

Ripasso Sistemi Operativi

Matteo Franchini

31 agosto 2023

- 1. Introduzione
- 2. Evoluzione dei sistemi operativi
- 3. Appofondimento delle interruzioni
- 4. Servizi e organizzazione del SO
- 5. Virtualizzazione
- 6. Processi
- 7. Thread e concorrenza
- 8. Scheduling
- 9. Sincronizzazione dei processi
- 10. Sincronizzazione dei processi in ambiente globale mediante semafori
- 11. Deadlock
- 12. Sistema di protezione
- 13. Gestione della memoria principale e virtuale

Sommario

1. Introduzione

2. Evoluzione dei sistemi operativi

3. Appofondimento delle interruzioni

4. Servizi e organizzazione del SO

5. Virtualizzazione

6. Processi

7. Thread e concorrenza

8. Scheduling

9. Sincronizzazione dei processi

10. Sincronizzazione dei processi in ambiente globale mediante semafori

11. Deadlock

12. Sistema di protezione

13. Gestione della memoria principale e virtuale

Che cos'è un sistema operativo?

Introduzione

Un programma che funge da intermediario tra l'utente di un computer e l'hardware del computer stesso

Obiettivi del sistema operativo:

- Eseguire i programmi dell'utente e facilitarne la risoluzione dei problemi
- Rendere il sistema informatico comodo da usare
- Utilizzare l'hardware del computer in modo efficiente

Definizione di SO

Introduzione

Il sistema operativo è un allocatore di risorse:

- gestisce tutte le risorse
- decide tra richieste in conflitto per un uso efficiente ed equo delle risorse

Il sistema operativo è un programma di controllo

- controlla l'esecuzione dei programmi per prevenire gli errori e l'uso improprio del computer

Non c'è una definizione universalmente accettata

- con il termine sistema operativo si intende quell'insieme di programmi che provvedono alla gestione Hw e Sw di un sistema di calcolo
- "*Tutto ciò che viene fornite quando si ordina un SO*"

Una definizione alternativa (**Tanenbaum**):

un sistema operativo è un programma che controlla le risorse di un calcolatore e fornisce ai suoi utenti un'interfaccia o macchina virtuale più agevole da utilizzare della macchina "nuda"

L'unico programma che è sempre in esecuzione sul computer è il kernel (nucleo) del SO.

Il resto è

- un programma di sistema (fornite con il SO di cui costituisce una parte)
- un programma applicativo

Sistema Operativo

Introduzione

Può essere visto come:

1. Allocatore di risorse Hw e Sw:

- tempo di CPU, spazio di memoria, dispositivo di I/O, compilatori
- Le risorse devono essere assegnate a programmi specifici secondo determinate politiche

2. Programma di controllo: controlla l'esecuzione dei programmi per prevenire errori ed usi impropri del calcolatore

Obiettivi principali del SO:

- rendere più **semplice** l'uso di un sistema di elaborazione
- rendere più **efficiente** l'uso delle risorse del sistema di elaborazione

Il SO è costituito dall'insieme dei programmi (sw o fw) che **rendono praticamente utilizzabile** l'elaboratore agli utenti cercando contemporaneamente di **ottimizzarne le prestazioni**.

- **Visione top-down**: il sistema operativo come una macchina estesa (**astrazione**)
- **Visione bottom-up**: il sistema operativo come un gestore di risorse (**fornisce protezione, risoluzione dei conflitti**)

Avvio dell'elaboratore

Introduzione

Vi è un **programma di bootstrap** che normalmente è memorizzato in una **memoria non volatile**, inizializza e verifica il corretto funzionamento dei componenti Hw del sistema. **Carica il kernel del SO e inizia l'esecuzione.**

Nei PC più vecchi c'era il **BIOS**, ora invece è stato sostituito con il **UEFI**.

Proprietà fondamentali di un SO

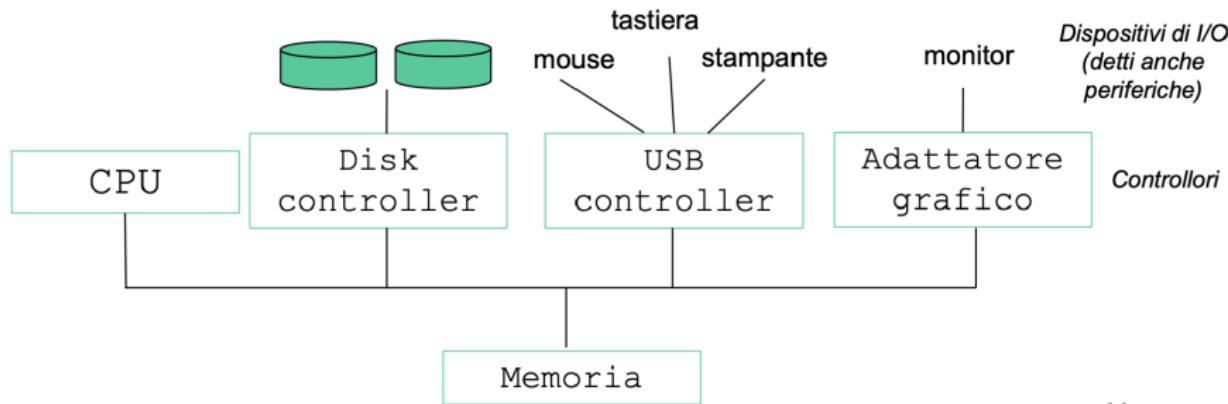
Introduzione

- **Affidabilità**
- **Efficienza**
- **Sicurezza**

Organizzazione di un elaboratore

Introduzione

Uno o più CPU si connettono mediante un **bus condiviso**, vi è l'esecuzione concorrente delle CPU e dei dispositivi che competono per i cicli di memoria



14

Funzionamento di un elaboratore

Introduzione

- I dispositivi di I/O e la CPU possono operare simultaneamente
- Ogni controller di dispositivo è responsabile di un particolare dispositivo
- Ogni controllore di dispositivo ha un buffer locale
- La CPU sposta i dati da/alla memoria principale a/dai buffer locali
- L'I/O è dal dispositivo al buffer locale del controllore
- Il controllore del dispositivo informa la CPU di aver terminato la sua operazione causando un interrupt

Funzioni comuni degli interrupt

Introduzione

La gestione degli interrupt deve salvare l'indirizzo dell'istruzione interrotta

Eccezione

Una trap o eccezione è un'interruzione generata dal Sw e causata da un errore o da una richiesta al SO da parte di un programma

Un sistema operativo è guidato dagli interrupt

Gestione degli interrupt

Introduzione

Il sistema operativo preserva lo stato della CPU memorizzando i registri e il contatore del programma

Struttura dell'I/O

Introduzione

1. Dopo l'avvio dell'I/O, il controllo ritorna al programma utente solo al completamento dell'I/O
 - Un'istruzione *wait* mette a riposo la CPU fino al prossimo interrupt
 - Ciclo di attesa
 - È in sospeso al massimo una richiesta di I/O alla volta, nessuna elaborazione simultanea di I/O
2. Dopo l'avvio dell'I/O, il controllo torna al programma utente senza attendere il completamento dell'I/O
 - **System call:** richiesta al SO per consentire al programma di attendere il completamento dell'I/O
 - La tabella di stato dei dispositivi contiene una voce in cui è indicato l'**indirizzo** e lo **stato**
 - Il SO **indicizza la tabella dei dispositivi di I/O** per determinare lo stato del dispositivo e modificare la voce della tabella per includere interrupt

Funzioni specifiche di gestione

Introduzione

Gestione della memoria centrale

- caricare in memoria programmi e dati
- evitare interferenze fra programmi diversi
- assegnare la memoria in base a criteri di efficienza
- minimizzare i trasferimenti tra memoria centrale e memoria di massa

Gestione della memoria secondaria

- consentire l'accesso all'informazione in base alla sua organizzazione logica anziché fisica
- controllare i diritti di accesso ai file da parte degli utenti
- consentire creazione, modifica, cancellazione dei file

Gestione dei dispositivi periferici

- mascherare al programmatore la complessità delle operazioni di I/O
- effettuare controlli sul corretto funzionamento delle operazioni
- risolvere conflitti nell'utilizzo di una stessa periferica da parte di più programmi

Gestione dei processi

- decidere quale programma userà il processore (**scheduling**) in base a criteri di corretto funzionamento e di efficienza
- verificare che i programmi rilascino il processore entro il tempo stabilito

Funzioni di un SO

Introduzione

- definizione e gestione dell'interfaccia utente
- gestione dei job
- gestione delle risorse di sistema
- ausili per la messa a punto dei programmi
- ausili per la gestione dei dati - file system

Struttura della memoria

Introduzione

Memoria principale

solo supporti di memoria di grandi dimensioni a cui la CPU può accedere direttamente

Memoria secondaria

estensione della memoria principale che **fornisce un'ampia capacità di memorizzazione non volatile**

- Dischi rigidi
- A stato solido

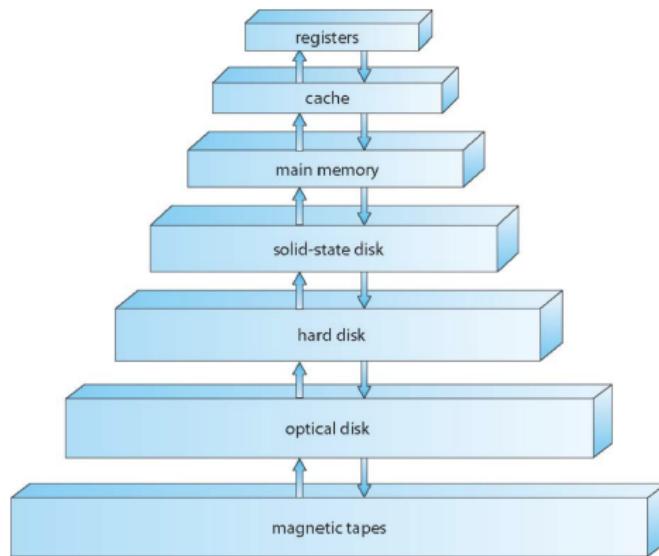
Gerarchia di memoria

Introduzione

Sistemi di storage organizzati in una gerarchia.

Caching: copiare le informazioni in un sistema di archiviazione più veloce

Driver del dispositivo: per ogni controller di dispositivo per gestire l'I/O



Caching

Introduzione

Le informazioni in uso vengono **copiate temporaneamente da una memoria più lenta a una più veloce**

La memoria più veloce (**cache**) viene controllata per prima per determinare se le **informazioni sono lì**, in caso contrario, i dati vengono copiati nella cache e utilizzati lì.

La **cache è più piccola della memoria di cui si fa il caching**, la gestione della cache è un importante problema di progettazione.

Struttura di accesso diretto alla memoria

Introduzione

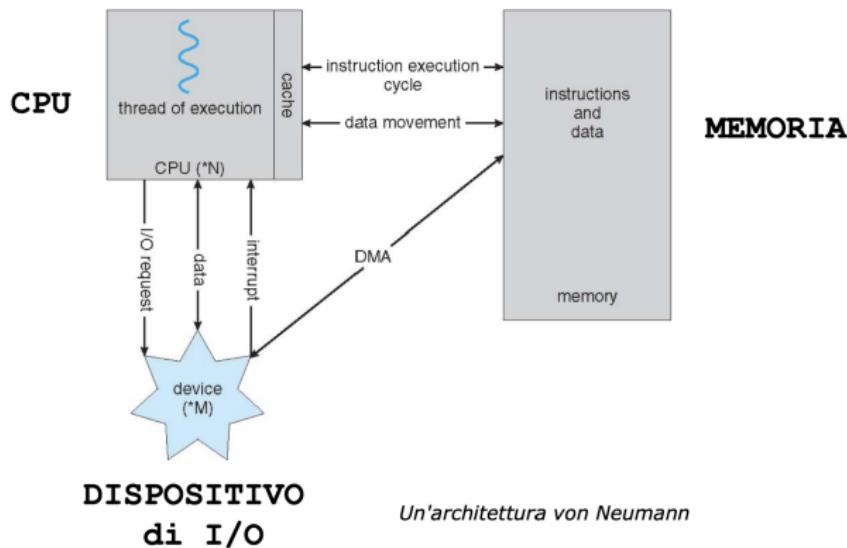
DMA (Direct Memory Access)

Utilizzata per dispositivi di I/O ad alta velocità in grado di trasmettere informazioni a velocità prossime a quelle della memoria.

Il controller del dispositivo trasferisce blocchi di dati dalla memoria tampone direttamente alla memoria principale senza l'intervento della CPU

Funzionamento PC moderno

Introduzione



Architettura del sistema informatico

Introduzione

L'uso e l'importanza dei sistemi multiprocessore è in crescita.

I vantaggi includono:

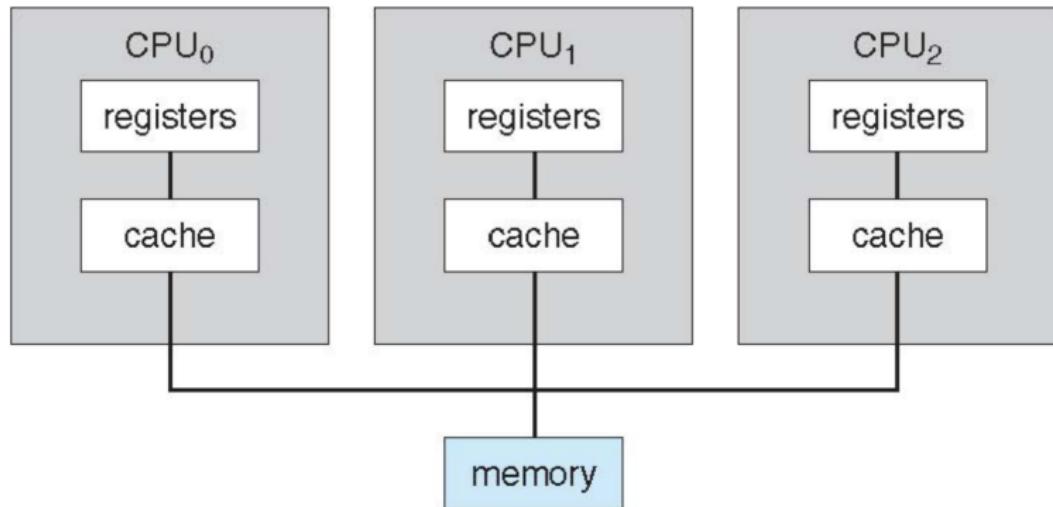
1. **Aumento del throughput**
2. **Economia di scala**
3. Maggiore affidabilità: *graceful degradation o tolleranza ai guasti*

Ci sono due tipi di multiprocesso:

1. **Multiprocesso asimmetrico**: a ogni processore
2. **Multiprocesso simmetrico**: ogni processore esegue tutti i compiti

Architettura del multiprocessore simmetrico

Introduzione



Sistemi cluster

Introduzione

Sono come i sistemi multiprocessore, ma con **più sistemi che lavorano insieme**, di solito condividono lo **storage** tramite una **rete SAN (Storage Area Network)**.

Tipi di cluster

- **Clustering asimmetrico:** prevede una macchina in modalità **hot-standby**
- **Clustering simmetrico:** prevede più nodi che eseguono applicazioni, monitorandosi a vicenda

Alcuni cluster sono destinati al calcolo ad alte prestazioni (HPC), altri hanno un **gestore di lock distribuito (DLM)** per evitare operazioni in conflitto.

