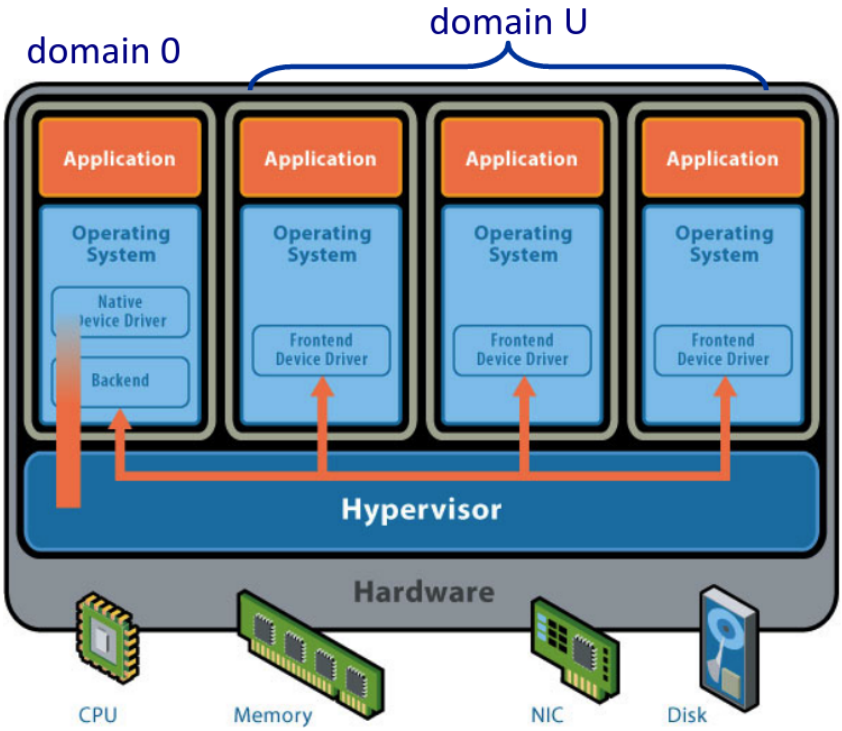
”…”

1 — xen

VMM nato come opensoruce paravirtualizzato.

Figura 1.1: Architettura xen



Xen è un VMM di sistema, ovvero si appoggia direttamente sull'hardware e quindi può eseguire direttamente chiamate system call che necessita del rin 0.

Il vmm si occupa della virtualizzazione della CPU, della memoria e dei dispositivi per ogni macchina virtuale, qui definiti domain.

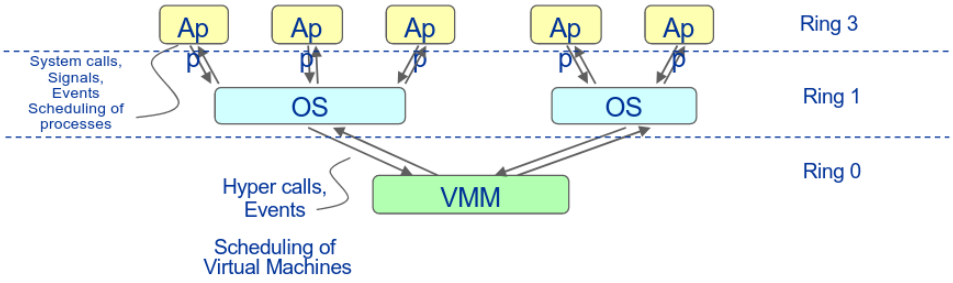
Xen offre una interfaccia di controllo in grado di gestire la divisione delle risorse tra i vari domini. L'accesso a questa interfaccia è consentito soltanto da una speciale VM, la domain 0.

1.1 Caratteristiche

Data la natura **paravirtualizzata** delle VM gestite da xen, le VM possono eseguire direttamente system calls che vengono delegate al VMM tramite hypercalls.

Per quanto riguarda la **protezione**, i guest OS sono collocati nel ring 1.

Figura 1.2: Protezione e hypercalls guest OS



1.2 Gestione memoria e paginazione

Gestione della memoria

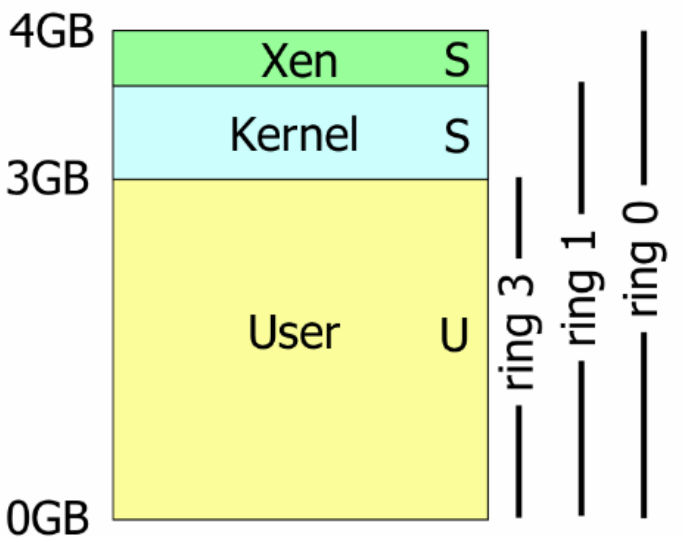
Gli OS guest gestioscono la memoria virtuale mediante i meccanismi e le politiche tradizionali.

La soluzione adottata si basa sulle tabelle delle pagine delle VM:
Vengono mappate nella memoria fisica dal VMM (**shadow page tables**, possono essere accedute in scrittura soltanto dal VMM stesso ma sono disponibili in modalità read-only ai guest.

In caso di necessità di update, il VMM valuta la richiesta e la esegue.

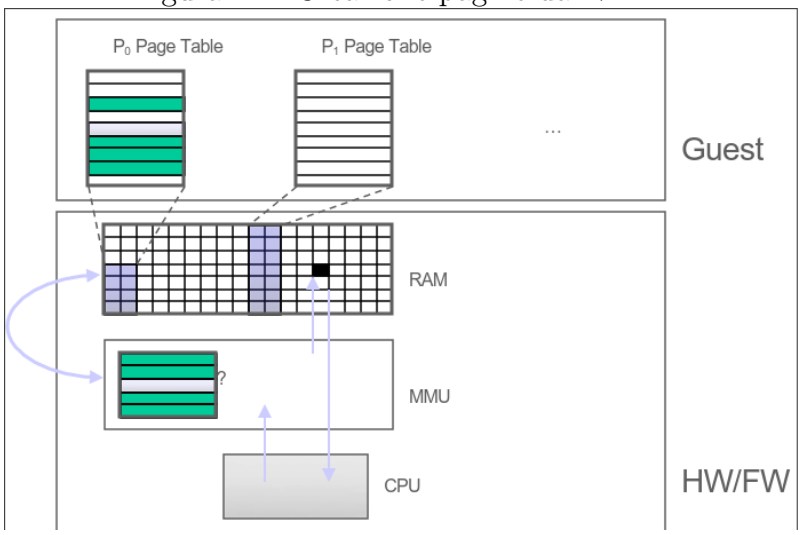
Per alleggerire l'onere del procedimento di update, viene implementato il **memory split**, per permettere una maggiore efficienza delle hypercalls: xen risiede nei primi 64MB del virtual address space.

