si usa lista dello spazio libero

Creare un file = togliere spazio dalla lista, cancellarlo = aggiungere spazio alla lista

GRANDEZZA DEI BLOCCHI
Blocchi piccoli = ridotte prestazioni di seek
Blocchi grandi = spreco di memoria (frammentazione interna)

MAPPA DI BIT

Ad ogni blocco corrisponde un bit in una bitmap.

I blocchi liberi sono associati ad un bit di valore 0, i blocchi occupati sono associati ad un bit di valore 1.

Metodo semplice ed efficiente per trovare n blocchi liberi

Svantaggio: occupano molto spazio

LISTA CONCATENATA

Collegare gli spazi liberi, tenere un puntatore al primo di questi in una speciale locazione del disco e caricarlo in memoria.

Vantaggi: Richiede poco spazio in memoria centrale

Svantaggi: L'allocazione di un'area di ampie dimensioni è costosa

L'allocazione di aree libere contigue è molto difficoltosa.

LISTA CONCATENATA DI BLOCCHI

È costituita da una lista concatenata di blocchi contenenti puntatori a blocchi liberi

Vantaggi: Non è necessaria una struttura dati a parte, ma basta mantenere in memoria un blocco contenente elementi liberi.

Svantaggi: Ha gli stessi svantaggi della lista concatenata

- Implementazione delle directory

Una directory è un file speciale contenente informazioni sui file contenuti nella directory ed è suddivisa in un certo numero di directory entry.

Ogni entry deve permettere di accedere a tutte le informazioni necessarie per gestire il file (in UNIX le entry contengono indici Inode).

Gli attributi delle directory possono essere contenuti nelle directory entry (MS-DOS) o negli Inode (in UNIX tutte le info tranne il nome).

NOMI DELLE DIRECTORY

L'approccio più semplice è quello con nomi di lunghezza fissa, tuttavia la lunghezza massima deve essere ben calcolata:

Nomi corti -> problema di espressività

Nomi lunghi -> spreco di memoria

Per implementare i nomi a lunghezza variabile si potrebbe porre come lunghezza massima 255 caratteri, anche se spreca molto spazio

Un'altra idea sarebbe avere entry di lunghezza diversa, che oltre agli attributi contengano la propria lunghezza

I nomi dei file sono terminati dal carattere 0

Un quarto approccio è mantenere la lunghezza fissa delle entry e inserire i nomi di lunghezza variabili in un heap alla fine della directory

SCANSIONE DELLE DIRECTORY

La scansione lineare delle directory risulta inefficiente in presenza di cartelle molto grandi

Si può utilizzare una tabella hash, più efficiente ma con possibili collisioni

LINKING

Hard link: i blocchi dei file non vengono listati nelle directory, ma in una struttura dati associata

Quando si crea un hard link, viene incrementato il valore nell'Inode del file, in modo che questo sappia da quante directory entries è puntato.

Se viene rimosso un link, l'Inode rimane attivo finché il suo link counter non scende a 0.

Symbolic link: creare un nuovo file che contenga il path del file a cui vuole essere linkato.

Un link simbolico è un file speciale che contiene un riferimento sotto forma di cammino assoluto al file in questione

Quando si ricerca un file, lo si cerca nella directory e se si scopre che si tratta di un link questo viene risolto tramite il suo contenuto.

- Tecniche per ottimizzare le prestazioni e garantire la coerenza

Le cache permettono di accelerare l'accesso a dati in memoria secondarie.

Ma se le copie in memoria primaria vengono modificate, queste modifiche devono ripercuotersi anche sulle

copie in memoria secondaria, altrimenti si causa incoerenza nel file system.

FILE SYSTEM CHECKER

Confrontano i dati delle directory con quelli contenuti nei blocchi dei dischi, tentando di correggere ogni incoerenza.

Questi programmi non garantiscono di recuperare tutti i dati che dovevano essere scritti su disco.

Tutti i file system checker verificano ogni file system (partizione del disco) indipendentemente dagli altri.

FUNZIONAMENTO DI FSCK

1. Scandisce la tabella degli Inode, controllando le incoerenze.
2. Controlla le directory, che devono puntare ad Inode legali.
3. Scandisce l'albero da root per vedere se tutti i file sono raggiungibili.

Se trova Inode validi che non sono collegati li mette nella cartella Lost + Found.