Multiprogrammazione e multitasking

Introduzione

Multiprogrammazione (sistema batch)

- Un singolo utente non può tenere occupati CPU e dispositivi di I/O in ogni momento
- La **multiprogrammazione organizza i lavori** (codice e dati) in modo che la CPU ne abbia sempre uno da eseguire
- Un sottoinsieme di lavori totali nel sistema viene tenuto in memoria
- Quando deve aspettare, il sistema operativo **passa a un altro lavoro**

Multitasking

Il **timesharing** è un'estensione logica in cui la **CPU passa ai lavori con una frequenza tale da consentire agli utenti di interagire con ciascun lavoro mentre è in esecuzione**

- Il tempo di risposta deve essere < 1 secondo
- Ogni utente ha almeno un programma in esecuzione
- Se diversi lavori sono pronti per essere eseguiti contemporaneamente abbiamo la **schedulazione della CPU**
- La **memoria virtuale** consente l'esecuzione di processi non completamente in memoria

Operazioni del sistema operativo

Introduzione

Guidato dalle interruzioni (Hw e Sw):

- Interruzione hardware da parte di uno dei dispositivi
- Interruzione software (eccezione o trap): errore, richiesta di servizio, altri problemi

Il funzionamento in **doppia modalità** permette al sistema operativo di proteggere se stesso e gli altri componenti del sistema: **modalità utente e modalità kernel**.

Alcune istruzioni designate come *privilegiate* sono eseguibili solo in **modalità kernel**.

Transizione dalla modalità utente a quella kernel

Introduzione

C'è un **timer per prevenire il loop infinito**.

Il timer è impostato per interrompere il computer dopo un certo periodo di tempo e conserva un **contatore che viene decrementato dall'orologio fisico**.

Quando il **contatore è zero viene generato un interrupt**.

Gestione del processo

Introduzione

Processo

Un processo è un **programma in esecuzione**. È un'unità di lavoro all'interno del sistema. Il programma è un'**entità passiva**, il processo è un'**entità attiva**.

Un **processo a singolo thread** ha un contatore di programma che specifica la posizione della prossima istruzione da eseguire.

Un **processo multi-thread** ha un contatore di programma per ogni thread.

Attività di gestione dei processi

Introduzione

Il sistema operativo è responsabile delle seguenti attività relative alla gestione dei processi:

- Creazione e cancellazione di processi utente e di sistema
- Sospendere e riprendere i processi
- Meccanismi di sincronizzazione dei processi
- Meccanismi di comunicazione tra i processi
- Meccanismi per la gestione dei deadlock

Gestione della memoria di massa

Introduzione

Generalmente i dischi sono utilizzati per memorizzare dati che non stanno nella memoria principale o dati che devono essere tenuti per un periodo di tempo "lungo".

Attività del sistema operativo

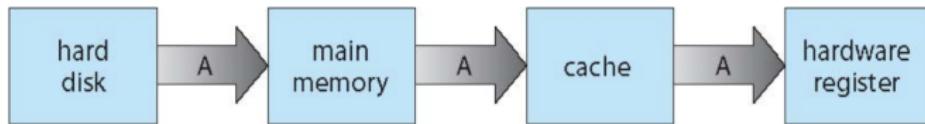
- Gestione dello spazio libero
- Allocazione dello spazio di archiviazione
- Pianificazione dei dischi

Migrazione del dato "A" da un disco ad un registro

Introduzione

Gli ambienti multitasking devono fare attenzione a utilizzare il valore più recente, indipendentemente dalla sua posizione nella gerarchia di memorizzazione.

Gli ambienti multiprocessore devono garantire la coerenza della cache in Hw, in modo che tutte le CPU abbiano il valore più recente nella loro cache



Sottosistema I/O

Introduzione

Uno degli scopi del sistema operativo è quello di **nascondere all'utente le peculiarità dei dispositivi Hw.**

Il sottosistema I/O è responsabile di:

- Gestione della memoria dell'I/O comprende il buffering, il caching in una memoria più veloce, lo spooling
- Interfaccia generale dispositivo-driver
- Driver per dispositivi Hw specifici

Protezione e sicurezza

Introduzione

Protezione

Qualsiasi meccanismo di controllo dell'accesso dei processi o degli utenti alle risorse definite dal SO

Sicurezza

Difesa del sistema da attacchi interni ed esterni

I sistemi in genere distinguono innanzitutto fra gli utenti, per determinare chi può fare cosa:

- Le identità degli utenti (**ID utente**)
- L'ID utente viene poi associato a tutti i file e processi di quell'utente per determinare il controllo degli accessi
- L'identificativo di gruppo (**ID di gruppo**) consente di definire un insieme di utenti e di gestire i controlli, quindi anche di associarlo a ciascun processo, file o altro
- L'**escalation dei privilegi** consente all'utente di passare ad un ID effettivo con maggiori diritti

Ambienti di elaborazione - Tradizionali

Introduzione

I **portali** forniscono accesso web ai sistemi interni. I **computer in rete** sono come terminali web.

I computer mobili si collegano tramite **reti wireless**

Ambienti informatici - Mobile

Introduzione

Permette nuovi tipi di applicazioni come la realtà aumentata.
Utilizza reti wireless IEEE 802.11 o reti dati cellulari per la connettività

Ambienti di elaborazione - Distribuiti

Introduzione

Collezione di **sistemi separati**, eventualmente eterogenei, collegati in rete tra loro.

La **rete** è un percorso di comunicazione, il **TCP/IP** è il più comune:

- Rete Locale (LAN)
- Rete geografica (WAN)
- Rete metropolitana (MAN)
- Rete personale (PAN)

Ambienti informatici - Virtualizzazione

Introduzione

Emulazione

Utilizzata quando il tipo di CPU di origine è diverso da quello di destinazione.

Questo metodo è generalmente più lento.

Quando il linguaggio del computer non viene compilato in codice nativo siamo di fronte all'**interpretazione**

Virtualizzazione

Sistema operativo compilato in modo nativo per la CPU, che esegue sistemi operativi **guest** anch'essi compilati in codice nativo

Ambienti di elaborazione - Cloud Computing

Introduzione

Fornisce elaborazione, storage e persino applicazioni come servizio attraverso una rete.

Estensione logica della virtualizzazione perché utilizza la virtualizzazione come base per le sue funzionalità.

Alcuni tipi:

- **Cloud pubblico:** disponibile via internet a chiunque sia disposto a pagare
- **Cloud privato:** gestito da un'azienda per uso proprio
- **Cloud ibrido:** include componenti di cloud pubblico e privato
- **SaaS (Software as a Service):** uno o più applicazioni disponibili via internet
- **PaaS (Platform as a Service):** stack di software pronto per l'uso di applicazioni via internet
- **IaaS (Infrastructure as a Service):** server o storage disponibili via internet

Aree di applicazione di un SO

Introduzione

- **Sistemi di tipo generale**
- **Sistemi in tempo reale**
 - applicazioni per il controllo di processo e di apparati fisici
 - applicazioni interattive, interrogazione di basi di dati, query web

Ambienti di elaborazione - Sistemi embedded in tempo reale

Introduzione

I sistemi **embedded in tempo reale** sono la forma più diffusa di computer.

Il sistema operativo in tempo reale ha vincoli temporali fissi ben definiti

- L'elaborazione deve essere eseguita entro i vincoli
- Operazione corretta solo se i vincoli sono rispettati

Sistemi operativi open source

Introduzione

Sistemi operativi resi disponibili in formato di codice sorgente piuttosto che solo binari closed-source.

Avviato dalla **Free Software Foundation (FSF)**, che ha una licenza pubblica GNU (GPL).

Esempi: GNU/LINUX

Sistemi operativi open source

Introduzione

Sistemi aperti

- Realizzati e mantenuti da comunità di sviluppatori volontari
- Movimento per il software "open source"
- Le applicazioni beneficiano della piena conoscenza del SO, il SO evolve nel tempo e migliora grazie alla sua trasparenza
- Esempio canonico: Linux
- Interfaccia utente e **look and feel** sono modificabili

Utenti del SO

Introduzione

- **Utenti finali del sistema:** per essi il sistema operativo è trasparente
- **Programmatori applicativi:** utilizzano i servizi del SO per la realizzazione e l'esecuzione dei loro programmi
- **Programmatori di sistema:** aggiornano e modificano i programmi del SO per adeguarli a nuove necessità del sistema o degli utenti applicativi
- **Operatori:** controllano il funzionamento e rispondono alle richieste di intervento da parte del sistema
- **Amministratore del sistema:** stabilisce le politiche di gestione del sistema e ne cura l'osservanza

Tipi di SO

Introduzione

Sistemi proprietari

- Progettati da costruttori al fine di sfruttare in modo ottimale le risorse di ciascun tipo di macchina
- Programmi utente e applicazioni si interfacciano al SO in modo diverso tra le diverse famiglie di sistemi
- IBM: OS/360 370, VM, MVS

Sistemi standard

- Progettati da aziende software o da grandi utenti per consentire lo sviluppo di applicazioni portabili su sistemi diversi
- Interfaccia di programmazione con cui le applicazioni interagiscono con il SO rimane costante nelle diverse versioni
- Esempi: UNIX, MS-DOS

Proprietà fondamentali di un SO

Introduzione

- **Architettura:** com'è organizzato?
- **Condivisione:** quali risorse vengono condivise?
- **Efficienza:** come massimizzare l'utilizzo delle risorse disponibili?
- **Affidabilità/tolleranza ai guasti**
- **Estensibilità**
- **Protezione e sicurezza**
- **Conformità a standard**

Sommario

1. Introduzione

2. Evoluzione dei sistemi operativi

3. Appocondimento delle interruzioni

4. Servizi e organizzazione del SO

5. Virtualizzazione

6. Processi

7. Thread e concorrenza

8. Scheduling

9. Sincronizzazione dei processi

10. Sincronizzazione dei processi in ambiente globale mediante semafori

11. Deadlock

12. Sistema di protezione

13. Gestione della memoria principale e virtuale

Stadi evolutivi e modalità d'uso dei sistemi

Evoluzione dei sistemi operativi

- Sistema isolato ('50-'60): batch
- Sistema centralizzato ('60-'70): remote job entry, teleprocessing, time sharing
- Sistemi decentrati ('70-): minicalcolatore, controllo real-time, data logging
- Sistemi distribuiti ('80-): multiprocessori, reti locali

Client-Server

Evoluzione dei sistemi operativi

I dispositivi **client** sono utilizzati dagli utenti, molti sistemi sono **server** che rispondono alle richieste di servizio dei client offrendo: servizi di calcolo o gestione (DBMS), servizi di gestione dei file (NFS), altri servizi

Sistema di elaborazione distribuito

Evoluzione dei sistemi operativi

È costituito da nodi tra loro collegati in ciascuno dei quali sono presenti capacità di: **elaborazione, memorizzazione, comunicazione**.

Vantaggi:

- Tolleranza al guasto: un guasto non provoca l'arresto del sistema, ma solo una riduzione delle prestazioni
- Prestazioni: l'elaborazione di norma è effettuata nel posto stesso di utilizzazione, si ha un miglioramento delle prestazioni
- Condivisione: la capacità di elaborazione, i programmi ed i dati esistenti nell'intero sistema sono patrimonio comune di tutti gli utenti

Evoluzione dei SO

Evoluzione dei sistemi operativi

- **Primi calcolatori:** privi di SO, problemi di complessità di operazioni, inefficienza
- **Prima generazione ('50-'60):** virtualizzazione dell'I/O, librerie di controllo dei device, problemi di debug, nasce il **sistema operativo** come stratificazione successiva di funzioni volte ad aumentare l'efficienza e la semplicità d'uso della macchina
- **Seconda generazione ('60-'65):** indipendenza tra programmi e dispositivi usati, parallelizzazione degli utenti tramite **multiprogrammazione e time sharing**
- **Terza generazione ('65-'75):** SO unico per una famiglia di elaboratori, risorse virtuali, sistemi multifunzione

- **Quarta generazione ('75-'85)**: sistemi a macchine virtuali, sistemi multiprocessore e distribuiti
- **Quinta generazione ('85-'95)**: elaboratori personali, reti locali, avvio di internet
- **Sesta generazione ('95-'05)**: elaborazione distribuita, peer to peer, servizi online

Tecniche di gestione di un sistema di calcolo

Evoluzione dei sistemi operativi

Monoprogrammazione

Gestisce in **modo sequenziale** nel tempo i diversi programmi. L'inizio di un programma avviene solamente dopo il completamente del programma precedente.

Tutte le risorse sono dedicate ad un solo programma.

Bassa utilizzazione delle risorse

$$\text{uso CPU} = \frac{T_p}{T_t}$$

T_p = tempo dedicato dalla CPU all'esecuzione del programma

T_t = tempo totale di permanenza nel sistema del programma

Sistema multiprogrammato

Carica in memoria e gestisce **simultaneamente più programmi indipendenti**, nel senso che ciascuno di essi può iniziare o proseguire l'elaborazione prima che un altro sia terminato.
Le risorse risultano meglio utilizzate in quanto **riducono i tempi morti**

Gestione Batch

Evoluzione dei sistemi operativi

Significa raggruppare i lavori o programmi in **lotti** per conseguire una **maggior utilizzazione delle risorse**, cioè un *throughput* più elevato.

Nel caso di multiprogrammazione il SO deve provvedere algoritmi per la scelta di quell'insieme di programmi che, in esecuzione contemporanea, massimizza il throughput.

La **gestione batch** può essere **locale** (unità centrale direttamente collegata ai dispositivi di I/O) o **remota** (è presente una trasmissione dei job e dei risultati ed eventualmente una memorizzazione intermedia).

Time sharing

Evoluzione dei sistemi operativi

L'elaboratore serve **simultaneamente** una pluralità di utenti, dotati di terminali, dedicando a ciascuno di essi tutte le risorse del sistema per **quanti fissati di tempo**.

Migliora i **tempi di risposta** ma peggiora l'utilizzazione delle risorse. Normalmente una modalità di gestione **time-sharing** è adottata nei **sistemi conversazionali**, in cui più utenti contemporaneamente "colloquiano" con il sistema

Spooling

Evoluzione dei sistemi operativi

Per accrescere la velocità di esecuzione dei programmi conviene utilizzare la **memoria a disco per simulare i dispositivi di I/O**: il disco viene usato come un buffer di grosse dimensioni a cui accedono sia il lettore di schede che la CPU.

I trasferimenti lettore di scheme-memoria di massa (**spool-in**) e memoria di massa-stampante (**spool out**) sono effettuati da appositi programmi detti di **spooling**.

Sommario

1. Introduzione

2. Evoluzione dei sistemi operativi

3. Appfondimento delle interruzioni

4. Servizi e organizzazione del SO

5. Virtualizzazione

6. Processi

7. Thread e concorrenza

8. Scheduling

9. Sincronizzazione dei processi

10. Sincronizzazione dei processi in ambiente globale mediante semafori

11. Deadlock

12. Sistema di protezione

13. Gestione della memoria principale e virtuale

Interruzioni

Appofondimento delle interruzioni

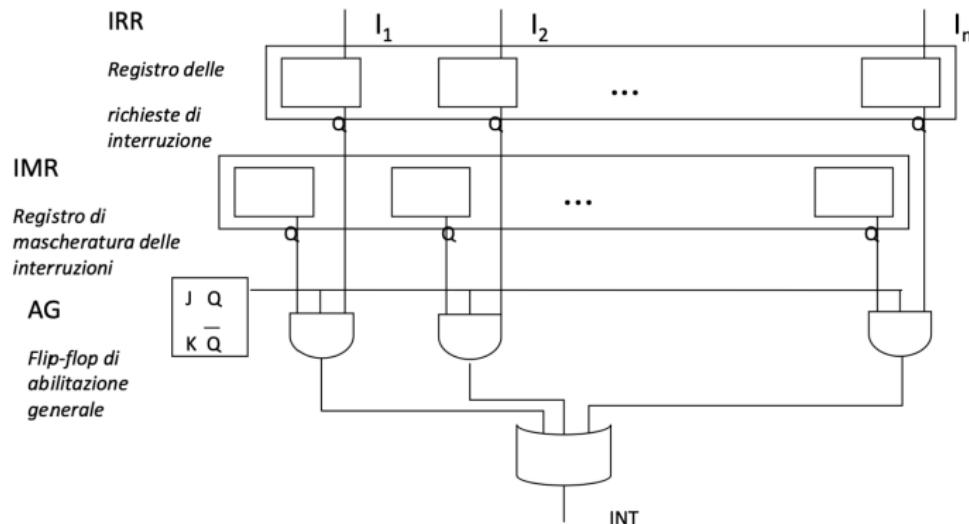
1. **Interruzioni hardware (asincrone)**: per terminazione di un trasferimento dati o per condizione di errore rilevata dalle o nelle periferiche
2. **Interruzioni software (sincrone)**: eccezioni o trap (es. divisione per zero), programmate o supervisory call (per utilizzare funzioni del SO)

Sistema delle interruzioni

Appofondimento delle interruzioni

A ciascuna causa di interruzione è associata un'azione che verrà effettuata dal programma di risposta alle interruzioni.