Figura 1.3: Struttura virtual address space- memory split



I guest OS si occupano della paginazione, delegando al VMM la scrittura delle page table entries, una volta create sono disponibili in read-only per il guest che le ha richieste.

Figura 1.4: Creazione pagine dal VMM

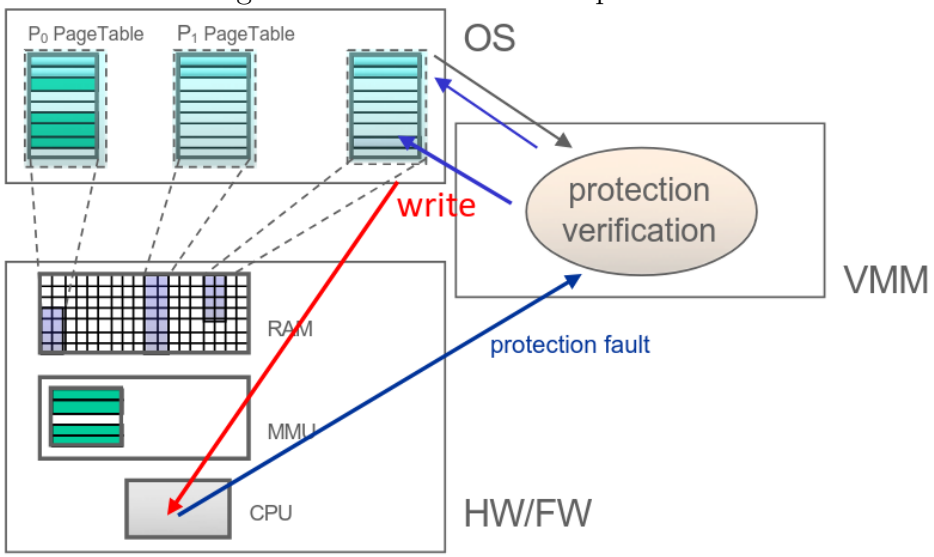


Creazione di un processo

Il SO richiede una nuova tabella delle pagine al VMM:

- aggiunte alla tabella le pagine appartenenti al segmento di xen
- xen registra la nuova tabella e acquisisce il diritto di scrittura esclusiva
- ogni successiva update da parte del guest provoca un protection-fault, comporta la verifica e l'aggiornamento della PT

Figura 1.5: Creazione di un processo



Balloon process

Dato che la paginazione è a carico dei guest, il VMM necessita di avere un meccanismo per reclamare da altre macchine virtuali pagine di memoria meno utilizzate. Su

ogni macchina virtuale è in esecuzione un **ballon process** che, in caso di necessità, si "gonfia" per ottenere altre pagine che poi cede al VMM.

1.3 Virtualizzazione della CPU

Il VMM definisce una archiettura simile a quella del processore, con istruzioni privilegiate sostituite da opportune hypercalls.

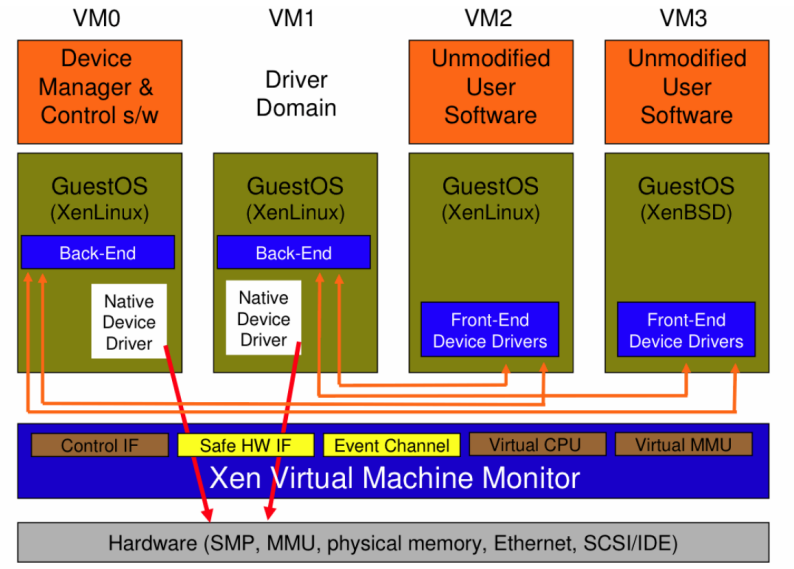
Il VMM si occupa dello scheduling delle macchine virtuali: **Borrowed Virtual Time** scheduling algorithm:

- si basa sulla nozione di virtual-time
- algoritmo general-purpose, consente di ottenre schedulazioni efficienti in caso di vincoli stringenti

Esistono due clock:

- real-time, inizia al boot
- virtual-time, associato a VM, avanza solo quando la VM esegue

Figura 1.6: Virtualizzazione dell'I/O

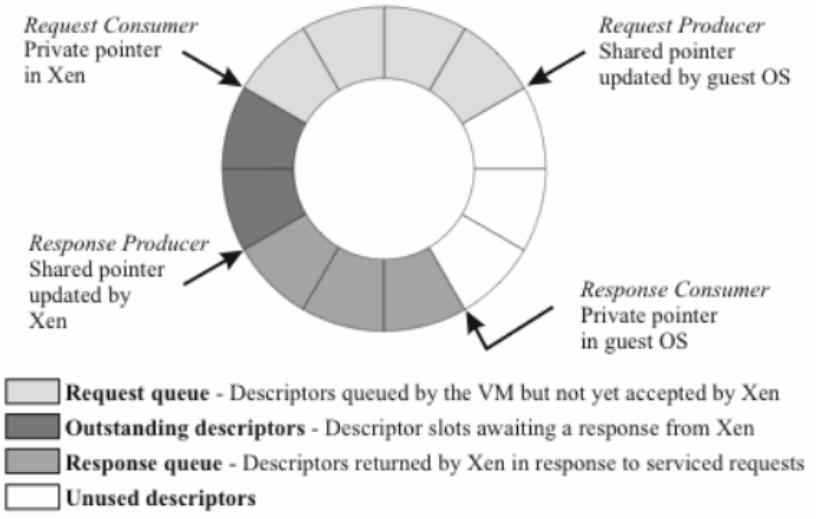


Esiste un **back-end driver** per ogni dispositivo, il suo driver è isolato all'interno di una particolare macchine virtuale (tipicamente dom0), ha accesso diretto all'hardware.

Ogni guest prevede un **front-end driver** virtuale semplificato che consente l'accesso al device tramite il back-end.

Questa tecnica comporta una semplificazione della portabilità a scapito della necessità di comunicaazione don il back-end attraverso degli asynchronous I/O rings.

Figura 1.7: I/O rings



1.4 Gestione interruzioni e eccezioni

La gestione delle interruzione viene virtualizzata in modo da lasciare a ogni guest la gestione delle interruzioni, il vettore punta direttamente alle routine del kernel guest.