4. Verifica il numero di riferimenti ad ogni file, controllando che sia corretto.

I blocchi dati devono essere usati da 0 (liberi) o 1 file.

L'Inode deve comparire in uno o più directory, tante quante sono nel suo counter.

5. Verifica Inode e blocchi liberi-occupati.
6. Aggiorna le tabelle per salvare i cambiamenti

SISTEMI BASATI SU JOURNALING

Tutte le modifiche si annotano in modo sequenziale in un file di registrazione, detto log.

Ogni insieme di operazioni che esegue uno specifico compito si chiama transazione.

Una transazione è un'operazione che viene eseguita in modo atomico: o tutto, o niente.

Una transazione si considera completata (committed) quando è stata memorizzata nel log

Periodicamente, le transazioni nel log vengono effettuate nel file system.

Quando un'intera transazione è stata completata, se ne rimuovono le

annotazioni dal log.

In caso di crollo del sistema nel log ci potranno essere zero o più transazioni.

Le transazioni presenti non sono mai state ultimate nel file system e devono essere ripetute.

SICUREZZA

- Nozioni base

Non esistono sistemi sicuri al 100%.
Il livello massimo di sicurezza è uguale al suo anello più debole.
Un sistema sicuro e composto da: hardware, software e HUMANWARE.
Un sistema "sicuro" oggi non è detto sia sicuro un domani.
Il livello di sicurezza di un sistema si misura in "trust".

- proprietà garanti di un sistema sicuro

Data confidentiality: Se viene meno abbiamo disclosure.
Data integrity: Se viene meno abbiamo alteration.
System availability: Se viene meno abbiamo denial of service.

- Policy

Le componenti che descrivono la policy sono completamente estranee ai meccanismi di implementazione delle stesse rendendo i meccanismi
che ne garantiscono l'integrità capaci di cambiare senza impattare le policy lo stesso ragionamento vale al contrario.

- Crittografia

Processo mediante il quale si rende un messaggio plaintext -> ciphertext in modo da garantirne la segretezza per tutti tranne il destinatario del messaggio.

Implementazione

Le implementazioni avvengono tramite one-way function "OWF"

Dato x, f(x) è relativamente semplice da calcolare, ma dato f(x), calcolare x è “computazionalmente difficile" o “impossibile”.

Solitamente "mescolano" i bit in modo complesso, spesso tramite diverse iterazioni sullo stesso insieme di bit

Operazioni di bit swapping, inversioni etc.

Un’alternativa è la fattorizzazione di numeri primi molto grandi.

Due tipi di implementazione:

Crittografia a chiave segreta

Algoritmo DES

Crittografia a chiave pubblica

Algoritmo RSA

- Tipologie di attacco

Attivi

Facile da individuare (dos)

Passivi

Difficili da scoprire (Spyware)

Esterni

Sistema solitamente compromesso attraverso la rete (dos)

Interni

Sistema compromesso dall'interno (Malware)

Elementi sensibili: Hardware, Sistema operativo, Librerie e tool di sistema, Applicazioni,

Utenti (Punto più critico).

Esempio di attacco

Step 1: Individuare il sistema da attaccare.
Step 2: Ottenere informazioni sul sistema.
Step 3: Ottenere un nome utente
Step 4: Ricercare la password di quell'utente (Brute Force o attacco dizionario se la password è banale)
Step 5: Cercare di acquisire privilegi da superuser.
Step 6: Si installa un trojan horse.

Buffer Overflow

Si verifica una vulnerabilità di overflow del buffer quando si forniscono a un programma troppi dati.

I dati in eccesso danneggiano lo spazio vicino permettendo all'attaccante di fare una injection di codice.

In particolare, il buffer viene riempito con codice malevolo e si fa in modo che il codice in eccesso imposti come indirizzo di ritorno l'inizio del buffer.

I linguaggi più suscettibili sono C e C++ data la loro natura a basso livello

Fornisce totale libertà allo sviluppatore riguardo l'allocazione di memoria.

Contromisure:

Per difendersi dal buffer overflow bisogna:

Scrivere codice sicuro, evitando lo “sloppy programming” (codice poco versatile)

Non usare funzioni come gets(), strcpy() e sprintf().

Soluzione 1: Aggiungere controlli a tempo di compilazione, che permettono di verificare i limiti del buffer.

Svantaggio: In questo modo diminuiscono le prestazioni.

Soluzione 2: Aggiungere patch per compilatori che controllano i valori di ritorno, copiandoli in posizione sicura.