Una causa **non produce di per se un'interruzione** ma solo una **richiesta di interruzione**. Affinché la richiesta di interruzione segua effettivamente un'interruzione è necessario che **la causa di interruzione sia abilitata**.



Sistema delle interruzioni

Appofondimento delle interruzioni

L'interruzione "i" si manifesta se:

- $AG = 1$ (il sistema di interruzione è abilitato)
- $IRRi = 1$ (si è presentata la causa di interruzione "li")
- $IMRi = 1$ (l'interruzione "i" è abilitata nel registro di maschera)

Processo delle interruzioni

Appofondimento delle interruzioni

Al verificarsi di un'interruzione occorre

- 1. salvare tutte le informazioni necessarie per la ripresa del programma interrotto**

l'Hw provvede in generale a salvare PC e PSW, mentre i registri generali vengono tipicamente salvati in software. È necessario che il salvataggio dei registri avvenga ad interrupt disabilitati.

- 2. individuare la causa dell'interruzione**

l'Hw può trasferire il controllo sempre al medesimo programma di risposta, che provvederà quindi ad individuare la causa dell'interruzione tramite *skip chain*

- 3. eseguire le azioni richieste (servizio dell'interruzione)**

Durante il servizio dell'interruzione il sistema delle interruzioni può essere riabilitato (AG), eventualmente selettivamente (IMR)

- 4. ripristinare lo stato del programma interrotto e riavviato**

Il ripristino dei registri deve avvenire ad interrupt disabilitati.

Un'apposita istruzione di ritorno dell'interrupt (RTI) ripristina i registri.

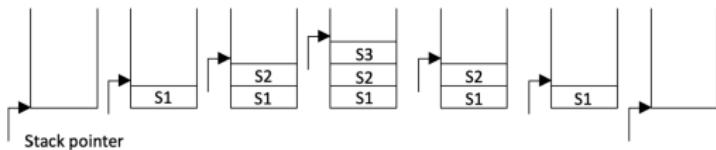
Livelli di priorità

Appofondimento delle interruzioni

I diversi tipi di fonti di interruzione possono suggerire una gestione con diversi livelli di priorità, se all'atto del ritorno esistono più richieste pendenti si serve di quella di livello più elevato.

Salvataggio dello stato tramite stack

Approfondimento delle interruzioni



S1: stato del processore salvato all'arrivo della prima interruzione

S2: stato del processore relativo all'esecuzione del programma di risposta della prima interruzione salvato all'arrivo della seconda interruzione

S3: stato del processore relativo all'esecuzione del programma di risposta della seconda interruzione salvato all'arrivo della terza interruzione

Programma di risposta ad interruzione di livello L:

- salvo lo stato del processore nello stack e modifica SP
- abilita interruzioni di livello $K > L$
- esegue programma di risposta
- ripristina lo stato del processore modificando SP

Sommario

1. Introduzione

2. Evoluzione dei sistemi operativi

3. Appocondimento delle interruzioni

4. Servizi e organizzazione del SO

5. Virtualizzazione

6. Processi

7. Thread e concorrenza

8. Scheduling

9. Sincronizzazione dei processi

10. Sincronizzazione dei processi in ambiente globale mediante semafori

11. Deadlock

12. Sistema di protezione

13. Gestione della memoria principale e virtuale

Servizi del sistema operativo

Servizi e organizzazione del SO

Forniscono un ambiente per l'esecuzione di programmi e servizi a programmi e utenti:

- **Interfaccia utente**
- **Esecuzione del programma**
- **Operazioni di I/O**
- **Manipolazione del file system**
- **Comunicazioni**
- **Rilevamento di errori**

Permette anche il funzionamento efficiente del sistema stesso attraverso la condivisione delle risorse

- **Allocazione delle risorse:** quando più utenti o più job vengono eseguiti simultaneamente
- **Contabilità:** per tenere traccia di quali utenti utilizzano la quantità e il tipo di risorsa del computer
- **Protezione e sicurezza**

Interfaccia utente del SO - CLI

Servizi e organizzazione del SO

La CLI consente l'immissione diretta di comandi, a volte implementato nel **kernel**, a volta da un **programma di sistema**.
A volte sono implementate più versioni: **shell**

Interfaccia utente del SO - GUI

Servizi e organizzazione del SO

Interfaccia desktop con metafora facile da usare.

Molti sistemi includono sia **CLI che GUI**:

- Microsoft Windows è un'interfaccia GUI con una shell di comando CLI
- Apple Mac OS X è una GUI "Aqua" con kernel UNIX

Chiamate di sistema (SysCall)

Servizi e organizzazione del SO

Interfaccia di programmazione per i servizi forniti dal SO, tipicamente scritta in C o C++.

I programmi vi accedono tramite un'**interfaccia di programmazione delle applicazioni (API)** di alto livello.

Le **API** più comuni sono **Win32 API**, **POSIX API** (per tutte le versioni di UNIX)

Esempio di API standard

Servizi e organizzazione del SO

Come API standard, si consideri la funzione

`read()`

si ottiene dalla pagina *man* invocando il comando

`man read`

Implementazione della chiamata di sistema

Servizi e organizzazione del SO

In genere, a ciascuna SysCall è associato un numero (l'interfaccia delle chiamate di sistema mantiene una tabella indicizzata in base a questi numeri).

L'interfaccia SysCall invoca la chiamata di sistema prevista nel kernel del SO e restituisce lo stato della chiamata di sistema e gli eventuali valori di ritorno.

Il chiamante non deve sapere nulla di come viene implementata la SysCall.

Passaggio dei parametri della SysCall

Servizi e organizzazione del SO

Spesso sono necessarie più informazioni della semplice identità della chiamata di sistema desiderata.

Tre metodi generali utilizzati per passare parametri al SO

- Passarli nei registri (più semplice)
- Parametri memorizzati in un blocco, o tabella, in memoria
- Parametri messi sullo stack dal programma

Tipi di System Call

Servizi e organizzazione del SO

- **Controllo del processo**
 - creare un processo, terminare un processo
 - caricare, eseguire
 - allocare e liberare memoria
 - debugger per determinare i bug, esecuzione a passo singolo
 - blocchi per gestire l'accesso ai dati condivisi
- **Gestione dei file**
- **Gestione dei dispositivi**
 - richiedere un dispositivo, rilasciare un dispositivo
- leggere, scrivere, riposizionare
- **Manutenzione delle informazioni**
- **Comunicazione**
 - inviare, ricevere messaggi nel **modello a scambio di messaggi**
 - creare e accedere a regioni di memoria nel **modello a memoria condivisa**
- **Protezione**

Esempio: MS-DOS

Servizi e organizzazione del SO

- Lavoro a task singolo
- Shell invocata all'avvio del sistema
- Metodo semplice per eseguire un programma
- Singolo spazio di memoria
- Uscita dal programma → viene ricaricato lo shell

Esempio: FreeBSD

Servizi e organizzazione del SO

- Variante UNIX
- Multitasking
- Lo shell esegue la chiamata di sistema *fork()* per creare un processo
- Il processo esce con
 - *codice* = 0 - nessun errore
 - *codice* > 0 - codice errore

Servizi di sistema

Servizi e organizzazione del SO

I programmi di sistema forniscono un comodo **ambiente per lo sviluppo e l'esecuzione dei programmi**. Possono essere suddivisi in:

- Manipolazione dei file
- Informazioni di stato a volte memorizzate in un file
- Supporto ai linguaggi di programmazione
- Caricamento ed esecuzione dei programmi
- Comunicazioni
- Servizi di background: avvio a tempo di boot, servizi come il controllo del disco
- Programmi applicativi

Linker e loader

Servizi e organizzazione del SO

Il **linker** combina i file oggetto in singoli file eseguibili e il **loader** li carica in memoria

Programmi e SO

Servizi e organizzazione del SO

Application Binary Interface (ABI) è un'architettura equivalente all'API che definisce come differenti componenti del codice possono interfacciarsi ad un dato sistema operativo o ad una data architettura

Struttura del sistema operativo

Servizi e organizzazione del SO

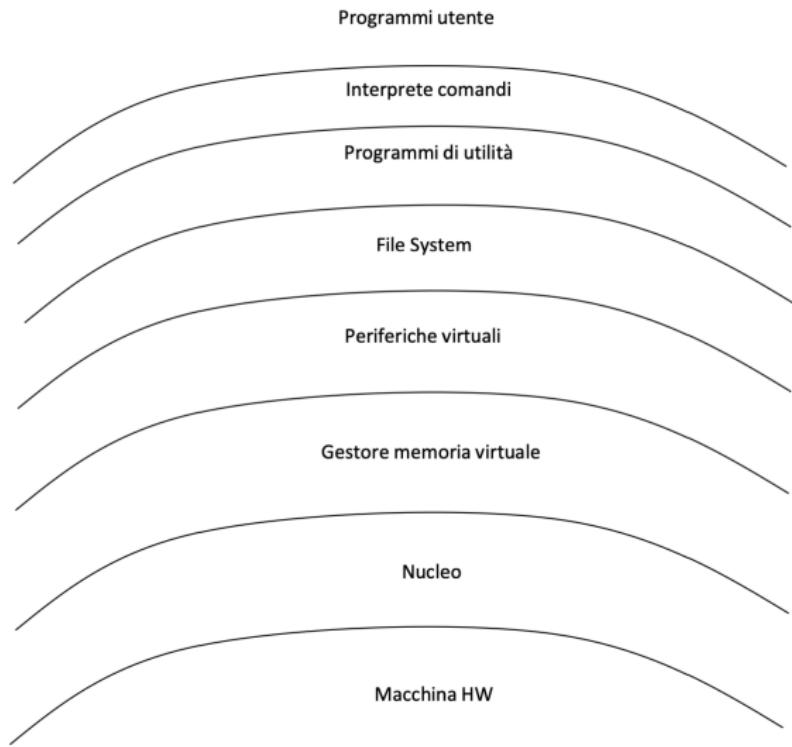
Sistema a livelli: il livello più **interno** è l'Hw, quello più **esterno** è l'**interfaccia utente**

Ci sono vari modi per strutturare un SO:

- Struttura semplice **MS-DOS**
- Più complessa **UNIX**
- A strati **un'astrazione**
- Microkernel **Mach**

Struttura a strati

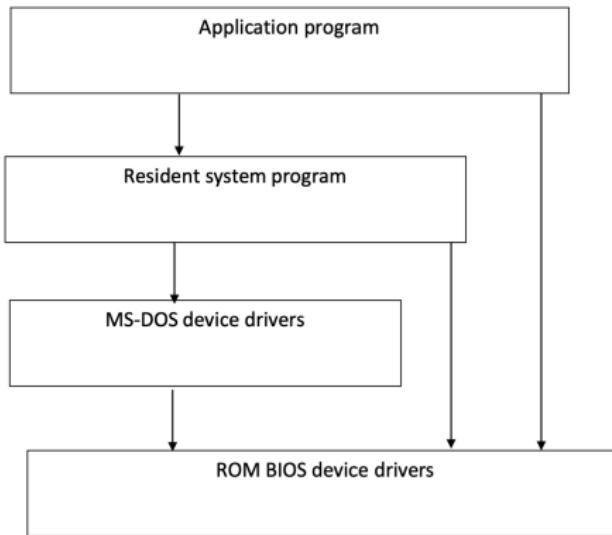
Servizi e organizzazione del SO



Struttura a livelli di MS-DOS

Servizi e organizzazione del SO

- MS-DOS - scritto per fornire la maggior parte delle funzionalità nel minor spazio possibile
 - Non suddiviso in moduli
 - Sebbene MS-DOS abbia una certa struttura, le sue interfacce e i suoi livelli di funzionalità non sono ben separati



Struttura non semplice - UNIX

Servizi e organizzazione del SO

UNIX - limitato dalla funzionalità Hw, il sistema operativo UNIX originale aveva una struttura limitata.

UNIX è composto da due parti separabili: **programmi di sistema, kernel** (ovvero tutto quello che trova sotto l'interfaccia delle chiamate di sistema e al di sopra dell'Hw fisico)

Struttura monolitica

(Users)			
Shells and commands Compilers and interpreters System libraries			
<i>System-call interface to the kernel</i>			
Signals Terminal handling Character I/O system Terminal drivers			
File system swapping block I/O system disk and tape drivers	CPU scheduling Page replacement Demand paging Virtual memory		
<i>Kernel interface to the hardware</i>			
Terminal controllers terminals	Device controllers Disk and tapes	memory controllers physical memory	

Con i moduli del kernel caricabili dinamicamente, supportati dai moderni SO incluso Linux, si può avere un kernel è costituito da un insieme di componenti di base, integrati da funzionalità aggiunte dinamicamente all'avvio o in esecuzione per mezzo di moduli.

113

Microkernel

Servizi e organizzazione del SO

Sposta il più possibile dal kernel allo spazio comune.

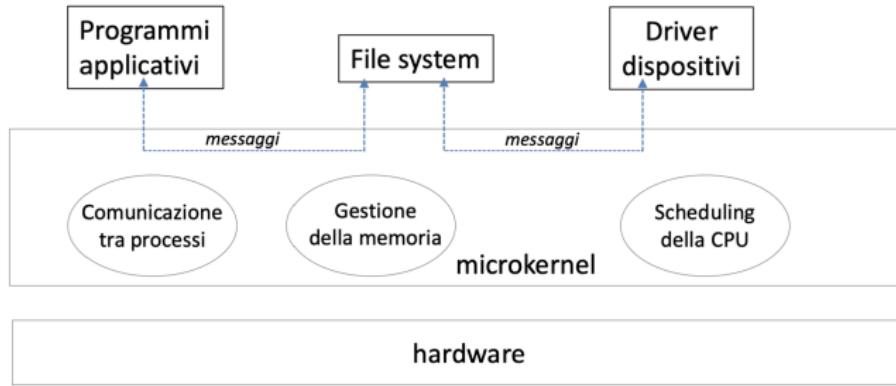
Esempio di **microkernel per Mach**: la comunicazione tra i moduli utente avviene tramite il passaggio di messaggi

Vantaggi: estensione più semplice del SO, più affidabile, più sicuro

Svantaggi: sovraccarico di prestazioni della comunicazione tra spazio utente e spazio kernel

Architettura a microkernel

Servizi e organizzazione del SO



Esempi di microkernel: Darwin (componente kernel dei SO dei sistemi operativi macOS e iOS) contiene anche il microkernel Mach, QNX Neutrino alla base del SO real-time QNX per sistemi embedded

Moduli

Servizi e organizzazione del SO

Molti sistemi operativi moderni implementano moduli del kernel caricabili (approccio orientato agli oggetti, ogni componente centrale è separato, ciascuno parla con gli altri attraverso interfacce note).