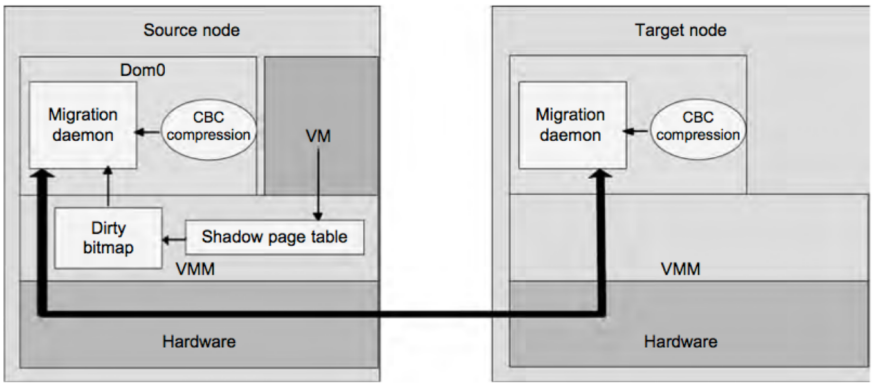
Il page-fault è un caso particolare, nel quale è necessario l'intervento del VMM, in quanto l'indirizzo che ha provocato il page-fault è contenuto nel registro CR2, inaccessibile al guest: il guest punta a codice xen che esegue una copia del CR2 all'interno di una variabile nello spazio guest.

1.5 Live migration in xen

La migrazione è **guest-based**: il comando di migrazione viene eseguite da un demone di migrazione nel domain0 del server di origine della macchine da migrare.

La realizzazione si basa sulla modalità pre-copy, le pagine da migrare vengono compresse per ridurre l'occupazione di banda.

Figura 1.8: Migrazione live



2 — Protezione

Sicurezza

Insieme delle tecniche per regolamentare l'accesso degli utenti al sistema di elaborazione. La sicurezza impedisce accessi non autorizzati al sistema e i conseguenti tentativi dolosi di alterazione e distruzione dei dati.

Meccanismi per **identificazione**, **autenticazione** e **autorizzazione** di utenti "fidati".

Protezione

Insieme di attività volte a garantire il controllo dell'acecsso alle risorse logiche e fisiche da parte degli utenti autorizzati all'uso di un sistema di calcolo.

Definizione, per ogni utente autorizzato, di:

- quali **risorse** sono accessibili
- quali **operazioni** può effettuare

Sono stabilite tramite tecniche di controllo degli accessi.

2.1 Protezione

In un sistema, il controllo degli accessi si esprime tramite la definizione di tre livelli concettuali:

- modelli
- politiche
- meccanismi

2.1.1 Modelli

Un modello di protezione definisce **soggetti**, **oggetti** ai quali i soggetti possono accedere e i **diritti** di accesso:

- oggetti, risorse fisiche e logiche alle quali applicare limitazione di accesso;
- soggetti, entità che possono richiedere l'accesso agli oggetti, utenti e processi;
- diritti di accesso, operazioni con le quali è possibile operare sugli oggetti;

Dominio di protezione

Ad ogni soggetto è associato un **dominio** che rappresenta l'ambiente di protezione nel quale il soggetto esegue, il dominio specifica i diritti di accesso posseduti dal soggetto nei confronti di ogni risorsa.

Un dominio di protezione è **unico per ogni soggetto**, mentre un processo può eventualmente cambiare dominio durante la sua esecuzione.

2.1.2 Politiche

Le **politiche di protezione** definiscono le regole con le quali i soggetti possono accedere agli oggetti.

Classificazione delle politiche:

- **Discetional Access Control** (DAC): il creatore di un oggetto controlla i diritti di accesso per quell'oggett (UNIX), definzione delle politiche decentralizzata.
- **Mandatory Access Control** (MAC): i diritti di accesso vengono definiti in modo centralizzato. Installazione ad alta sicurezza (es. enti gonvernativi)
- **Role Based Access Control** (RABC): un ruolo ha specifici diritti di accesso alle risorse, gli utenti possono appartenere a diversi ruoli, i diritti sono assegnati in modo centralizzato.

Principio del privilegio minimo

Ad ogni soggetto sono garantiti i diritti di accesso solo agli oggetti strettamente necessari per la sua esecuzione, è una caratteristica desiderabile in tutte le politiche di protezione.

2.1.3 Meccanismi

I meccanismi di protezione sono gli strumenti messi a disposizione dal sistema di protezione per imporre una determinata politica.

Principi di realizzazione

- **Flessibilità** del sistema di protezione, i meccanismi di protezione devono essere sufficientemente generali per consentire l'applicazione di diverse politiche;
- **Separazione** tra meccanismi e politiche, la politica definisce il *cosa* va fatto e il meccanismo il *come* va fatto.

In UNIX si utilizza la politica DAC e il SO offre un meccanismo per definire e interpretare i tre bit dei permessi.

2.1.4 Dominio di protezione

Un dominio definisce una serie di coppie che associano un oggetto all'insieme delle operazioni che il soggetto associato al dominio può eseguire.

Ogni dominio è associato univocamente a un soggetto.

Domini disgiunti o con diritti di accesso comune

Esiste la possibilità per due o più soggetti di effettuare alcune opezione comuni su un oggetto condiviso, le operazioni vengono svolte da processi che operano per conto di soggetti, tuttavia un processo appartiene a un solo dominio in ogni istante.

Associazione tra processo e dominio

Modalità statica L'insieme di risore disponibili a un processo rimane **statico** durante tutto il suo tempo di vita.

L'associazione statica non è adatta nel caso si voglia limitare per un processo l'uso delle risorse a quello strettamento necessario.

Modalità dinamica L'associazione tra processo e dominio varia durante l'esecuzione del processo.

Matrice degli accessi

Un sistema di protezione può essere rappresentato a livello astratto utilizzando una **matrice degli accessi**.

Figura 2.1: Matrice degli accessi

	O1	O2	O3
S1	read,write	execute	write
S2		execute	read,write,

- Ogni riga è associata a un oggetto
- Ogni colonna è associata a un oggetto

La matrice consente di rappresentare il modello e le politiche valide nel sistema considerato, specificando

- **soggetti**
- **oggetti**
- **diritti** accordati ai soggeti sugli oggetti

Le informazioni contenute nella matrice possono variare nel tempo, per effetto di operazioni che ne consentono la modifica → le informazioni contenute nella matrice all'istante t rappresenta lo **stato di protezione** del sistema in t.

La matrice degli accessi offre ai **meccanismi** di protezione le informazioni che consentono di verificare il rispetto dei vincoli di accesso.

Il meccanismo di protezione:

- verifica se ogni richiesta di accesso che proviene da un processo che opera in un determinato dominio è consentita oppure no
- autorizza l'esecuzione delle richieste se permesse
- esegue la modifica dello stato di protezione in seguite ad ogni richiesta autorizzata da pate di un processo

Quando un operazione M deve essere eseguita nel dominio D_i sull'oggetto O_j, il meccanismo consente di controllare che M sia contenuta nella casella `access(i,j)`.