Svantaggio: Richiede ri-compilazione delle applicazioni.

Soluzione 3: È possibile effettuare controlli a run-time che ad esempio verificano le dimensioni degli stack frame

Se questi presentano qualche errore terminano il programma.

Time-of-check to Time-of-use

TOCTTOU è causato da modifiche in un sistema tra il controllo di una condizione (come verifica delle credenziali) e l'uso dei risultati di tale controllo.

Trojan horse

I Trojan horse sono programmi che replicano le funzionalità di programmi di uso comune o programmi dall’apparenza innocua ma che contengono codice malevolo.

Soluzione: Comportarsi da utente consapevole quindi non lanciando eseguibili non trusted.

Bombe Logiche

Una bomba logica è un pezzo di codice inserito in un sistema software che attiva una funzione dannosa quando vengono soddisfatte le condizioni specificate.

Ad esempio, se un dipendente viene licenziato e il suo nome sparisce dall'elenco dei salari

Soluzione: controllare il codice sorgente.

Backdoor / Trapdoor

Backdoor / Trapdoor sono fatte per creare accessi secondari nel sistema, che aggirano i sistemi di protezione ordinari.

Soluzione: controllare il codice sorgente.

Virus/Worm

Un virus è un frammento di programma che viene inserito in programmi esistenti modificandoli

Viene creata quindi la possibilità di trasmettersi come frammento di programma in altri programmi (riproduzione) per poi attivarsi una volta diffuso.

Un worm invece è esso stesso un programma che diffonde copie di sé stesso in rete.

Oggi la distinzione tra virus e worm non è molto chiara.

Worm di Morris

Il Worm di Morris fu il primo worm ad attirare l'attenzione mediatica.

Il worm sfruttava diverse vulnerabilità per ottenere l'accesso ai sistemi:

- Un buco nella debug mode del programma sendmail di UNIX

Gli utenti potevano mettere all'interno di un file l'indicazione per decidere a chi inoltrare la posta.

In questo file si poteva inserire una pipe per fare elaborare la posta da un processo.

- Un buffer overflow presente nel servizio finger.
- La possibilità di accedere alla rete tramite esecuzione remota con rsh senza password.

Questo worm non faceva nulla di male, era solo programmato per autoreplicarsi, ma in breve tempo saturò la rete.

- Autenticazione

L'autenticazione consiste nel verificare l'identità di un utente in modo da fornirgli i permessi a lui concessi.

I meccanismi di autenticazione sono basati su:

- Qualcosa che l'utente "conosce".
- Qualcosa che l'utente "ha".
- qualcosa che l'utente "è".

L'autenticazione basata su password è il tipo di autenticazione più utilizzato.

Tipi di attacco

Meccanismo di salt - attacco database

Per velocizzare l'operazione di cracking si può applicare la funzione hash al vocabolario, e memorizzare i valori criptati in un database.

In questo modo, con il tempo di ricerca in un database si confrontano gli hash della vittima con quelli già memorizzati nel database.

Soluzione: Non si cripta solo la password ma vi si aggiungono dei byte casuali (sale), il sale viene memorizzato in chiaro nel file delle password.

Inoltre, è meglio che l'elenco degli utenti non sia pubblico in modo da complicare i metodi di attacco.

Login spoofing

Viene creata dall’attaccante una finta schermata di login. Si attende che la vittima inserisca i dati di autenticazione.

Quando l'utente immette i dati mostra un messaggio di login fallito, a questo punto fa partire il vero programma di login

Nel mentre ha rubato i dati immessi (che erano giusti).

La vittima crede di aver digitato male la password, la reimmette ma questa volta entrando senza problemi nel sistema.

Soluzione 1: Essere molto accorti come utenti.

Soluzione 2: Autenticazione tramite oggetti fisici.

Packet sniffing

Un packet sniffer è un software che analizza il traffico di rete.

Cerca di individuare pacchetti contenenti coppie login/password spediti "in chiaro" da meccanismi di comunicazione come telnet e ftp (oggi deprecato).

Memorizza le coppie login/password per uso futuro.

Soluzione 1: Questo attacco è aggirabile utilizzando protocolli di criptazione dei dati in rete come SSL.

Soluzione 2: Password challange e response.

Soluzione 3: Password one-shot.

PAM

Pluggable Authentication Module (PAM) è un servizio generale di autenticazione basato su file di configurazione.