Nel complesso è simile ai livelli ma con una maggiore flessibilità

La **maggior parte dei SO moderni non sono un unico modello puro**, l'ibrido combina più approcci per rispondere alle esigenze delle prestazioni. I kernel di Linux sono nello spazio degli indirizzi del kernel, quindi monolitici, ma anche modulari per il caricamento dinamico delle funzionalità.

Windows per lo più monolitico, più microkernel per le diverse personalità dei sottosistemi.

Sistema operativo mobile di Apple, strutturato su Mac OS X con funzionalità aggiuntive.

API Cocoa Touch Objective-C per lo sviluppo delle applicazioni.
Livelli di servizi multimediali per grafica, video e audio.

Sviluppato principalmente da Google ed ha uno stack simile a quello di iOS (open source).

Basato sul kernel Linux ma modificato, l'ambiente di runtime include un set di librerie di base e la macchina virtuale Dalvik

Debug del sistema operativo

Servizi e organizzazione del SO

Il debug consiste nel trovare e correggere gli errori o i bug.

I SO generano file di log contenenti informazioni sugli errori.

Il fallimento di un'app può generare un file di core dump che cattura la memoria del processo.

Il fallimento del SO può generare un file di crash dump contenente la memoria del kernel

Lo strumento **DTrace** in Solaris, FreeBSD e MacOS X consente la strumentazione live sui sistemi di produzione.

Le sonde si attivano quando il codice viene eseguito all'interno di un provider, catturando i dati di stato e inviandoli ai consumatori di tali sonde.

Generazione del sistema operativo

Servizi e organizzazione del SO

- I sistemi operativi sono progettati per funzionare su una qualsiasi classe di macchine
- Il programma **SYGEN** ottiene informazioni sulla configurazione specifica del sistema Hw

Avvio del sistema

Servizi e organizzazione del SO

Quando l'alimentazione viene inizializzata sul sistema, l'esecuzione inizia da una posizione di memoria fissa.

Il SO deve essere reso disponibile all'Hw in modo che quest'ultimo possa avviarlo.

Il comune **bootstrap loader** permette di selezionare il kernel da più dischi, versioni, opzioni del kernel.

Struttura del SO

Servizi e organizzazione del SO

La stratificazione più opportuna può risultare non evidente, dipende dall'evoluzione tecnologica dell'Hw.

Sistema a macchine virtuali (VM IBM): usando lo scheduling della CPU e la tecnica della memoria virtuale si possono creare macchine virtuali, una per ogni processo

Realizzazione in linguaggi ad alto livello.

Nucleo o kernel: mette a disposizione le SysCall ai programmi di sistema ed applicativi

Nucleo di un SO

Servizi e organizzazione del SO

Fornisce un meccanismo per la creazione e la distruzione dei processi.
Provvede allo **scheduling della CPU**, alla **gestione della memoria** e dei
dispositivi di I/O.

Fornisce strumenti per la sincronizzazione

Sommario

1. Introduzione

2. Evoluzione dei sistemi operativi

3. Appofondimento delle interruzioni

4. Servizi e organizzazione del SO

5. Virtualizzazione

6. Processi

7. Thread e concorrenza

8. Scheduling

9. Sincronizzazione dei processi

10. Sincronizzazione dei processi in ambiente globale mediante semafori

11. Deadlock

12. Sistema di protezione

13. Gestione della memoria principale e virtuale

Macchine virtuali

Virtualizzazione

Creano un'illusione di processi multipli, ciascuno in esecuzione sul suo processore privato e con la propria memoria virtuale privata, messa a disposizione dal proprio kernel del SO, che può essere diverso per processi diversi

Virtualizzazione

Virtualizzazione

Dato un sistema caratterizzato da un insieme di risorse, virtualizzare il sistema significa presentare all'utilizzatore una visione delle risorse del sistema diversa da quella reale.

Obiettivo: disaccoppiare il comportamento delle risorse Hw e Sw di un sistema di elaborazione, così come viste dall'utente, dalla loro realizzazione fisica

Esempi di virtualizzazione

Virtualizzazione

Astrazione: in generale un oggetto astratto (risorse virtuale) è la rappresentazione semplificata di un oggetto (risorsa fisica)

Linguaggio di programmazione: la capacità di portare lo stesso programma su architetture diverse è possibile grazie alla **definizione di un VM** in grado di **interpretare ed eseguire ogni istruzione del linguaggio**

Virtualizzazione a livello di processo: i sistemi multitasking permettono la contemporanea esecuzione di più processi, ognuno dei quali dispone di una VM dedicata

Sistemi operativi per la virtualizzazione

Virtualizzazione

La macchina fisica viene trasformata in n interfacce (VM), ognuna delle quali è una replica della macchina fisica.

Su ogni macchina virtuale è possibile installare ed eseguire un sistema operativo: **VM monitor**

Virtualizzazione di sistema

Virtualizzazione

Il disaccoppiamento è realizzato da un componente chiamato VM monitor il cui compito è consentire la condivisione da parte di più macchine virtuali di una singola piattaforma Hw

Il VMM è il mediatore unico nelle interazioni tra le macchine virtuali e l'Hw sottostante, che garantisce

- **Isolamento tra le VM**
- **Stabilità del sistema**

VMM di sistema vs VMM ospitato

Virtualizzazione

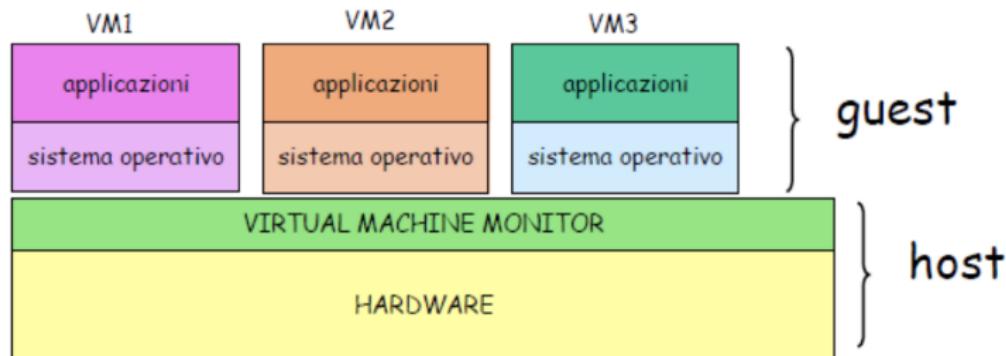
VMM di sistema: le funzionalità di virtualizzazione vengono integrate in un SO leggero, costituendo un unico sistema posto direttamente sopra l'Hw dell'elaboratore.

È necessario corredare il VMM di tutti i driver necessari per pilotare le periferiche

VMM di sistema

Virtualizzazione

- **Host:** piattaforma di base sulla quale si realizzano macchine virtuali
- **Guest:** la VM



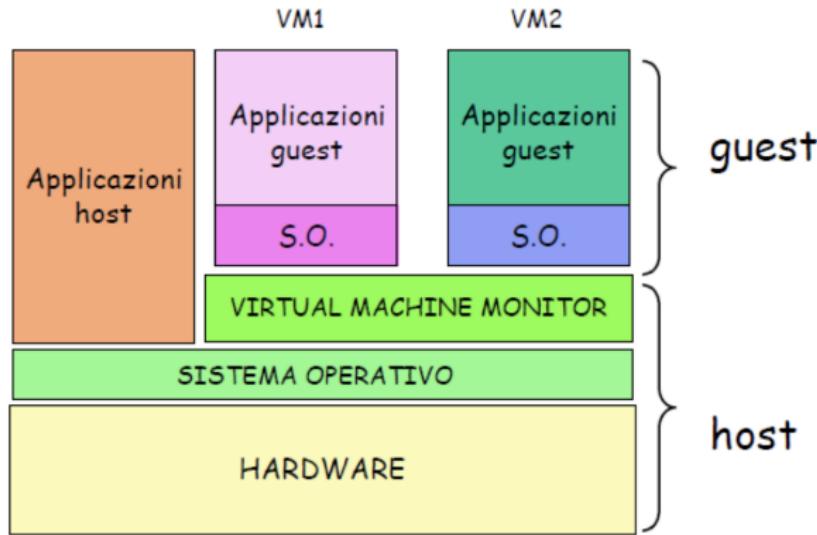
VMM di Sistema

VMM ospitato

Virtualizzazione

Il VMM viene installato come un'app sopra un sistema operativo esistente, che opera nello spazio utente e accede all'Hw tramite le SysCall del SO su cui viene installato.

Prodotti: User Mode Linux, VMWare, Microsoft Virtual Server



Emulazione vs Virtualizzazione

Virtualizzazione

- **Emulazione**

- eseguire app (o SO) compilate per un'architettura su un'altra
- uno strato sw che emula le funzionalità dell'architettura

- **Virtualizzazione**

- definizione di contesti di esecuzione multipli su di un singolo processore, partizionando le risorse

Vantaggi della virtualizzazione

Virtualizzazione

**Uno o più SO sulla stessa macchina fisica
Isolamento dell'ambiente di esecuzione**

Vantaggi della virtualizzazione

Virtualizzazione

Consolidamento Hw: possibilità di concentrare più macchine su un'unica architettura Hw per un utilizzo efficiente dell'Hw

Gestione facilitata della macchine: migrazione a caldo di macchine virtuali tra macchine fisiche, ovvero la possibilità di manutenzione Hw senza interrompere i servizi forniti dalle macchine virtuali, disaster recovery

Virtualizzazione a livello del SO

Virtualizzazione

Una modalità di virtualizzazione in cui il kernel consente l'esistenza di molteplici istanze isolate nello spazio utente.

Consentono di migliorare la sicurezza, l'indipendenza dall'Hw e la gestione delle risorse.

Rispetto alla **virtualizzazione piena (basata su VMM)** l'overhead è **inferiore ma è anche inferiore la flessibilità**

Sommario

1. Introduzione

2. Evoluzione dei sistemi operativi

3. Appofondimento delle interruzioni

4. Servizi e organizzazione del SO

5. Virtualizzazione

6. Processi

7. Thread e concorrenza

8. Scheduling

9. Sincronizzazione dei processi

10. Sincronizzazione dei processi in ambiente globale mediante semafori

11. Deadlock

12. Sistema di protezione

13. Gestione della memoria principale e virtuale

Concetto di processo

Processi

Un SO esegue una serie di programmi:

- Sistema batch - lavori (job)
- Sistema time-sharing - programmi o attività dell'utente

Processo

È un **programma in esecuzione**, l'esecuzione del processo deve procedere in modo sequenziale

Parti multiple

- Codice del programma
- Attività corrente
- Stack che contiene dati temporanei
- Sezione dati
- Heap che contiene la memoria allocata

Algoritmo, Programma, Processo

Processi

Algoritmo

Procedimento logico che deve essere eseguito per risolvere il problema in esame

Programma

Descrizione dell'algoritmo tramite un opportuno formalismo (linguaggio di programmazione) che rende possibile l'esecuzione dell'algoritmo da parte di un particolare elaboratore

Processo (sequenziale)

Sequenza di eventi cui dà luogo un elaboratore quando opera sotto il controllo di un particolare programma

Stato del processo

Processi

Per tenere traccia e gestire correttamente i programmi in memoria principale, il SO deve associare esplicitamente ad essi un concetto di **stato del processo**

- **In esecuzione**
- **Pronto per l'esecuzione se dispone di tutte le risorse e le condizioni logiche per eseguire ma non dispone del processore**
- **In attesa se non dispone delle risorse e delle condizioni logiche per essere eseguito**

Durante l'esecuzione, un processo cambia stato

- **Nuovo**: il processo viene creato
- **Running**: le istruzioni vengono eseguite
- **Waiting**: il processo è in attesa di qualche evento
- **Ready**: il processo è in attesa di essere assegnato ad un processore
- **Terminated**: il processo ha terminato l'esecuzione

Gestione dei processi

Processi

La possibilità che la CPU venga commutata in un qualsiasi istante da un processo ad un altro rende indispensabile ad ogni commutazione salvare tutte le informazioni contenute nei registri della CPU e relative al processo che è stato sospeso

Descrittore di processo o PCB

Area di memoria, mantenuta all'interno dell'area protetta del SO, associata al processo e contenente tutte le informazioni proprie del processo

Process Control Block (PCB)

Processi

Informazioni associate a ciascun processo

- Stato del processo
- Program counter
- Registri della CPU
- Informazioni sulla programmazione della CPU
- Informazioni sulla gestione della memoria
- Informazioni di accounting: CPU utilizzata, tempo di clock
- Informazioni sullo stato dell'I/O

Si consideri la **possibilità di avere più contatori di programma per processo**, quindi più **thread di controllo**. Deve quindi esserci una memoria per i dettagli dei thread, quindi dei contatori multipli nel PCB

Scheduling dei processi

Processi

Massimizzare l'uso della CPU, passare rapidamente i processi alla CPU per la condivisione del tempo.

Lo **scheduler dei processi** seleziona tra i processi disponibili per la successiva esecuzione sulla CPU

Mantiene le code di scheduling dei processi

- **Ready queue:** insieme di tutti i processi del SO che risiedono nella memoria principale
- **Wait queues:** insieme dei processi in attesa di un evento

Commutazione di contesto

Processi

Quando la CPU passa a un altro processo, il sistema deve salvare lo stato del vecchio processo e caricare lo stato salvato per il nuovo processo tramite un context switch.

Contesto di un processo rappresentato nel PCB.

Il tempo di commutazione del contesto è un **overhead**; il sistema non svolge alcun lavoro utile durante la commutazione.

Il tempo necessario per lo switch dipende dal supporto Hw

Multitasking nei sistemi mobili

Processi

A causa della superficie dello schermo e dei limiti dell'interfaccia utente, iOS prevede un:

- singolo processo in **foreground** primo piano, controllato dalla GUI
- processi multipli in **background**

Tipi di scheduler

Processi

- **Scheduler a breve termine (o della CPU)**: seleziona quale processo deve essere eseguito successivamente e alloca la CPU, a volte è l'unico scheduler del sistema
- **Scheduler a lungo termine (o job scheduler)**: seleziona quali processi devono essere messi in coda di attesa
- I processi possono essere descritti come:
 - processo I/O-bound: passa più tempo a fare I/O che calcoli
 - Processo CPU-bound: passa più tempo a eseguire i calcoli

Creazione del process

Processi

I processi padre (**parent**) creano processi figli (**children**) che a loro volta creano altri processi, formando un albero di processi.

In generale i processi vengono identificati e gestiti tramite un **identificatore di processi (PID)**.

Opzioni di condivisione delle risorse:

- padre e figli condividono tutte le risorse
- i figli condividono un sottoinsieme delle risorse del padre
- padre e figlio non condividono alcuna risorsa

Opzioni di esecuzione

- il genitore e i figli eseguono simultaneamente
- il genitore attende che i figli terminino

Spazio degli indirizzi:

- i figli duplicano quello del genitore
- il figlio ha un programma caricato al suo interno

Esempi UNIX

- la chiamata di sistema
`fork()`
crea un nuovo processo
- La chiamata di sistema
`exec()`
è usata dopo una
`fork()`
per sostituire lo spazio di memoria del processo con un nuovo
programma

Terminazione del processo

Processi

Il processo esegue l'ultima istruzione e poi chiede al SO di eliminarlo utilizzando la chiamata di sistema

`exit()`

Il genitore può terminare l'esecuzione dei processi figli:

- il processo figlio ha superato le risorse allocate
- il compito assegnato al figlio non è più necessario
- il processo padre sta terminando e il SO non consente a un figlio di continuare se il suo genitore termina

Alcuni sistemi operativi non consentono ai figli di esistere se il loro genitore è terminato. Se un processo termine, **anche tutti i suoi figli devono essere terminati**.