2.1.5 Modello di Graham-Denning

La modifica controllata dello stato di protezione può essere ottenuta tramite un opportuno insieme di comandi (Graham e Denning, 1972):

- create object
- delete object
- create subject
- delete subject
- read access right
- grant access right
- delete access right
- transfer access right

Propagazione dei diritti di accesso

La possibilità di copiare un diritto di accesso per un oggetto da un dominio ad un altro della matrice di accesso è indicato con il copy flag `*`.

Un soggetto S_i può trasferire un diritto di accesso α per un ogget X ad un altro soggetto S_j ad un altro soggetto S_j solo se S_i ha accesso a X con il diritto α, e α ha il copy flag.

L'operazione di propagazione può essere realizzata in due modi:

- trasferimento del diritto, viene perso dal soggetto originale
- copia del diritto: viene mantenuto dal soggetto originale

Diritto owner

Se un soggetto S_i ha il diritto **owner** su un oggetto X, può assegnare/revocare un qualunque diritto di accesso a un soggetto S_j.

Diritto control

Se un soggetto S_i ha il diritto **control** su un soggetto S_j, può assegnare/revocare un qualunque diritto di accesso a un soggetto S_j per un qualsiasi oggett X.

Figura 2.2: Matrice degli accessi

	O1	O2	O3	S1	S2
S1	read*, write	execute write	write owner		control
S2		owner	read, ~~write~~		

Switch

Un processo che esegue nel dominio del soggetto può commutare al dominio di un altro soggetto S_j. L'operazione è consentita solo se il diritto **switch** appartiene a `access(`S_i`,`S_j`)`.

2.1.6 Realizzazione della matrice degli accessi

La matrice degli accessi è una notazione astratta, realizzare in memoria una struttura dati matriciale $N_s \times N_o$ non sarebbe ottimale, considerando il fatto che è una matrice sparsa.

Esistono due approcci possibili:

- **Access Control List** (ACL): rappresentazione per colonne, ogni oggetto possiede una lista che contiene tutti i soggetti che possono accedervi, con relativi diritti di accesso.
- **Capability List**: Rappresentazione per righe, ogni soggetto possiede una lista che contiene gli oggetti accessibili, con relativi diritti di accesso.

Access control list

La lista degli accessi, per ogni oggetto, è rappresentata da un insieme di coppie:

`<soggetto, insieme dei diritti>`

limitatamente ai soggetti con un insieme non vuoto di diritti per l'oggetto.

Quando si deve eseguire l'operazione M su un oggetto O_j da parte di S_i, si cerca nella lista degli accessi

<S_i, R_k>, con M appartente a R_k

La ricerca può essere fatta in precedenza su una lista di default che contiene i diritti di accesso applicabili a tutti gli oggetti.

Utenti e gruppi Solitamente ogni soggetto rappresenta un singolo utente, molti sistemi tuttavia hanno il concetto di **gruppo di utenti**, liste di utenti identificate da un nome ch e possono essere inclusi nell'ACL.

Se i gruppi sono presenti, l'ACL ha la seguente forma:

UID_1 GID_1: `<insieme di diritti>`

con UID user identifier e GID group identifier.

Capability list

La lista delle capability, per ogni soggetto, è la lista di elementi ognuno dei quali:

- è associato a un oggetto a cui il soggetto può accedere
- contiene i diritti di accessi consentiti su tale oggetto

ogni elemento della lista prende il nome di **capability** .

La capability si compone di un identificatore (indirizzo) che identifica l'oggetto e la rappresentazione dei vari diritti concessi.

Quando S intende eseguire una operazione M su O_j, il meccanismo di protezione controlla se tra le capability di S ne esista una relativa ad O_j che contiene M

Le liste di capability devono essere protette da manomissioni, proprietà ottenibile spostando l'effettiva lista nello spazio del kernerl e esponendo soltato un riferimento a tale lista.

L'utilizzo di una sola delle due soluzioni è inefficiente, in ACL tutti i diritti di un soggetto sono sparsi nelle varie ACL degli oggetti, e nelle CL tutti i diritti di accesso applicabili a un oggeto sono sparsi nelle varie CL dei soggetti.

Revoca dai diritti di accesso

In un sistema di protezione dinamica può essere necessario revocare i diritti di accesso per un oggetto, la revoca può essere:

- **generale** o **selettiva**: valere per tutti gli utenti o solo per un sottoinsieme
- **parziale** o **totale**: tutti i diritti o un sottoinsieme
- **temporanea** o **permanente**: il diritto di accesso non sarà più disponibile, oppure potrà essere successivmente riottenuto

In ACL revocare diritti su un oggetto risulta semplice, in quanto sono raccolte in un unica entri della lista, al contrario di CL nel quale è necessario agire su ogni entry che riguarda anche l'oggetto in esame.

Cancellazione/aggiunta di un utente

In un sistema di multi-user è possibile modificare l'insieme degli utenti autorizzati:

- **cancellazione** utente esistente → eliminazione di ogni traccia dell'utente dal sistema di protezione
- **aggiunta** nuovo utente → inizializzare il sistema di protezione per l'utente e il suo accesso alle risorse

In ACL eliminare un utente è complesso, in quanto è necessario agire su ogni entry che riguarda anche l'utente in esame, al contrario di CL nel quale basta eliminare l'entry associata all'utente.

Si implementa spesso una soluzione mista, con ACL (unita a una cache in RAM per accessi frequenti), tutte le operazioni successiva su una risorsa vengono effettuate tramite il fd e le capability.

2.2 Sicurezza

La sicurezza riguarda il controllo degli accessi al sistema, la protezione di un sistema può essere inefficace se un utente non fidato riesce a fare eseguire programmi che agiscono sulle risorse del sistema.

2.2.1 Sicurezza multilivello

La maggior parte dei sistemi operativi permette ai singoli utenti di gestire accesso ai loro file e oggetti, tuttavia in alcuni ambiti è richiesto un più stretto controllo sulle regole di accesso alle risorse, ottenibile stabilendo regole più generali (MAC).

L'organizzazione che gestisce il sistema definisce le politiche MAC che stabiliscono regole generali su chi può accedere e a che cosa tramite l'adozione di un **modello di sicurezza**.

I modelli di sicurezza più usati sono:

- modello **Bell-La Padula**
- modello **Biba**

Entrambi sono modelli multilivello.

2.2.2 Modelli di sicurezza multilivello

I **soggetti** (utenti) e gli **oggetti** (risorse) sono classificati in **livelli** di accesso:

- clearance levels
- sensititivity levels

Il modello fissa inoltre le **regole di sicurezza** che controllano il flusso delle informazioni tra i livelli.

2.2.3 Modello Bell-La Padula

Progettato principalmente per organizzazioni militari che necessitano di **confidenzialità** delle informazioni. Associa a un sistema di protezione due regole di sicurezza MAC che stabiliscono il flusso di propagazione delle informazioni nel sistema.