Generalmente, ogni servizio crea un contatto chiaro con PAM e si aspetta una risposta di tipo "sì" o "no".

Autorizzazione

L'autorizzazione rappresenta l'insieme di meccanismi e politiche con cui il SO decide se un soggetto (processi) ha il permesso di

eseguire una determinata azione su un oggetto (risorse).

È realizzata tramite meccanismi software:

- Trusted Computing Base, Reference Monitor.
- Matrice di Accesso (ACL, Capability).

Principi fondamentali

- Principio di Accesso mediato: tutti gli accessi ad un oggetto devono essere controllati.

  In genere si controllano i diritti solo al momento dell’apertura.
- Principio di Separazione dei privilegi: un sistema non dovrebbe concedere permessi in base ad una singola condizione.
- Principio di Failsafe default: nessun soggetto ha diritti per default.
- Principio di Privilegio minimo: ogni soggetto ha, in ogni istante, i soli diritti necessari per quella fase dell’elaborazione.

  Le azioni in più creano un problema di sicurezza.

Definizioni

Come detto, abbiamo un insieme di soggetti S che sono le entità attive (processi, thread).

Un insieme di oggetti O che sono le entità su cui si vogliono eseguire le azioni (S è un sottoinsieme di O).

Infine, abbiamo un insieme di diritti di accesso R che è l'insieme delle azioni che si possono eseguire (read, write ecc).

Per semplicità conviene uniformare i processi a livello di proprietario.

Un Dominio di protezione è un insieme di coppie <o, rs> dove o è un oggetto ed rs è un sottoinsieme di R.

Informalmente, ogni coppia specifica un oggetto e un insieme di azioni che possono essere eseguite su tale oggetto.

Normalmente ogni utente (identificato da UID e GID) opera all'interno di un dominio di protezione che determina cosa può fare e cosa no.

Matrice di accesso

Una matrice di accesso è una matrice domini/oggetti.

L'elemento A(i, j) contiene i diritti che il dominio Dj prevede per l'oggetto Oj.

Quando si crea un nuovo oggetto viene aggiunta una colonna e il contenuto di tale colonna è deciso al momento della creazione dell'oggetto.

L'associazione di un soggetto ad un dominio può essere statica o dinamica. Esempi di associazione dinamica sono SETUID di UNIX e run as di Windows.

Implementazione

- Access control list

La ACL associa ad ogni oggetto una lista di elementi <dominio, diritti di accesso>.

Ad esempio user::rw, group::r. L'ampiezza della lista può essere ridotta associando i diritti a insiemi di domini o usando diritti standard.

Ad esempio UNIX usa tre liste: owning user, owning group, others.

- Capability

Ad ogni dominio viene associata una lista di capability, ovvero coppie <oggetti, diritti di accesso>.

Sono una sorta di chiave che apre la serratura che protegge l'oggetto.

I Processi mantengono le capability e le presentano quali "credenziali" per accedere all'oggetto.

Per funzionare le capability non devono essere coniate.

Esistono quindi due possibili approcci:

- Capability mantenute nello spazio kernel associato al processo, protette dai meccanismi di protezione del kernel.
- Capability mantenute nello spazio utente: protette da meccanismi crittografici, memorizzate dai processi ma non modificabili (approccio usato nei sistemi distribuiti).

- Revoca

La revoca dei diritti di accesso può essere:

- Immediata o ritardata (subito o si può attendere).
- Selettiva o generale (per alcuni i domini o per tutti).
- Parziale o totale (tutti i diritti o solo alcuni).
- Temporanea o permanente.

Nei sistemi ACL è sufficiente aggiornare in modo corrispondente le strutture dati dei diritti di accesso (si rimuove un dominio dall'oggetto).

Tuttavia, nei sistemi basati su capability l'informazione è memorizzata nei processi. In questo caso abbiamo più opzioni da cui scegliere:

- Capability a validità temporale limitata: Una capability "scade" dopo un prefissato periodo di tempo.

- Doppia memorizzazione: ogni capability viene controllata prima di essere utilizzata (si perdono alcuni benefici delle capability).
- Capability indirette: si assegnano i diritti non agli oggetti ma ad elementi di una tabella globale che puntano agli oggetti.
È possibile revocare diritti cancellando elementi dalla tabella intermedia.
- Cambio identità dell'oggetto: I processi devono chiedere nuovamente l'autorizzazione di accesso.