In UNIX il processo genitore può attendere la terminazione di un processo figlio utilizzando la chiamata di sistema

`wait()`

Se nessun genitore è in attesa (non ha invocato `wait()`), il processo è uno **zombie**.

Se il genitore è terminato senza invocare `wait()`, il processo è **orfano**

Architettura multiprocesso - Browser Chrome

Processi

Molti browser web venivano eseguiti come singoli processi, il **browser chrome** è multiprocesso con 3 diversi tipi di processi:

- Il processo del **browser** gestisce l'interfaccia utente
- Il processo **render** esegue il rendering delle pagine web
- Processo dei plug-in per ogni tipo di plug-in

Comunicazione tra processi

Processi

I processi all'interno di un sistema possono essere **indipendenti** o **cooperanti**.

I processi **cooperanti** necessitano di **comunicazione interprocesso (IPC)**
Due **modelli** di IPC:

- Memoria condivisa
- Passaggio di messaggi

Modelli di interazione tra processi

Processi

Modello ad ambiente globale o modello a memoria comune, modello ad ambiente locale o modello a scambio di messaggi.

Tipi di interazione tra processi:

- **Competizione**
- **Cooperazione**

Modello ad ambiente globale

Processi

Il sistema è visto come un insieme di processi e oggetti (risorse)

Il modello ad ambiente globale rappresenta la natura astrazione di un sistema multiprogrammazione costituito da uno o più processori che hanno accesso ad una memoria comune.

Ad ogni processore può essere eventualmente associata una **memoria privata**, ma ogni interazione avviene tramite oggetti contenuti nella **memoria comune**

Modello a scambio di messaggi

Processi

Il sistema è visto come un insieme di processi ciascuno operante in un ambiente locale non accessibile direttamente a nessun altro processo. Ogni forma di interazione tra processi (**comunicazione, sincronizzazione**) avviene tramite scambio di messaggi
Modello a scambi di messaggi rappresenta la naturale astrazione di un sistema privo di memoria comune, in cui a ciascun processore è associata una memoria privata

Processi cooperanti

Processi

Un processo indipendente non può influenzare o essere influenzato dall'esecuzione di un altro processo

Processi cooperanti possono influenzare o essere influenzati dall'esecuzione di un altro processo.

Vantaggi della cooperazione tra processi

- Condivisione delle informazioni
- Velocità di calcolo
- Modularità
- Convenienza

Problema produttore-consumatore

Processi

Paradigma dei processi cooperanti, il processo produttore produce informazioni che vengono consumate da un processo consumatore

- **Unbounded-buffer** non pone limiti pratici alla dimensione del buffer
- **Bounded-buffer** presuppone l'esistenza di una dimensione fissa del buffer

Comunicazione tra processi (IPC) - Memoria condivisa

Processi

Un'area di memoria condivisa tra i processi che desiderano comunicare. La comunicazione è sotto il controllo dei processi degli utenti e non del SO. Il problema principale è fornire un meccanismo che permetta ai processi utente di sincronizzare le loro azioni quando accedono alla memoria condivisa.

Comunicazione tra processi - Scambio di messaggi

Processi

Meccanismo che consente ai processi di comunicare e sincronizzare le loro azioni

Sistema di messaggi - i processi comunicano tra loro senza ricorrere a variabili condivise.

La struttura IPC fornisce due operazioni:

- **send(messaggio)**
- **receive(messaggio)**

Implementazione del collegamento di comunicazione

- **Fisico**

- Memoria condivisa
- Bus hardware
- rete

- **Logico**

- Diretto o indiretto
- Sincrono o asincorno
- Buffering automatico o esplicito

Costrutti linguistici per il modello a scambio di messaggi

Processi

Classificazione

- **Designazione** dei processi sorgente e destinatario di ogni comunicazione
 - designazione diretta o esplicita
 - simmetrica
 - asimmetrica
 - designazione indiretta o globale
 - mailbox
 - porte
- **Tipo di sincronizzazione** tra i processi comunicanti
 - sincrona
 - asincrona

Primitive con designazione esplicita

Processi

send <expression_list> **to** <destination_designator>

receive <variable_list> **from** <source_designator>

- L'esecuzione della *send* determina il contenuto del messaggio mediante la valutazione delle espressioni in <expression_list>.
- <destination_designator> dà al programmatore il controllo su dove inviare il messaggio.
- L'esecuzione della *receive* determina l'assegnamento dei valori contenuti nel messaggio alle variabili in <variable_list>, e la successiva distruzione del messaggio.
- <source_designator> dà al programmatore il controllo sull'origine dei messaggi.

Designazione esplicita

Processi

- La coppia (*<destination_designator>*, *<source_designator>*) definisce un *canale di comunicazione*.

- **Schema simmetrico**

I processi si nominano *esplicitamente* (*direct naming*) l'un l'altro:

send message to P2
receive message from P1

- P1 invia un messaggio che può essere ricevuto solo da P2
- P2 riceve un messaggio che può provenire solo da P1
- Semplice da realizzare e da utilizzare: un processo può controllare in maniera selettiva gli intervalli di tempo in cui riceve messaggi dagli altri processi.
- E' usato nei modelli del tipo *pipeline*: collezione di processi concorrenti in cui l'output di un processo costituisce l'input di un altro. Il sistema è concepito in termini di flusso di informazione.

Direct naming: schema asimmetrico

Processi

Il mittente nomina **esplicitamente** il destinatario, mentre questi, al contrario, non esprime il nome del processo con cui desidera comunicare. La **designazione asimmetrica** facilita l'organizzazione dell'interazione tra processi secondo il paradigma **client-server**, in cui un processo gestore di una risorsa (server) riceve richieste da più processi client.

Modello client-server

Processi

Corrisponde all'uso di un **processo come gestore di una risorsa**

Schema da-molti-a-uno: i processi client inviano richieste non ad un particolare server, ma ad uno qualunque scelto tra un insieme di **server equivalenti**.

È difficile realizzazione con designazione diretta. Richiede di passare ad una **designazione indiretta o globale (mailbox)**

Modello client-server e naming

Processi

Il **direct naming** è in generale poco adatto al modello client-server.
In presenza di più **client** la *receive* di un server dovrebbe consentire la ricezione di un messaggio da un qualsiasi client.
In presenza di più **server equivalenti** la *send* di un clinet dovrebbe produrre un messaggio che possa essere ricevuto da un qualsiasi server.
Occorre uno schema più sofisticato per la definizione dei canali di comunicazione: **designazione globale o indiretta** che fa uso di nomi globali detti **mailbox**

Uso delle mailbox

Processi

La **mailbox** consente in modo immediato la **programmazione delle interazioni client-server** anche nel caso da-molti-a-molti.

I client eseguono una *send* sulla mailbox associata al servizio, i server una *receive*.

La realizzazione delle mailbox in ambiente distribuito presenta problemi di **natura realizzativa**. Il supporto a tempo di esecuzione del linguaggio deve garantire che:

- un messaggio di richiesta indirizzato ad una mailbox è inviato a tutti i processi che possono eseguire una *receive* su di essa
- non appena il messaggio è ricevuto da un processo, esso deve diventare indisponibile per tutti gli altri server

Sono mailbox il cui nome può comparire solamente in un processo come *<source-designator>* in uno statement di *receive*.

Un processo può selezionare i messaggi che desidera ricevere attraverso l'uso di **porte distinte**

Designazione (naming)

Processi

- **Direct naming simmetrico:** comunicazione *one to one*
- **Direct naming asimmetrico e port naming:** comunicazione *many to one*
- **Global naming:** comunicazione *many to many*

Naming statico:

- Impedisce ad un programma di comunicare tramite canali non noti a tempo di compilazione
- il potenziale accesso di un programma ad un canale deve essere assicurato fin dall'inizio

Naming dinamico

- uno schema statico di base di designazione dei canali viene arricchito mediante variabili per la designazione di sorgente o destinazione

Sincronizzazione - send asincrona

Processi

Il processo mittente **continua la sua esecuzione** immediatamente dopo che il messaggio è stato inviato.

Il messaggio ricevuto contiene informazioni che non possono essere associate allo stato attuale del mittente

L'interazione viene definita come **scambio di messaggi asincrono**.

In analogia con il meccanismo semaforico, la **send asincrona** è caratterizzata da:

- **flessibilità d'uso**
- **carenza espressiva**

Richiede, a livello realizzativo, un buffer di capacità limitata. Si può ovviare **modificandone la semantica**

1. un processo mittente si blocca qualora la coda dei messaggi sia piena
2. la primitiva send, in caso di coda piena, solleva un'**eccezione** che viene notificata al processo mittente

Sincronizzazione - send sincrona

Processi

Il processo mittente si blocca in attesa che il messaggio sia stato ricevuto.
Un messaggio ricevuto contiene informazioni corrispondenti allo **stato attuale del processo mittente**.

L'invio di un messaggio costituisce un **punto di sincronizzazione** sia per il mittente che per il destinatario.

L'interazione viene definita come **scambio di messaggi sincrono**

Send di tipo "chiamata di procedura remota"

Processi

"Rendez-vous" esteso: il processo mittente **rimane in attesa** fino a che il ricevimento non ha terminato di svolgere l'azione richiesta.

Un processo client "chiama" una procedura eseguita da un processo server su una macchina potenzialmente remota.

Il nome della procedura remota identifica un **processo in caso di direct naming**.

I programmi risultano più facilmente verificabili grazie alla localizzazione dei vincoli di sincronizzazione

Receive

Processi

Normalmente è bloccante se non vi sono messaggi sul canale. Costituisce un **punto di sincronizzazione** per il processo ricevente.

Si ricorre ad una primitiva che verifica lo stato del canale e restituisce un messaggio se esso è presente, ovvero un'indicazione di canale vuoto (**receive non bloccante**).

Inconveniente: per l'attesa di messaggi da specifici canali occore fare uso di cicli di attesa attiva

Chiamata di procedura remota (RPC)

Processi

Consente di esprimere a più alto livello e in maniera più sintetica le interazioni di tipo client-server.

Service è il nome di un canale:

- se la designazione è diretta, service indica il processo servitore
- se la designazione è indiretta (porte o mailbox), service indica il tipo di servizio richiesto

Specifica lato server:

1. come procedura dichiarate separatamente
2. come statement collocato in un punto qualunque di un processo

RPC: specifica lato server

Processi

La procedura remota viene dichiarata come una procedura in un linguaggio sequenziale e realizzata come un processo server che attende la ricezione di un messaggio, esegue il corpo e trasmette un messaggio di risposta.

Può essere realizzata come un **singolo processo** che esegue le richieste una alla volta in modo sequenziale, oppure con la creazione di un **nuovo processo per ogni chiamata**. Le varie istanze sono eseguite concorrentemente, e potranno eventualmente doversi sincronizzare tra loro. La RPC è uno **statement**, e come tale può essere collocato in un punto qualunque del processo server.

L'esecuzione delle **accept** sospende il servitore fino all'arrivo di un messaggio corrispondente alla **call** di servizio.

L'esecuzione del corpo può fare uso dei valori dei parametri e di tutte le variabili accessibili dallo scope dello statement.

Al termine viene trasmesso il messaggio di risposta al processo chiamante, dopo di che il processo server **continua la propria esecuzione**.

Uso delle accept

Processi

Quando il lato server è specificato con una **accept**, la RPC viene chiamata *extended rendez-vous*: client e server si "incontrano" per la durata dell'esecuzione del corpo della accept per poi proseguire separatamente.

Vantaggi:

- il server può fornire più tipi di servizi
- il server può decidere quando servire le call dei client
- le istruzioni di accept possono essere alternate o innestate
- vi possono essere più accept di chiamate allo stesso servizio con diverso body

Uso delle RPC

Processi

Lo schema di comunicazione realizzato dal meccanismo della chiamata a procedura remota è di tipo asimmetrico e da molto ad uno.

L'accoppiamento tra una chiamata priva di parametri ed una accept priva di corpo rappresenta la trasmissione ed il relativo riconoscimento di un segnale di sincronizzazione.

Problemi della RPC:

- interazioni non client-server
- perdita di messaggi nelle architetture distribuite

Esempi di sistemi IPC - Mach

Processi

La comunicazione di Mach è basata su messaggi

- Anche le chiamate di sistema sono messaggi
- Ogni task riceve due mailbox alla creazione: Kernel e Notify
- Le mailbox necessarie per la comunicazione sono create tramite `port_allocate()`
- L'invio e la ricezione sono flessibili, ad esempio quattro opzioni se la casella di posta è piena:
 - Attendere indefinitamente
 - Attendere al massimo n millisecondi
 - Restituire immediatamente
 - Mettere temporaneamente in cache un messaggio

Esempi di sistemi IPC - Windows

Processi

Centralità del **message passing** tramite una struttura avanzata di chiamata di procedura locale (LPC).

Funziona solo tra processi sullo stesso sistema, utilizza porte per stabilire e mantenere i canali di comunicazione.

La comunicazione funziona come segue:

- Il client apre un handle dell'oggetto porta di connessione del sottosistema
- Il cliente invia una richiesta di connessione
- Il server crea due porte di comunicazione private e restituisce al cliente l'handle di una di esse
- Il client e il server utilizzano l'handle della porta corrispondente per inviare messaggi o callback e per ascoltare le risposte

Comunicazioni nei sistemi client-server

Processi

- Socket
- Chiamate di procedura remota
- Pipe
- Invocazione di metodi remoti (Java)

Socket

Una **socket** è definita come un **endpoint** per la comunicazione.

Concatenazione di indirizzo IP e porta - numero incluso all'inizio del pacchetto di messaggi per differenziare i servizi di rete su un host.

La comunicazione consiste in una coppia di socket.

Tutte le porte inferiori a 1024 sono ben conosciute, utilizzate per servizi standard.

Indirizzo IP speciale 127.0.0.1 (loopback) per riferirsi al sistema su cui il processo è in esecuzione

Chiamate di procedura remota

Processi

RPC

La RPC astrae le chiamate di procedura tra processi su sistemi di rete

Stub - proxy lato client per la procedura effettiva sul server.

Lo stub sul lato client individua il server e fa il marshalling dei parametri.

Lo stub lato server riceve questo messaggio, scompatta i parametri ed esegue la procedura sul server

Rappresentazione dei dati gestita tramite il formato XDL per tenere conto delle diverse architetture (**Big endian, little endian**).

La comunicazione remota ha più scenari di fallimento rispetto a quella locale.

Il sistema operativo fornisce tipicamente un servizio di rendez-vous (o mathmaker) per connettere client e server

Agisce come un condotto che consente a due processi di comunicare
Le pipe ordinarie non sono accessibili dall'esterno del processo che le ha create. In generale, un processo padre crea una pipe e la usa per comunicare con un processo figlio che ha creato.
FIFO: è possibile accedervi senza una relazione genitore-figlio.

Le pipe ordinarie consentono la comunicazione in stile produttore-consumatore standard.

Il produttore scrive su un'estremità e il consumatore legge dall'altra parte.
Le pipe ordinarie sono quindi unidirezionali, richiedono una relazione genitore-figlio tra i processi comunicati.

Pipe con nome: FIFO

Processi

Le FIFO sono più potenti delle pipe ordinarie, la comunicazione è bidirezionale.