Livelli di sensibilità degli oggetti:

- non classificato
- confidenziale
- segreto
- top secret

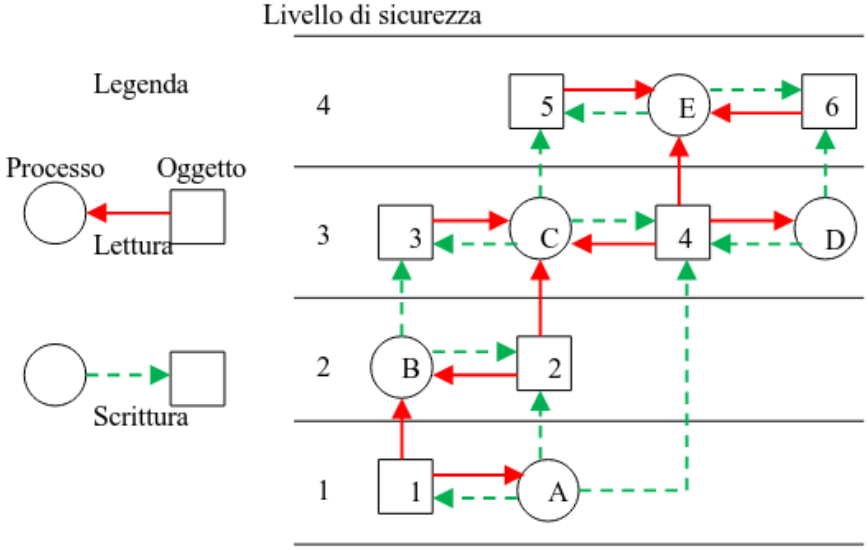
Livelli di autorizzazione (clearance) per i soggetti, assegnati a seconda del ruolo dell'utente nell'organizzazione ovvero dei documenti a quali è consentito accedere.

Regole di sicurezza

- proprietà di semplice sicurezza: un processo in esecuzione al livello di sicurezza k può leggere solo oggetti al suo livello o a livelli inferiori
- proprietà `*` : un processo in esecuzione allivello di sicurezza k può scrivere solamente oggetti al suol livello o a quelli superiori

Questo significa che i processi possono leggere verso il basso e scrivere verso l'alto, ma non il contrario, quindi il flusso delle informazioni è dal basso verso l'alto.

Figura 2.3: Diagramma sicurezza Bell-La Padula



Questo modello non è concepito per mantenere l'integrità dei dati ma per conservare segreti, è infatti ammesso sovrascrivere informazioni appartenenti a livelli superiori.

Esempio - difesa cavalli di troia

Si immagini un utente, Paolo, creatore del file `Fp` contenente una stringa riservata con permessi `r/w` solo per processi che appartengono a lui. Un utente ostile, Marco, ottenuto accesso al sistema, installa il file eseguibile `CT` e copia nel file system un file privato `Fm` che verrà utilizzato come "tasca posteriore".

Marco induce Paolo a eseguire il processo `CT`, che copia il contenuto di `Fp` in `Fm`, senza violare classiche regole di protezione (es. ACL).

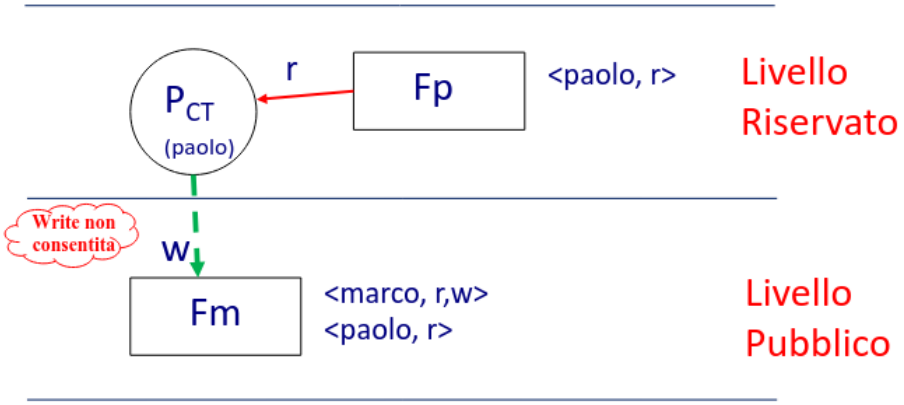
Il modello, per prevenire attacchi di questo tipo, implementa 2 livelli di sicurezza, **privato** e **pubblico**:

- ai processi e file di Paolo viene assegnato il livello *riservato*
- ai processi e file di Marco viene assegnato il livello *pubblico*

Quando il processo, avviato da Paolo con livello riservato, tenta di scrivere su `Fm` (pubblico) la proprietà `*` è violata e il tentativo è negato, nonostante ACL lo consenta.

La politica di sicurezza ha la precedenza sulle regole ACL.

Figura 2.4: Diagramma blocco cavallo di troia



2.2.4 Modello Biba

L'obiettivo di questo modello è l'integrità dei dati, diversamente dal Bell-La Padula.

- proprietà di semplice sicurezza: un processo in esecuzione al livello di sicurezza k può scrivere solo oggetti al suo livello o a livelli inferiori
- proprietà * : un processo in esecuzione al livello di sicurezza k può leggere solamente oggetti al suol livello o a quelli superiori

Questo modello è il duale matematico del Bell-La Padula, quindi non sono utilizzabili contemporaneamente.

2.2.5 Architettura dei sistemi ad elevata sicurezza

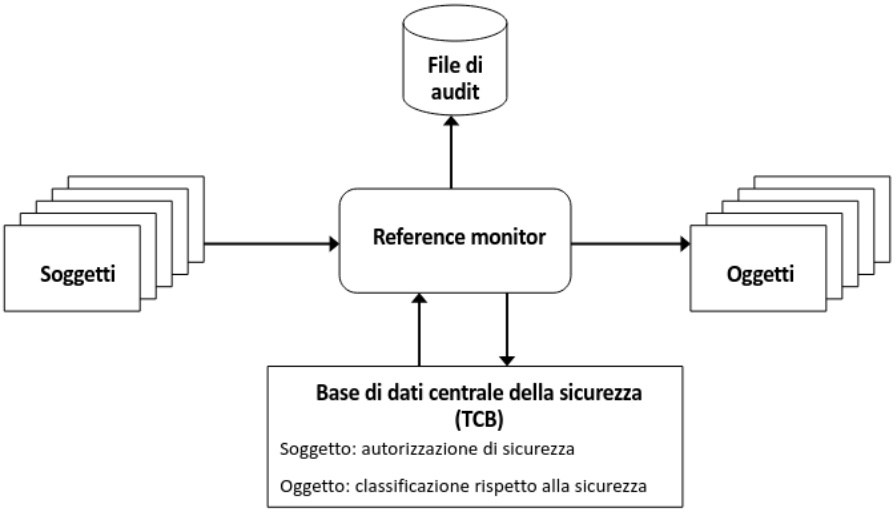
Sistemi operativi sicuri o fidati Sistemi per i quali è possibile definire formalmente dei requisiti di sicurezza.

Reference monitor É un elemento di controllo realizzato a livello hardware che regola l'accesso dei soggetti agli oggetti sulla base di parametri di sicurezza.