Non è necessaria alcuna relazione genitore-figlio tra i processi comunicanti. Diversi processi possono utilizzare le pipe con nome per comunicare

Sommario

1. Introduzione

2. Evoluzione dei sistemi operativi

3. Appofondimento delle interruzioni

4. Servizi e organizzazione del SO

5. Virtualizzazione

6. Processi

7. Thread e concorrenza

8. Scheduling

9. Sincronizzazione dei processi

10. Sincronizzazione dei processi in ambiente globale mediante semafori

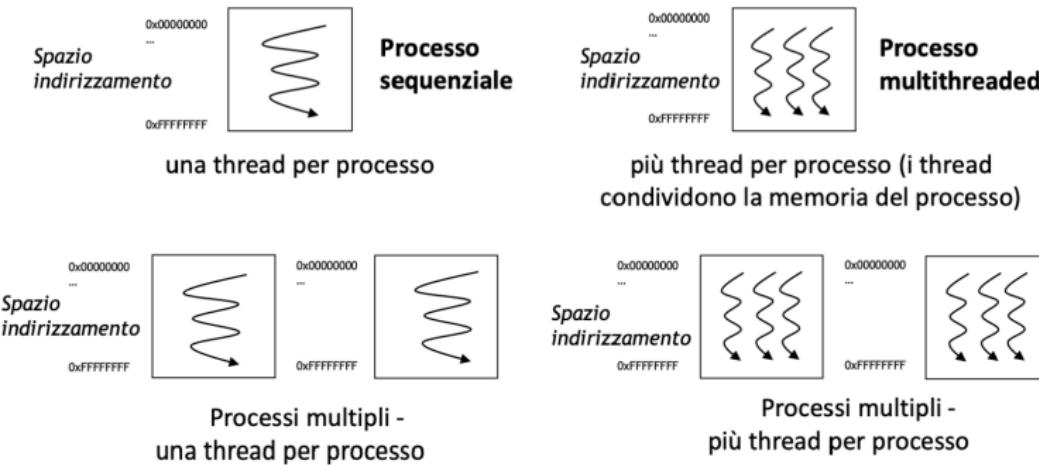
11. Deadlock

12. Sistema di protezione

13. Gestione della memoria principale e virtuale

Processi e thread

Thread e concorrenza



Processo (ad es. UNIX) \Rightarrow processo pesante (creato con `fork` - identico al padre)
Thread \Rightarrow processo leggero (creato con `pthread_create` - esegue la funzione indicata come argomento)
ogni thread ha un suo contesto di esecuzione (registri, PC, stack)
ma condivide codice, dati e file del processo

Vantaggi del multithreading

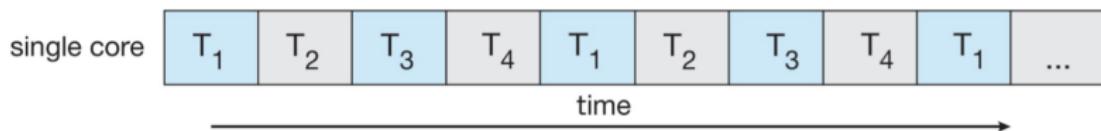
Thread e concorrenza

- **Reattività:** può consentire di continuare l'esecuzione se una parte del processo è bloccata, particolarmente importante per le interfacce utente
- **Condivisione delle risorse:** i thread condividono le risorse del processo, più facilmente della memoria condivisa o del passaggio di messaggi
- **Economicità:** meno costoso della creazione di un processo, la commutazione di thread ha un overhead rispetto alla commutazione di contesto
- **Scalabilità:** il processo può trarre vantaggio dalle architetture multicore

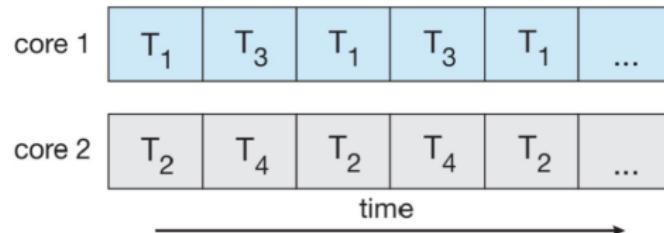
Concorrenza vs Parallelismo

Thread e concorrenza

- Esecuzione concorrente su un sistema a core singolo:



- Parallelismo su un sistema multi-core:

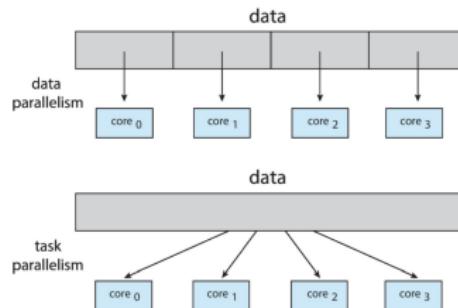


Programmazione multicore

Thread e concorrenza

Tipi di parallelismo:

- **Parallelismo dei dati:** distribuisce sottoinsiemi degli stessi dati su più core, la stessa operazione su ciascuno di essi
- **Parallelismo dei task:** distribuzione dei thread tra i core, ogni thread esegue un'unica operazione



Legge di Amdahl

Thread e concorrenza

Identifica i guardagni di prestazioni derivanti dall'aggiunta di core supplementari a un'applicazione che presenta componenti sia seriali che paralleli

Thread utente e thread kernel

Thread e concorrenza

Thread utente - gestione effettuata dalla libreria dei thread a livello utente (POSIX Pthread, Thread Windows, Thread Java).

Thread del kernel - supportati dal kernel

Modelli di multithreading

Thread e concorrenza

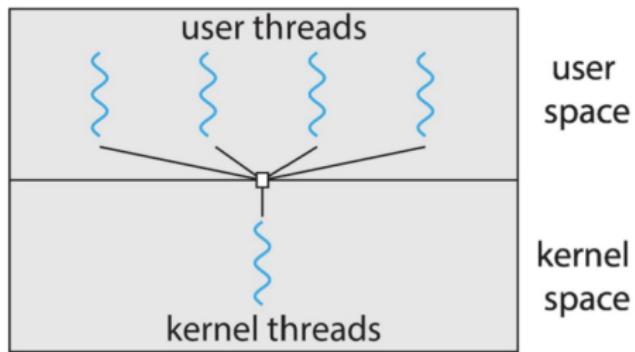
- Da uno a molti
- Da uno a uno
- Da molti a molti

Da molti a uno

Molti thread a livello utente mappati su un singolo thread del kernel
Il blocco di un thread causa il blocco di tutti.

Più thread non possono essere eseguiti in parallelo su un sistema multicore
ma **solo uno alla volta**.

Pochi sistemi attualmente utilizzano questo modello



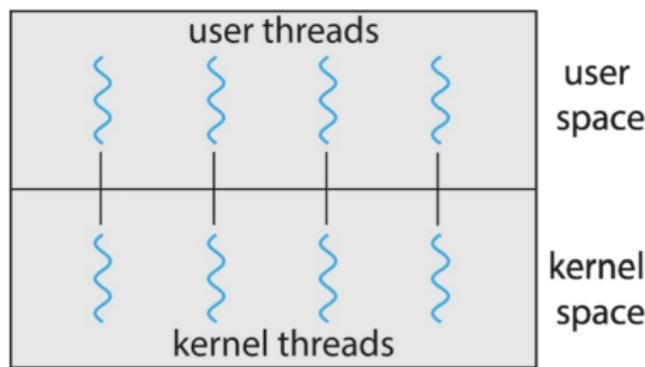
Uno a uno

Ogni thread a livello utente è mappato in un **thread kernel**.

La creazione di un thread a livello utente crea un thread del kernel.

Più **concorrenza rispetto a multi-a-uno**

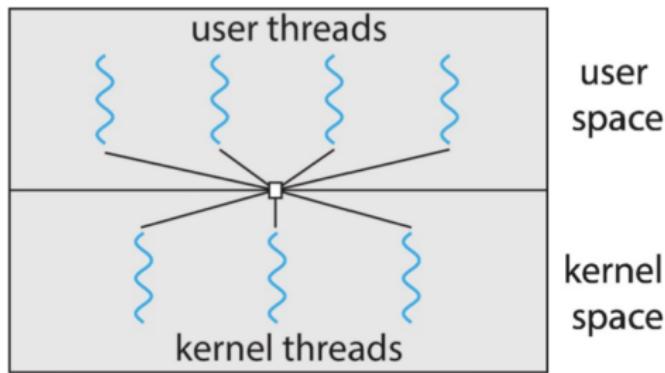
Numero di thread per processo a volte limitato a causa dell'overhead



Molti a molti

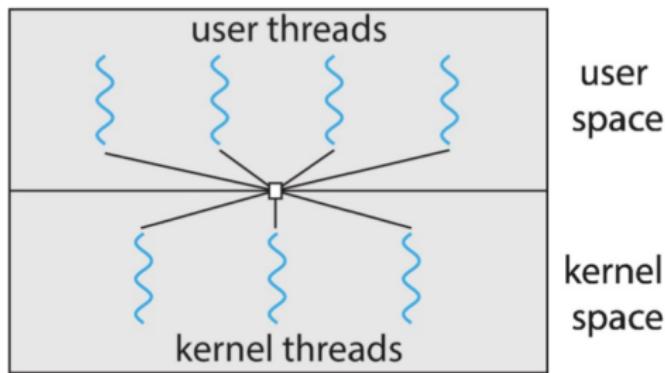
Permette a un thread a livello utente di essere mappato su molti thread del kernel.

Permette al sistema operativo di creare un numero sufficiente di thread del kernel



Modello a due livelli

Simile a multi-a-multi, tranne per il fatto che permette a un thread utente di essere legato a un thread kernel



Librerie di thread

Thread e concorrenza

Le librerie di thread forniscono al programmatore le API per la creazione e la gestione dei thread.

Due modi principali di implementare

- Libreria interamente in spazio utente
- Libreria a livello di kernel supportata dal sistema operativo

Pthreads

Thread e concorrenza

Può essere fornita a livello utente o a livello kernel

API a standard POSIX per la creazione e la sincronizzazione dei thread.

L'API specifica il comportamento della libreria di thread, l'implementazione spetta allo sviluppo della libreria stessa

Thread Java

Thread e concorrenza

I thread Java **sono gestiti dalla JVM**.

Tipicamente implementati utilizzando il modello di thread fornito dal sistema operativo sottostante.

Thread implicite

Thread e concorrenza

Sempre più diffusa per il numero crescente di thread, la verifica della correttezza del programma è sempre più difficile.

Creazione e gestione dei thread da parte dei compilatori e delle librerie runtime piuttosto che dei programmatori.

Cinque metodi esplorati:

- Pool di thread
- Fork-Join
- OpenMP
- Spedizione Grand Centrale
- Blocchi costruttivi della filettatura Intel

Pool di thread

Thread e concorrenza

Vengono creati un numero di thread in un pool in cui attendono di lavorare.

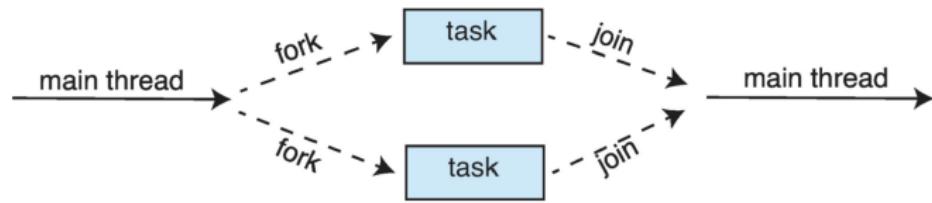
Vantaggi:

- Di solito è leggermente più veloce service una richiesta con un thread esistente piuttosto che crearne uno nuovo
- Consente di vincolare il numero di thread dell'applicazione alla dimensione del pool
- Separare l'attività da eseguire dalla meccanica di creazione dell'attività consente diverse strategie per l'esecuzione dell'attività

Parallelismo Fork-Join

Thread e concorrenza

Più thread vengono biforcati e poi uniti



OpenMP

Thread e concorrenza

Insieme di direttive del compilatore e API per C, C++

Fornisce supporto per la programmazione parallela in ambienti a memoria condivisa

Identifica le regioni parallele-blocchi di codice che possono essere eseguiti in parallelo

Grand Central Dispatch

Thread e concorrenza

- Tecnologia Apple per i sistemi operativi macOS e iOS
- Estensioni dei linguaggi C, C++, API e libreria di esecuzione che incapsulano codice e dati
- Permette di identificare le sezioni parallele
- Gestisce la maggior parte dei dettagli del threading

Due tipi di code di invio:

- **Seriale**: blocchi rimossi in ordine FIFO, la coda è per processo, chiamata **coda principale**
- **Concorrente**: rimozione in ordine FIFO, ma possono esserne rimossi diversi alla volta

Per il linguaggio Swift, un task è definito come una closure, simile a un blocco, senza il trattino

Intel Threading Building Block (TBB)

Thread e concorrenza

Libreria di modelli per la progettazione di programmi paralleli in C++

Problemi di threading

Thread e concorrenza

Semantica delle chiamate di sistemi *fork()* ed *exec()*.

Signal handling: *sincrono* e *asincrono*

Cancellazione del thread di destinazione: asincrona o differita

Memoria locale del thread

Attivazione dello scheduler

Gestione dei segnali

Thread e concorrenza

- I **segnali** sono utilizzati nei sistemi UNIX per notificare a un processo che si è verificato un particolare evento
- Un **gestore di segnali** viene utilizzato per elaborare i segnali
 - Il segnale è generato da un particolare evento
 - Il segnale è consegnato ad un processo
 - Il segnale viene gestito da uno dei due possibili gestori del segnale
- Ogni segnale ha un gestore di default che il kernel esegue quando gestisce un segnale

Cancellazione di un thread

Thread e concorrenza

- Terminare un thread prima che sia terminato
- Il thread da cancellare è il **thread target**
- Due approcci generali:
 - **Cancellazione asincrona** termina immediatamente il thread target
 - **Cancellazione differita** consente al thread target di controllare periodicamente se deve essere cancellata

L'invocazione della cancellazione del thread richiede la cancellazione, ma la cancellazione effettiva dipende dallo stato del thread

Il tipo predefinito è differito: la cancellazione avviene solo quando il thread raggiunge il **punto di cancellazione**

Cancellazione di un thread in Java

Thread e concorrenza

La cancellazione differita utilizza il metodo `interrupt()`, che imposta lo stato di interruzione di un thread

Memorizzazione locale del thread

Thread e concorrenza

Il **Thread-Local Storage (TLS)** consente a ciascun thread di avere la propria copia di dati.

Utile quando non si ha il controllo sul processo di creazione del thread.
Diverso dalle variabili locali, le variabili locali sono visibili solo durante l'invocazione di una singola funzione, il TLS è visibile tra le invocazioni di funzioni.

È simile ai dati statici, è unico per ogni thread.

Attivazione dello scheduler

Thread e concorrenza

Entrambi i modelli M:M e a due livelli richiedono una comunicazione per mantenere il numero appropriato di thread del kernel allocate all'applicazione.

Tipicamente usano una struttura dati intermedia tra i thread user e quelli kernel - **lightweight process (LWP)**.

Le attivazioni dello scheduler forniscono **upcall** - un meccanismo di comunicazione dal kernel al gestore di upcall nella libreria dei thread

Thread di Windows

Thread e concorrenza

Windows API - API principale per le applicazioni windows, implementa la mappatura uno-a-uno, a livello di kernel.