Trusted computing base Il RM ha accesso a una base di calcolo fidata (TCB) che contiene:

- privilegi di sicurezza per ogni soggetto
- attributi (classificazione di sicurezza) di ciascun oggetto

Figura 2.5: Architettura sistemi ad elevata sicurezza



2.2.6 Sistemi fidati

Il reference monitor impone le regole di sicurezza (Bell-La Padula) e ha le seguenti proprietà:

Mediazione completa Le regole di sicurezza vengono applicate ad ogni accesso. Per motivi di efficienza, le soluzioni devono essere almeno parzialmente hardware.

Isolamento il reference monitor e la base di dati sono protetti da modifiche non autorizzate.

Verificabilità La correttezza del reference monitor deve essere provata, deve essere possibile dimostrare formalmente che il monitor impone le corrette regole di sicurezza e fornisce mediazione completa e isolamente.

Viene inoltre compilato un **audit file** dove vengono mantenuti gli eventi importanti per la sicurezza, come i tentativi di violazione alla sicurezza e le modifiche non autorizzate al nucleo di sicurezza.

3 — Programmazione concorrente

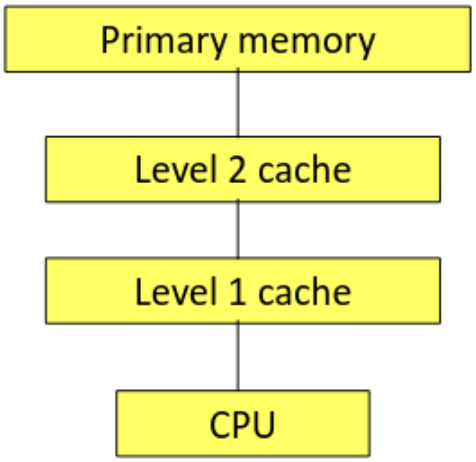
La programmazione concorrente è l'insieme delle tecniche, metodologie e strumenti per il supporto all'esecuzione di sistemi software da insieme di attività svolte simultaneamente.

Inizialmente implementata attraverso interruzioni (problemi con variabili comuni), oggi la programmazione concorrente è resa più facile dal reale parallelismo reso possibile da sistemi multiprocessore sempre più diffusi.

Le decisioni prese in merito di metodi di suddivisione dei processi e corretta sincronizzazione dipendono da tipo di applicazione e di architettura disponibile.

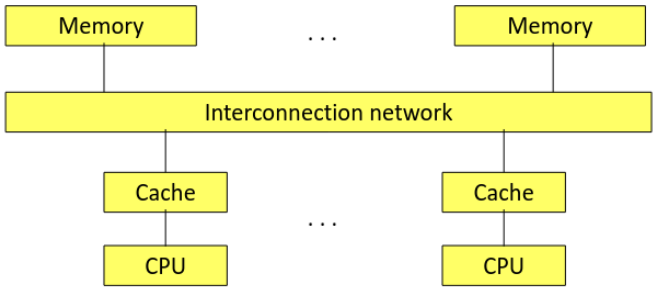
3.1 Tipi di architettura

Figura 3.1: Architettura single processor



3.1.1 Sistemi multiprocessore

Figura 3.2: Architettura memory multiprocessor

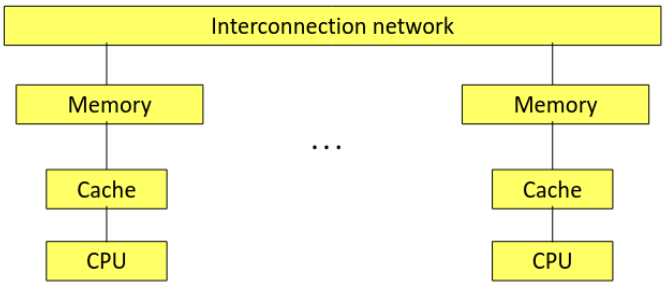


Due modelli:

- **UMA**: sistemi a multiprocessore con un numero ridotto di processori (da 2 a 30 circa)
  - la rete di interconnessione realizzata tramite *memory bus* o *crossbar switch*
  - Uniform Memory Access: tempo di accesso uniforme a da ogni processore a ogni locazione di memoria, chiamati anche **SMP** (symmetric multiprocessors).
- **NUMA**: sistemi con un numero elevato di processori (decine o centinaia)
  - memoria organizzata gerarchicamente per evitare congestioni sui bus
  - rete di interconnessione composta da insieme di *switches* e *memorie* strutturato ad albero, distanze variabili dai processori
  - Non Uniform Memory Access: tempo di accesso non uniforme

3.1.2 Distributed memory

Figura 3.3: Architettura Multicomputers e Network systems



Due modelli:

- **Multcomputer**: processori e rete fisicamente vicini → *tightly coupled machine*, la rete di interconnessione offre un cammine di comunicazione tra i processori ad alta velocità

- **Network systems**: nodi collegati da una rete locale o geografica → *loosely coupled systems*.

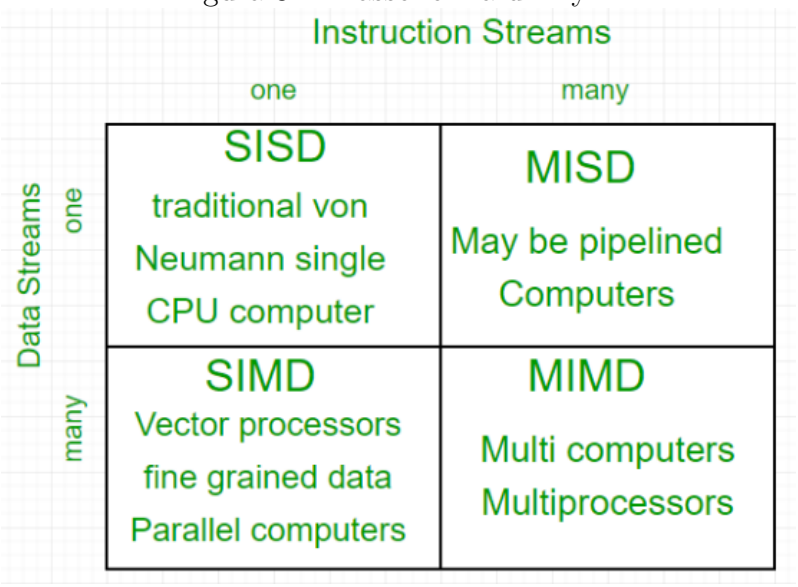
I nodi di un distributed memory system possono essere o singoli processori o shared memory multiprocessor.

3.1.3 Classificazione di Flynn

La tassonomia di Flynn è basata su due concetti:

- parallelismo a livello di istruzioni
  - **single instruction stream**: esecuzione di un singolo flusso di istruzioni
  - **multiple instruction stream**: esecuzione di più flussi in parallelo
- parallelismo a livello di dati:
  - **single data stream**: elaborazione di un singolo flusso sequenziale di dati
  - **multiple data stream**: elaborazione di multipli flussi di dati paralleli

Figura 3.4: Tassonomia di Flynn



3.2 Applicazioni

- multithreaded
  - strutturate come un insieme di processi per far fronte alla **complessità**, aumentare l'**efficienza** e per semplificare la programmazione.
  - i processi possono condividere variabili
  - esistono più processi che processori (generalmente)
  - processi schedulati ed eseguiti indipendentemente

- sistemi multitasking/distribuiti
  - le coponenti dell'applicazione vengono eseguite su nodi collegati tramite opportuni mezzi di interconnessione
  - comunicazione tramite scambio di messaggi
  - tipicamente client server
- applicazioni parallele
  - risolvere un dato problema più velocemente sfruttando il parallelismo disponibile a livello HW
  - a seconda del modello, istruzioni/thread/processi paralleli interagenti tra di loro

3.3 Processi non sequenziali e tipi di interazione

Algoritmo Procedimenti logico che deve essere eseguite per risolvere un determinato problema.