Ogni thread contiene

- un id di thread
- un set di registri che rappresentano lo stato del processore
- stack utente e stack kernel separati per quando il thread viene eseguito in modalità utente o in modalità kernel
- area di memorizzazione dati privata utilizzata dalle librerie run-time

Il set di registri, gli stack e l'area di archiviazione privata sono noti come **contesto del thread**

Le strutture dati primarie di un thread includono:

- **ETHREAD**: include il puntatore al processo a cui appartiene il thread
- **KTHREAD**: informazioni sullo scheduling e sulla sincronizzazione, stack in modalità kernel
- **TEB (blocco ambiente thread)**: id del thread, stack in modalità utente, memoria locale dei thread

Thread di Linux

Thread e concorrenza

Linux le chiama **task**.

La creazione di un thread avviene tramite la chiamata sistema *clone()* che permette ad un task figlio di condividere lo spazio degli indirizzi del task genitore (processo).

Sommario

1. Introduzione

2. Evoluzione dei sistemi operativi

3. Appofondimento delle interruzioni

4. Servizi e organizzazione del SO

5. Virtualizzazione

6. Processi

7. Thread e concorrenza

8. Scheduling

9. Sincronizzazione dei processi

10. Sincronizzazione dei processi in ambiente globale mediante semafori

11. Deadlock

12. Sistema di protezione

13. Gestione della memoria principale e virtuale

Introduzione

Scheduling

La gestione delle risorse impone al SO di prendere **decisioni sulla loro assegnazione** in base a criteri di efficienza e funzionalità.

Le risorse più importanti, a questo riguardo, sono la **CPU** e la **memoria principale**.

Scheduler (della CPU)

Parte del SO che **decide** a quale dei processi pronti presenti nel sistema assegnare il controllo della CPU

Algoritmo di scheduling

Realizza un particolare criterio di scelta tra i processi pronti (**politica**)

Scheduler della CPU

Scheduling

Long-term scheduler (o job scheduler)

Determina quali processi dalla memoria di massa devono essere caricati in memoria principali pronti per l'esecuzione.

Controlla il grado di multiprogrammazione

Criterio di selezione è basato su un mix equilibrato di job I/O bound e CPU bound

Short-term scheduler

Seleziona tra tutti i processi in memoria pronti per l'esecuzione quello cui assegnare la CPU.

Deve essere efficiente in quanto interviene **frequentemente**

Nei sistemi **time-sharing** non esiste il long-term scheduling.
I processi entrano immediatamente in memoria centrale, il limite è imposto o dal numero dei terminali connessi o dal tempo di risposta che diviene troppo lungo.

Medium-term scheduler

È coinvolto soprattutto nei sistemi operativi **multitasking** e **multiprogrammazione**, dove più processi concorrono per l'accesso alla CPU.

Il medium-term scheduler può decidere di sospendere temporaneamente l'esecuzione di alcuni processi, spostandoli dalla memoria principale a uno spazio di scambio su disco (**swap space**)

Scheduling e revoca della CPU

Scheduling

La riassegnazione della CPU può avvenire a seguito di uno dei seguenti eventi

1. Un processo commuta dallo **stato di esecuzione a sospeso**
2. Un processo commuta dallo **stato di esecuzione a pronto**
3. Il processo in esecuzione termina
4. Un processo **commuta dallo stato sospeso a pronto**

Scheduling non-preemptive

- Un processo in esecuzione prosegue fino al rilascio spontaneo della CPU
- La riassegnazione della CPU avviene solo a seguito di eventi di tipo 1 e 3, o di tipo 2 nel caso di *yield*

Scheduling preemptive

- Il processo in esecuzione può perdere il controllo della CPU anche se "logicamente" in grado di proseguire
- La riassegnazione della CPU può avvenire anche a seguito di eventi di tipo 2 e 4



Criteri dello scheduling

Scheduling

- **Utilizzo della CPU** - mantiene la CPU il più possibile occupata
- **Throughput** - numero di processi che completano la loro esecuzione per unità di tempo
- **Turnaround time** - tempo di esecuzione di un particolare processo
- **Tempo di attesa** - quantità di tempo in cui un processo è rimasto in attesa nella coda di attesa
- **Tempo di risposta** - quantità di tempo che intercorre tra l'invio di una richiesta e la produzione della prima risposta, non l'output
- **Fairness** - assenza di privilegi

Criteri di ottimizzazione degli algoritmi di pianificazione

Scheduling

- Utilizzo massimo della CPU
- Throughput massimo
- Tempo minimo di esecuzione
- Tempo di attesa minimo
- Tempo di risposta minimo - variazna del tempo di risposta minima

First come, first server (FCFS)

Scheduling

La CPU viene assegnata al processo che l'ha richiesta per primo.
La realizzazione di questa politica è ottenuta con code gestite in modo FIFO.
Le prestazioni di questo algoritmo sono in **genere scadetni**, in termini di tempo medio di attesa.
È semplice da realizzare

Shortest Job First (SJF)

Scheduling

A ciascun processo è associata la **lunghezza del successivo burst di CPU**, quando la CPU è libera, essa viene assegnata al processo con il burst di CPU più breve.

Fornisce la soluzione ottima per il criterio del tempo medio di attesa.

Priorità

Scheduling

Il SO mantiene i processi pronti in **code separate** per i diversi livelli di priorità

Lo scheduler seleziona sempre il primo processo nella coda al livello massimo di priorità che contiene processi pronti.

In presenza di **preemption**, in ogni istante è in esecuzione **un processo a priorità massima**.

Valutazione della priorità

- **Valutata internamente:** la priorità è individuata sulla base di qualche quantità misurabile (es. limiti di tempo, richieste di memoria, ecc.)
- **Valutata esternamente:** è impostata da condizioni esterne

Problema di starvation

Processi a bassa priorità possono rimanere indefinitamente ritardati.

Una soluzione è quella di **aumentare gradualmente la priorità dei job che attendono**

Priorità statica e priorità dinamica:

- **Priorità statica:** attribuita ai processi all'atto della creazione in base alle loro caratteristiche o a politiche riferite dal tipo di utente
- **Priorità dinamica:** modificata durante l'esecuzione del processo
 - per penalizzare processi che impegnano troppo la CPU
 - per evitare starvation
 - per favorire i processi che si dimostrano I/O-bound
 - per mantenere un buon job-mix

Round Robin

Scheduling

Politica caratteristica dei sistemi time-sharing: assegna un quanto di tempo prefissato ad ogni processo.

La coda dei processi pronti è **circolare** e la CPU è **assegnata a ciascuno dei processi per un quanto di tempo**.

Un processo, se introdotto per l'esaurimento del suo quanto viene inserito come **ultimo nella coda dei processi pronti**.

Elevata fairness e assenza di starvation

Varianti preemptive degli algoritmi di scheduling

Scheduling

- Gli algoritmi **FCFS**, **SJF** e **a priorità** sono di tipo **non-preemptive**, cioè quando la CPU è stata assegnata ad un processo questi ne mantiene il controllo fino al proprio completamento, o alla richiesta di una operazione di I/O.
- Gli algoritmi **SJF** e **a priorità** possono anche essere di tipo **preemptive**. Tale possibilità nasce quando, durante l'esecuzione di un processo, un nuovo processo entra nella coda dei processi pronti. Il nuovo processo può **richiedere un tempo di CPU inferiore e avere una priorità maggiore di quello in esecuzione**
- L'algoritmo SJF di tipo **preemptive** viene chiamato anche **shortest remaining time first (SRTF)**

Coda multilivello

Scheduling

Con lo scheduling a priorità vi sono code separate per ogni priorità.
Viene eseguito il processo a priorità più alta

Coda di feedback multivello

Scheduling

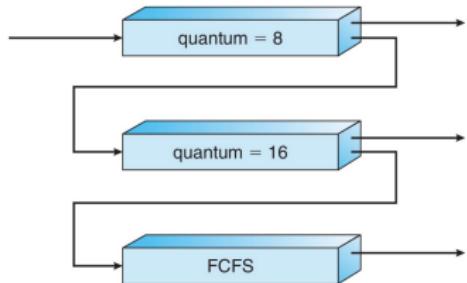
Un processo può spostarsi tra le varie code.

Uno scheduler di code a feedback multivello definito dai seguenti parametri

- numero di code
- metodo utilizzato per determinare quando aggiornare un processo
- metodo utilizzato per determinare quando retrocedere un processo
- metodo utilizzato per determinare in quale coda entra un processo quando ha bisogno di assistenza

Esempio di coda multivello

- Tre code:
 - Q0 - RR con quanto temporale di 8 millisecondi
 - Q1 - RR con quanto di tempo di 16 millisecondi
 - Q2 - FCFS
- Scheduling
 - Un nuovo lavoro entra nella coda Q0 che viene servita FCFS
 - . Quando ottiene la CPU, il job riceve al più 8 millisecondi di esecuzione
 - . Se non termina entro 8 millisecondi, il job viene spostato nella coda Q1
 - In Q1 il job viene nuovamente servito in FCFS e riceve 16 millisecondi aggiuntivi
 - . Se ancora non viene completato, subisce la preemption e spostato nella coda Q2



Scheduling dei thread

Scheduling

Distinzione tra **thread a livello utente e a livello kernel**.

Quando sono supportate, lo scheduling riguardo i thread, non i processi.

Modelli **molti-a-uno** e **molti-a-molti**, la libreria di thread pianifica i thread a livello utente per l'esecuzione su LWP. Conosciuto come **process-contention scope (PCS)**, poiché la competizione di scheduling è all'interno del processo.

Lo scheduling di un kernel thread su una CPU disponibile è un **system-contention scope (SCS)** - competizione tra tutti i thread del sistema.

Scheling a più processori

Scheduling

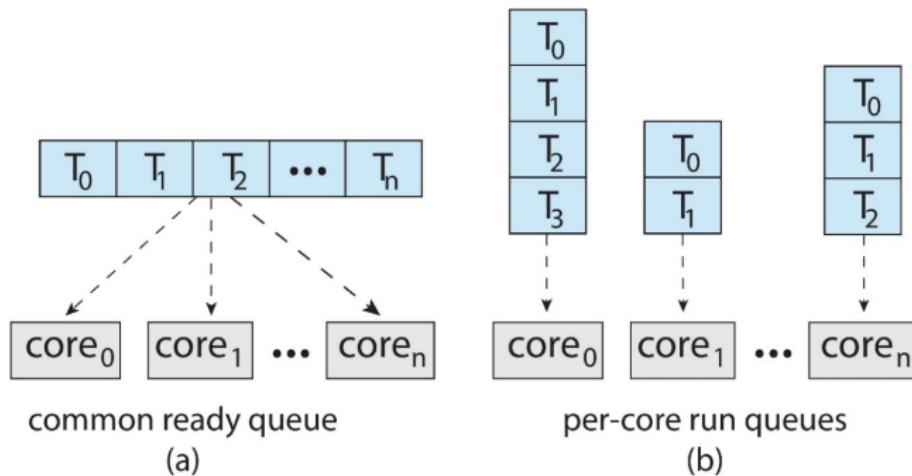
Lo scheduling della CPU è più complesso quando sono disponibili più CPU
Una qualsiasi delle seguenti architetture può essere multiprocessore

- CPU multicore
- Core multithread
- Sistemi NUMA
- Multiprocessing eterogeneo

Il multiprocessing simmetrico (SMP) è quello in cui ogni processore è auto-schedulante.

Tutti i thread possono trovarsi in una **ready queue** comune (a).

Ogni processore può avere la propria ready queue privata di thread (b)



Processori multicore

Scheduling

Tendenza recente a collocare più core di processore sullo stesso chip fisico.
Più veloce e consuma meno energia.
Crescono anche i thread multipli per core

Sistema multicore multithreading

Scheduling

Chip-multithreading (CMT) assegna a ciascun core più thread hardware.
(Intel lo chiama **hyperthreading**).

Su un sistema quad-core con 2 thread hardware per core, il SO vede 8 processori logici.

Abbiamo due livelli di **schedulazione**

1. Il SO decide quale thread software eseguire su una CPU logica
2. Ogni core decide quale thread hardware eseguire sul core fisico

Scheduling a più processori - Bilanciamento del carico

Scheduling

Se il sistema è SMP, è necessario che tutte le CPU siano caricate per garantire l'efficienza.

Il **load balancing** cerca di mantenere il carico di lavoro uniformemente distribuito.

Migrazione push - un task periodico controlla il carico su ciascun processore e, se lo trova, spinge il task dalla CPU sovraccarica ad un'altra CPU.

Migrazione pull - i processori inattivi prelevano i task in attesa dai processori occupati

Scheduling a più processori - Affinità dei processori

Scheduling

Quando un thread è in esecuzione su un processore, il contenuto della cache di quel processore memorizza gli accessi alla memoria effettuata da quel thread.

Si parla di thread che **hanno affinità con un processore**.

Soft affinity - il SO tenta di mantenere un thread in esecuzione sullo stesso processore, ma senza garanzie

Hard affinity - consente a un processo di specificare un insieme di processori su cui può essere eseguito

NUMA e lo scheduling delle CPU

Scheduling

Se il SO è **NUMA-aware**, assegnerà memoria vicina alla CPU su cui il thread è in esecuzione.

Scheduling della CPU in tempo reale

Scheduling

Sistemi soft real-time: i task critici in tempo reale hanno la massima priorità, ma non c'è garanzia su quando i task saranno posti in esecuzione

Sistemi hard real-time: i task **devono essere serviti entro la loro scadenza.**

Nei sistemi **real-time** organizzati a processi tipicamente sono presenti sia **processi critici**, che devono soddisfare i vincoli temporali, che **processi non critici**, eseguiti con algoritmi di scheduling convenzionali negli intervalli di tempo residui

Latenza degli eventi

La quantità di tempo che intercorre tra il momento in cui si verifica un evento e il momento in cui viene servito

Due tipi di latenza influenzano le prestazioni

- **Latenza dell'interrupt:** tempo che intercorre tra l'arrivo dell'interrupt e l'inizio della routine che lo serve
- **Latenza di dispatch:** tempo necessario al dispatcher per togliere il processo corrente dalla CPU e passare ad un altro

Schedulazione basata sulla priorità

Scheduling

Per una schedulazione in tempo reale, lo scheduler deve supportare una schedulazione **preemptive** e basata su priorità.

Per l'**hard real-time** deve anche fornire la capacità di rispettare le scadenze.

I processi hanno nuove caratteristiche: quelli periodici richiedono la CPU a intervalli costanti

Rate Monotonic Scheduling

Scheduling

Una priorità viene assegnata in base all'inverso del suo periodo

Schedulazione della scadenza più vicina (EDF)

Scheduling

Le priorità sono assegnate in base alle scadenze: quanto più è precoce la scadenza, tanto più alta è la priorità

Schedulazione a quote proporzionali

Scheduling

Le quote T sono distribuite tra tutti i processi del sistema.
Un'applicazione riceve N condivisioni dove $N < T$

Scheduling in tempo reale POSIX

Scheduling

Lo standard POSIX.1b API fornisce **funzioni per la gestione dei thread in tempo reale**.