Programma Descrizione di un algoritmo mediante un linguaggio, che rende possibile l'esecuzione da parte di un elaboratore.

Processo Insieme ordinato degli eventi cui da luogo un operatore sotto il controllo di un programma.

Elaboratore Entità astratta realizzata in hardware e parzialmente in software, in grado di eseguire programmi.

Evento Esecuzione di una operazione, ogni evento determina una transizione di stato dell'elaboratore.

3.3.1 Processo sequenziale

Sequenza di stato attraverso i quali passa l'elaboratore durante l'esecuzione di un programma.

Un programma può evere più processi associati ad esso, ognuno di questi rappresenta l'esecuzione dello stesso codice con dati di ingresso (possibilmente) diversi.

Un processo può essere rappresentato tramite un grado orientato, detto grafo di precedenza del processo, composto da nodi e archi orientati. Ogni nodo rappresenta un evento corrispondente all'esecuzione di una operazione.

Il grafo di precedenza è a **ordinamento totale**, ovvero ogni nodo ha un predecessore e un successore.

3.3.2 Processo non sequenziale

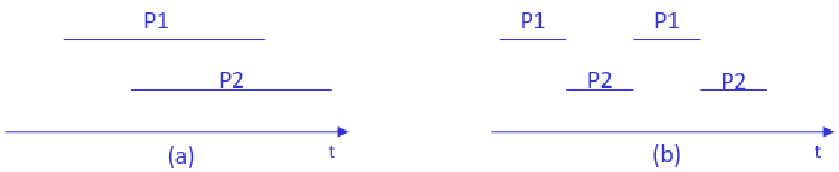
Non sempre un processo possiede la proprietà dell'ordinamento totale, molti problemi possono essere risolti più naturalmente tramite processi non sequenziali.

Un processo non sequenziale è una successione di eventi seconda una relazione d'ordine parziale.

L'esecuzione di un tale processo richiede un elaboratore in grado di supportare questo tipo di esecuzioni e un linguaggio di programmazione adatto, non sequenziale.

Elaboratore non sequenziale

- sistemi multielaboratori (a)
- sistemi monoelaboratori (b)



Linguaggio non sequenziale consente la descrizione di un insieme di attività concorrenti, sfruttando moduli che possono essere eseguiti in parallelo.

Slide 32-63!!!

3.4 Proprietà dei programmi

Traccia dell'esecuzione Sequenza degli stati attraversati dal sistema di elaborazione drante l'esecuzione del programma.

Stato Insieme dei valori delle variabili definite nel programma e di quelle implicite.

Programmi sequenziali Programmi che, su un insieme di dati D si ottiene sempre la stessa traccia.

Pogrammi concorrenti L'esito dell'esecuzione dipende dalla sequenza cronologica di esecuzione delle istruzioni contenute, lo stesso insieme di dati D può dare una traccia diversa, non determinismo.

Verificare che programmi di questo tipo siano corretti non è banale, un semplice debug non garantisce il soddisfacimento di una determinata proprietà.

3.4.1 Proprietà dei programmi

Una proprietà del programma P è un attributo che è sempre vero, data ogni traccia del programma P.

Le proprietà si possono classificare in due categorie:

- **safety properties**
- **liveness properties**

Safety

É una proprietà che garantisce che durante l'esecuzione di P, non si entrerà mai in uno stato "errato".

Liveness

É una proprietà che garantisce che durante l'esecuzione di P, prima o poi si entrerà inuno stato "corretto".

Nel caso di programmi sequenziali, entrambe le proprietà devono essere realizzate, il programma deve restituire un risultato valido per ogni esecuzione e prima o poi terminare.

Per i programmi concorrenti, si aggiungono anche altri fattori alla completezza della safety e liveness:

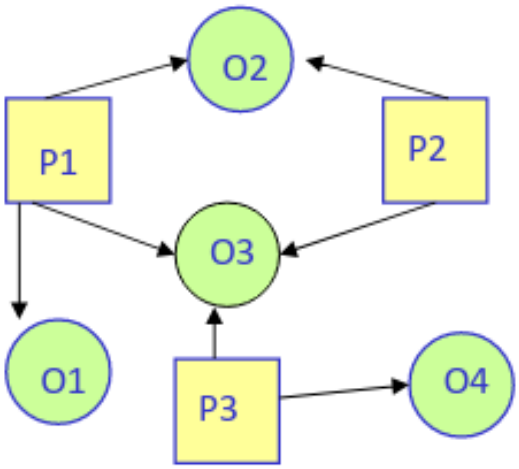
- Mutua esclusione nell'accesso a risorse (safety): nessun processo accede a una risorsa già occupata da un altro processo contemporaneamente
- Assenza di deadlock (safety): per ogni esecuzione non si devono verificare situazioni di blocco
- Assenza di starvation (liveness): prima o poi ogni processo potrà accedere akke risorse richieste

4 — Modello a memoria comune

Esistono due modelli principale di interazione tra i processi:

- memoria comune, ambiente globale con memoria condivisa
- scambio di messaggi, ambiente locale con memoria distribuita

In questo capitolo si analizza il modello a memoria comune.
Il sistema è visto come un insieme di **processi** e **oggetti**.



In questo grafo, O1 e O4 sono risorse private, mentre O2 e O3 sono comuni.
Esistono due tipi di interazione tra processi:

- **competizione**
- **cooperazione**

In questo modello, ogni applicazione viene strutturata come uninsieme di componenti, suddiviso in due sottoinsiemi disgiunti, i processi come componenti attivi e le risorse come componenti passivi.

Risorsa Qualunque oggetto fisico di cui un processo necessita per portare a termine il suo compito.

Le risorse sono raggruppate in classi, categorie che identificano l'insieme delle operazioni che un processo può eseguire.

4.1 Gestore di una risorsa

Per ogni risorsa R, il suo **gestore** definisce, in ogni istante t, l'insieme $SR(t)$ dei processi che hanno il diritto di operare su R.

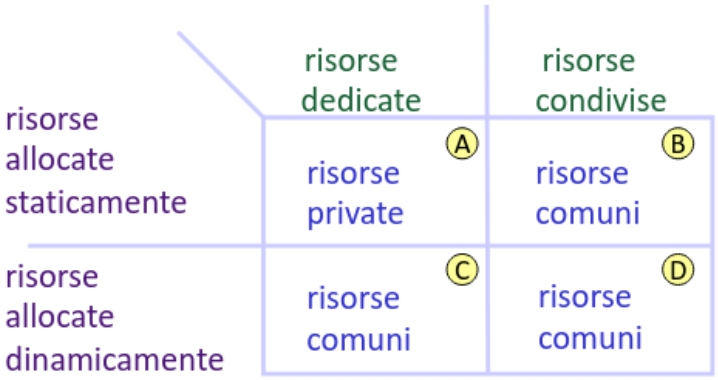
Classificazione delle risorse in base alla condivisione:

- **dedicata** se $SR(t)$ ha una cardinalità sempre ≤ 1
- **condivisa** in caso contrario

Classificazione delle risorse in base al tipo di allocazione:

- **allocata staticamente** se $SR(t)$ è una costante $SR(t) = SR(t_0), \forall t$
- **allocata dinamicamente** se $SR(t)$ è una funzione del tempo

Figura 4.1: Tipologia di allocazione delle risorse



Per ogni risorsa allocata *staticamente*, l'insieme $SR(t)$ è definito prima che il programma inizi la propria esecuzione, il gestore della risorsa è il programmatore che stabilisce quale processo può operare su R.

Per ogni risorsa allocata *dinamicamente*, il gestore G_R definisce l'insieme $SR(t)$ in fase di esecuzione e quindi deve essere un componenete della stessa applicazione, nel quale l'allocazione viene decisa a run-time in base alle politiche date.

4.1.1 Compiti del gestore di una risorsa

Il gestore di una risorsa deve essere in grado di:

- mantenere **aggiornato** l'insieme $SR(t)$ e lo stato di allocazione della risorsa
- fornire i **meccanismi** che un processo può utilizzare per ottenere i permessi per accedere alla risorsa e quindi entrare a far parte dell'insieme $SR(t)$ e per rilasciare questi permessi
- implementare la **strategia** di allocazione della risorsa e cioè definire quando, a chi e per quanto tempo allocare la risorsa

4.1.2 Accesso a risorse

Considerando un processo P che deve operare su una risorsa R di tipo T.

Allocata staticamente

Se R è allocata staticamente a P il processo, se appartiene a $SR(t)$ possiede il diritto di operae su R in qualunque istante.

Allocata dinamicamente

Se R è allocata dinamicamente a P, è necessario prevedere un gestore GR che implementa le funzioni di richiesta e rilascio, il processo deve richiedere accesso, eseguire operazione e rilasciare accesso.

Allocata condivisa

Se R è allocata come *risorsa condivisa* è necessario assicurare che gli accessi avvengano in modo non divisibile: le funzioni di acecsso alla risorsa devono essere programmate come una classe di sezioni critiche utilizzando meccanismi di sincronizzazione).

Allocata dedicata

Se R è allocata come *risorsa dedicata* essendo P l'unico processo che accede alla risorsa, non è necessario prevedere sincronizzazione.

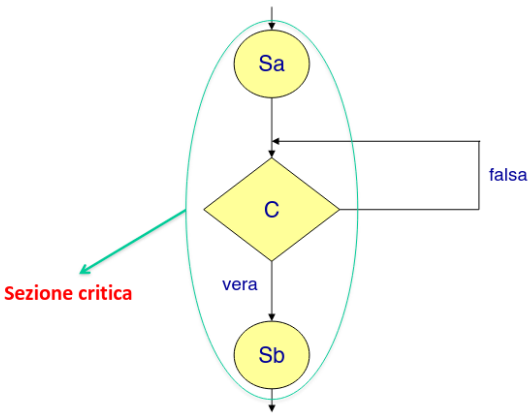
4.1.3 Specifica della sincronizzazione

Regione critica condizionale [Hoare, Brinch-hansen] Formalismo che consente di esprimere la specifica di qualunque **vincolo di sincronizzazione**.

```
region R << Sa; when(C) Sb;>>
```

Il corpo della region rappresenta una operazione da eseguire sulla risorsa condivisa R, è la **sezione critica** che deve essere eseguita in mutua esclusione con le altre operazioni.

Vengono eseguite le istruzioni `Sa` e, non appena C è vera, `Sb`.



4.1.4 Casi particolari

- `region R << S; >>` mutua esclusione senza ulteriori vincoli
- `region R << when(C) >>` specifica di un vincolo di sincronizzazione: P deve attendere che C si verifichi
- `region R << when(C) S; >>` in questo caso C è una precondizione necessaria per eseguire S

4.2 Il problema della mutua esclusione

Il problema della mutua esclusione nasce quando più processi possono avere accesso a variabili comuni, la regola impone che le operazioni non si sovrappongano nel tempo, nessun vincolo è imposto sull'ordine delle operazioni.

Sezione critica Una sequenza di istruzioni che accede e modifica un insieme di variabili comuni prende il nome di sezione critica.

La regola di mutua esclusione stabilisce che sezioni critiche della stessa classe non pososno essere in esecuzione contemporaneamente.

Il protocollo di esecuzione di una sezione critica è il seguente:

```
<prologo>
S;
<epilogo>
```

Nel prologo si richiede e ottiene l'autorizzazione a eseguire la sezione, nell'epilogo si rilascia la risorsa.

4.3 Strumenti linguistici per la programmazione di interazioni

4.3.1 Semaforo

É uno strumento di basso livello, realizzato dal kernel della macchina, utile a risolvere qualsiasi problema di sincronizzazione.

L'eventuale attesa nell'esecuzione può essere realizzata utilizzando i meccanismi di gestione dei thread (sospensione, riattivazione) offerti dal kernel.

Un semaforo è una variabile *interna non negativa* alla quale è possibile accedere solo tramite le due operazioni `P` e `V`.

Specifica delle operazioni di un semaforo:

```
void P(semaphore s):
    region s << when(val>0) val--; >>


void V(semaphore s):
    region s << val++; >>
```

Dato un semaforo S, siano:

- $\texttt{val}_\texttt{s}$: valore dell'intero non negativo associato al semaforo
- $\texttt{I}_\texttt{s}$: valore interno maggiore di zero di inizializzazione
- $\texttt{nv}_\texttt{s}$: numero di volte che l'operazione `V(s)` è stata eseguita
- $\texttt{np}_\texttt{s}$: numero di volte che l'operazione `P(s)` è stata eseguita

Relazione di invarianza

Ad ogni istante possiamo esprimere il valore del semaforo come $\texttt{val}_\texttt{s} = \texttt{I}_\texttt{s} + \texttt{nv}_\texttt{s} - \texttt{np}_\texttt{s}$ da cui si ottiene $\texttt{np}_\texttt{s} \leq \texttt{I}_\texttt{s} + \texttt{nv}_\texttt{s}$

La relazione di invarianza è sempre soddisfatta (safety property), si può usare questa proprietà per dimostrare formalmente le proprietà dei programmi concorrenti.

Utilizzo

Il semaforo è uno strumento generale che consente la risoluzione di qualunque problema di sincronizzazione, ne esistono però specializzazioni utili in particolari casi:

- semafori mutua esclusione
- semafori evento
- semafori binari composti
- semafori condizione
- semafori risorsa
- semafori privati

Sem mutua esclusione

Viene inizializzato a 1 ed è usato per realizzare le sezioni critiche di una stessa classe.

```
class risorsa {
    semaphore mutex = 1;
    public void op1() {
        P(mutex);
        <sez. critica>
        V(mutex);
    }
}
```