Definisce due classi di schedulazione per i thread in tempo reale:

- *SCHED_FIFO*: i thread sono programmati utilizzando una strategia FCFS con una coda FIFO. Non c'è time-slicing per i thread di uguale priorità
- *SCHED_RR*: simile a *SCHED_FIFO*, tranne per il fatto che il time-slicing (RR) avviene per i thread di pari priorità

Esempi di sistema operativo

Scheduling

- Schedulazione tradizionale UNIX
- Scheduling di Linux
- Scheduling Windows

Esempi di sistema operativo: Scheduling in UNIX

Scheduling

- L'algoritmo di scheduling favorisce i **job di tipo interattivo** (foreground)
- Si tratta di un **round-robin con priorità**
- Ad ogni processo è associata una priorità di scheduling. La priorità è rappresentata in senso decrescente
- La priorità **varia dinamicamente**: al crescere del tempo di CPU utilizzato da un processo diminuisce la sua priorità. Analogamente, al crescere del tempo di attesa di un processo aumenta anche la sua priorità
- Le priorità variano da 0 (massima) a 127 (minima). Da 0 a 49 per i processi che eseguono in modo kernel, da 50 a 127 per i processi in modo utente

- Viene usato il meccanismo di **time-out**: ogni quanto di tempo l'interruzione di clock mette in funzione una procedura che esegue l'azione richiesta e predisponde il clock per essere nuovamente chiamata
- Quando si verifica un **event**, il kernel provvede a risvegliare tutti i processi in attesa di **event**. I processi vengono messi in coda per essere scelti dal meccanismo di scheduling
- Possono nascere "condizioni di corsa" relative al meccanismo degli eventi. Se un processo decide di sospendersi in attesa di un evento e l'evento si verifica prima che il processo completi la primitiva **sleep**, il processo rimane in attesa indefinita (**deadlock**), una soluzione al problema consiste nell'impedire all'evento di verificarsi durante l'esecuzione della primitiva

Prima del **kernel 2.5** Linux utilizzava lo "0(1) scheduler". Per ogni CPU e per ogni livello di priorità abbiamo due code:

- **Active queue**: per processi che non hanno usato tutto il loro timeslice
- **Expired queue**: per quelli che l'hanno esaurito

se la coda active è vuota, le due code vengono scambiate.

Dal kernel 2.6.23 il **Completely Fair Scheduler** è lo scheduler di default per i task non RT: cerca di modellare una CPU multitasking ideale e precisa e che fornisce ad ogni processo la stessa capacità di elaborazione
Linux supporta il **bilanciamento di carico**, ma è anche consapevole di NUMA.

Il **dominio di scheduling** è un insieme di core della CPU che possono essere bilanciati tra loro

Esempi di sistema operativo: Scheduling di Windows

Scheduling

- Windows utilizza uno **scheduling preemptive basato sulle priorità**. I thread con priorità più alta vengono eseguiti successivamente
- Il **dispatcher** è lo scheduler
- Il thread viene eseguito fino a quando 1) non si blocca, 2) non utilizza un time slice, 3) non viene preempted da un thread a priorità superiore
- La **classe variabile** è 1-15, la **classe in tempo reale** è 16-31
- La priorità 0 è il thread di **gestione della memoria**
- Coda per ogni priorità

Classi di priorità di Windows

L'API Win32 identifica diverse classi di priorità a cui un processo può appartenere (*REALTIME_PRIORITY_CLASS*, *HIGH_PRIORITY_CLASS*,...)

Un thread all'interno di una determinata classe di priorità ha una priorità relativa.

La classe di priorità e la priorità relativa si combinano per dare una priorità numerica.

Valutazione degli algoritmi

Scheduling

Come selezionare l'algoritmo di schedulazione della CPU per un SO? Intanto dobbiamo **determinare i criteri**

- massimizzare l'utilizzazione della CPU con il vincolo che il tempo di risposta max sia 1 sec
- massimizzare il throughput in modo che il tempo di risposta sia proporzionale al tempo di esecuzione totale

Modellazione deterministica

- Tipo di valutazione analitica
- Prende un particolare carico di lavoro predeterminato e definisce le prestazioni di ogni algoritmo per quel carico di lavoro

Valutazione deterministica

Scheduling

Per ogni algoritmo, calcolare il tempo medio minimo di attesa

Modelli di code

Scheduling

Descrivono l'arrivo dei processi e dei burst di CPU e I/O in modo probabilistico.

Un sistema informatico descritto come una rete di server, ciascuno con una coda di processi di attesa.

Simulazioni

Scheduling

I modelli di code sono limitati.

Le simulazioni sono più accurate

- Modello programmato di sistema informatico
- Il clock è una variabile
- Si raccolgono le statistiche che indicano le prestazioni dell'algoritmo

Sommario

1. Introduzione

2. Evoluzione dei sistemi operativi

3. Appofondimento delle interruzioni

4. Servizi e organizzazione del SO

5. Virtualizzazione

6. Processi

7. Thread e concorrenza

8. Scheduling

9. Sincronizzazione dei processi

10. Sincronizzazione dei processi in ambiente globale mediante semafori

11. Deadlock

12. Sistema di protezione

13. Gestione della memoria principale e virtuale

Introduzione

Sincronizzazione dei processi

I task (processi e thread) possono essere eseguiti simultaneamente, nel modello ad ambiente globale, l'accesso concorrente dei task a dati condivisi in memoria può causare incoerenza dei dati a causa dell'interferenza. Il mantenimento della coerenza dei dati richiede meccanismi che assicurino l'esecuzione ordinata dei task cooperanti

Tipi di interazione tra processi

Sincronizzazione dei processi

Cooperazione:

- Comprende tutte le interazioni **prevedibili e desiderate**, insiste cioè nella logica dei programmi
- Prevede lo scambio di informazioni
 - Segnale temporale
 - Messaggi

Competizione

- La "macchina concorrente" su cui i processi sono eseguiti mette a disposizione un numero limitato di risorse
- Competizione per l'uso di risorse comuni che non possono essere usate contemporaneamente

Interferenza:

provocata da errori di programmazione

- Inserimento nel programma di interazioni tra processi **non richieste** dalla natura del problema
- Erronea soluzione a problemi di interazione necessari per il corretto funzionamento del programma

Esempi di interferenza

Sincronizzazione dei processi

- **Esempio di interferenza del primo tipo**

1. Solo il processo P deve operare su una risorsa R
2. Per un errore di programmazione viene inserita nel processo Q un'istruzione che modifica lo stato di R
3. La condizione di errore si presenta solo per particolari velocità relative dei processi

- **Esempio di interferenza del secondo tipo**

1. I processi P e Q competono per una stampante
2. Si garantisce la **mutua esclusione** solo per la stampa della prima linea
3. La condizione di errore si presenta solo per particolari velocità relative dei processi

Problema delle interferenze

Sincronizzazione dei processi

Un problema fondamentale della programmazione concorrente è l'eliminazione delle interferenze.

L'eliminazione delle interferenze del primo tipo risulta semplificata se la macchina concorrente fornisce **meccanismi di protezione degli accessi**. Per evitare le interferenze del secondo tipo, trattandosi di interazioni previste ma programmate in modo errato, è opportuno adottare tecniche di **multiprogrammazione strutturata**

Sezione critica

Sincronizzazione dei processi

Sezione critica

È la sequenza di istruzioni con le quali un processo accede e modifica un insieme di variabili comuni prende il nome di **sezione critica**

Ad un insieme di variabili comuni possono essere associate una sola sezione critica o più sezioni critiche.

La regola di mutua esclusione stabilisce che:

- "sezioni critiche appartenenti alla stessa classe devono escludersi mutuamente nel tempo"
- "una sola sezione critica di una classe può essere in esecuzione ad ogni istante"

Requisiti di una soluzione al problema della sezione critica

Sincronizzazione dei processi

- **Mutua esclusione:** se un processo P_i sta eseguendo la sua sezione critica, **nessun altro processo può eseguire la propria**
- **Avanzamento:** se nessun processo è in esecuzione nella sua sezione critica ed esistono processi che desiderano entrare nella loro sezione critica, la selezione dei processi che entreranno successivamente nella sezione critica non può essere rimandata all'infinito
- **Attesa limitata:** deve esistere un limite al numero di volte in cui è consentito ad altri processi entrare nella propria sezione critica dopo che un processo ha fatto richiesta di entrare nella propria sezione critica e prima che tale richiesta venga accolta

Soluzione al problema della mutua esclusione

Sincronizzazione dei processi

Tempificazione dell'esecuzione dei singoli processi da parte del programmatore.

Inibizione delle interruzioni del processore sul quale sono eseguite le sezioni critiche durante l'esecuzione di ciascuna di esse.

Strumenti di sincronizzazione: semafori e altri

Lock mutex

Sincronizzazione dei processi

Le soluzioni tramite algoritmi come quello di Peterson o istruzioni hardware di **test-and-set** atomiche sono complicate e generalmente inaccessibili ai programmatore di applicazioni.

I progettisti di SO costruiscono strumenti software per risolvere il problema delle sezioni critiche.

La più semplice è il **lock mutex**: protegge una sezione critica acquisendo (*acquire()*) prima un lock e poi rilasciandolo (*release()*) alla fine della sezione critica.

Sommario

1. Introduzione

2. Evoluzione dei sistemi operativi

3. Appocondimento delle interruzioni

4. Servizi e organizzazione del SO

5. Virtualizzazione

6. Processi

7. Thread e concorrenza

8. Scheduling

9. Sincronizzazione dei processi

10. Sincronizzazione dei processi in ambiente globale mediante semafori

11. Deadlock

12. Sistema di protezione

13. Gestione della memoria principale e virtuale

Semafori

Sincronizzazione dei processi in ambiente globale mediante semafori

Come garantire la mutua esclusione tra i processi nell'accesso alle sezioni critiche o alle risorse?

I **semafori**, utilizzati tramite le loro primitive **wait** e **signal**, sono uno strumento di sincronizzazione generale e flessibile al problema della mutua esclusione e ad altri problemi di sincronizzazione

Semaforo

Un **semaforo** è una variabile intera non negativa ($s \geq 0$) con valore iniziale $s_0 \geq 0$.

Al semaforo è associata una **lista di attesa** Q_s nella quale sono posti i descrittori dei processi che attendono l'autorizzazione a proseguire nell'esecuzione.

Wait e signal

Sincronizzazione dei processi in ambiente globale mediante semafori

wait(s) :

```
begin
  if s = 0 then
    <il processo viene sospeso e
      il suo descrittore inserito in Qs
  else s := s - 1;
end;
```

La *wait* può essere *passante* ($s > 0$) o *bloccante* ($s = 0$), nel qual caso si verifica un *context switch*

signal(s) :

```
begin
  if <esiste un processo in coda> then
    <il suo descrittore viene rimosso da Qs
      il suo stato modificato in pronto>;
  else s := s + 1;
end;
```

La *signal* è sempre *passante*

- ❑ L'esecuzione della signal (s) non comporta concettualmente alcuna modifica allo stato del processo che l'ha eseguita.
- ❑ La scelta del processo sospeso avviene tramite politica FIFO.

Mutua esclusione tramite semaforo

Sincronizzazione dei processi in ambiente globale mediante semafori

Ad ogni classe di sezioni critiche viene associata una **variabile semaforo s**; prologo ed epilogo vengono realizzati rispettivamente tramite *wait()* e *signal()*.

Sono risolti i problemi di attesa attiva e attesa indefinita. Un processo non può riappropriarsi della sezione critica che ha appena liberato se ci sono altre richieste pendenti

Indivisibilità di wait e signal

Sincronizzazione dei processi in ambiente globale mediante semafori

Occorre garantire che l'azione di analisi e modifica del semaforo non sia separata dall'azione di sospensione.

Si può ottenere indivisibilità **inibendo le interruzioni durante l'esecuzione di wait e signal**.

Nel caso di sistema multiprocessore occorre considerare *wait* e *signal* come **sezioni critiche brevi** e proteggerle mediante un meccanismo di più basso livello denominato **lock**

Lock e Unlock

Sincronizzazione dei processi in ambiente globale mediante semafori

Lock e Unlock devono essere **indivisibili**

Nell'ipotesi che l'Hw garantisca la mutua esclusione solo a livello di singola lettura o scrittura di una cella di memoria, solo *unlock(x)* è indivisibile.

Livelli di sezioni critiche

Sincronizzazione dei processi in ambiente globale mediante semafori

- I LIVELLO sezioni critiche: S_1, S_2 (codice a livello utente) mutua esclusione tramite *wait* e *signal*
- II LIVELLO sezioni critiche: *wait()* e *signal()*, mutua esclusione tramite *lock(x)* e *unlock(x)*
- III LIVELLO sezioni critiche: *lock(x)*, *unlock(x)*, mutua esclusione tramite hardware

Produttore-Consumatore

Sincronizzazione dei processi in ambiente globale mediante semafori

- La soluzione richiede due semafori:

"messaggio disponibile"
"spazio disponibile"

mess-disp	valore iniziale 0
spazio-disp	valore iniziale N

- ```
Produttore (P) Consumatore (C)
begin begin
repeat repeat
 <produzione messaggio> wait (mess-disp)
 wait (spazio-disp) <prelievo messaggio>
 <deposito messaggio> signal (spazio-disp)
 signal (mess-disp) <consumazione messaggio>
forever
end forever
end
```
- E' una soluzione *simmetrica*, non privilegia nessun processo.
- P e C possono operare in parallelo sul buffer su messaggi diversi: P e C non possono operare sul medesimo messaggio, indipendentemente dalla sua lunghezza. (P e C tentano di accedere allo stesso messaggio solo nelle condizioni limite di buffer pieno e buffer vuoto; in tali condizioni uno dei due processi è bloccato dalla wait).

# Regolazione dell'esecuzione di processi e thread

## Sincronizzazione dei processi in ambiente globale mediante semafori

### □ Problema:

- $n$  processi (o thread)  $P_1, P_2, \dots, P_n$  devono essere attivati ad intervalli di tempo prefissati da un processo gestore  $P_0$
- l'esecuzione di  $P_i$  non può iniziare prima che sia giunto il segnale da  $P_0$
- ad ogni segnale inviato da  $P_0$  deve corrispondere un'attivazione di  $P_i$

### □ Soluzione:

- Definiamo  $n$  semafori  $s_i$  con valore iniziale  $s_{i0}=0$

$P_0$

...  
*repeat*

wait (si);  
<do something>

*forever*

...  
*repeat*

...  
signal (si);

...  
*forever*

# Programmazione concorrente e semafori

## Sincronizzazione dei processi in ambiente globale mediante semafori

I semafori sono uno strumento **potente e generale** per la soluzione dei problemi di programmazione concorrente in ambiente globale, sia per competizione che per cooperazione di thread e processi.

I semafori sono uno **strumento potente ma a basso livello**: è difficile risolvere problemi complessi utilizzando semafori

Si utilizzano pertanto **meccanismi di più alto livello**:

- Costrutti come Monitor, Regioni Critiche
- Librerie per la programmazione multithread
- Adottando pattern progettuali di provata affidabilità ed efficacia

# Monitor

Sincronizzazione dei processi in ambiente globale mediante semafori

Un'astrazione di alto livello che fornisce un meccanismo comodo ed efficace per la sincronizzazione dei processi.

Tipo di dati astratto, variabili interne accessibili solo dal codice all'interno della procedura.

**All'interno del monitor può essere attivo un solo processo alla volta**

# Buffer di comunicazione nel modello a scambio di messaggi

Sincronizzazione dei processi in ambiente globale mediante semafori

È necessario un **processo gestore** (buffer control) della risorsa buffer che serve i processi produttori  $P_i$  e i processi consumatori  $C_j$ .

- $P_i$  manda un messaggio al processo *buffer control* che a sua volta lo deve inviare ad uno dei processi  $C_j$
- Ciascun  $C_j$  deve inviare un messaggio a *buffer control* per indicare che è pronto a ricevere il messaggio
- Devono esistere due code per memorizzare i due tipi di messaggi *dataq* e *readyq*

# Sommario

1. Introduzione

2. Evoluzione dei sistemi operativi

3. Appofondimento delle interruzioni

4. Servizi e organizzazione del SO

5. Virtualizzazione

6. Processi

7. Thread e concorrenza

8. Scheduling

9. Sincronizzazione dei processi

10. Sincronizzazione dei processi in ambiente globale mediante semafori

11. Deadlock

12. Sistema di protezione

13. Gestione della memoria principale e virtuale

# Deadlock (blocco diretto)

## Deadlock

Più processi possono entrare in competizione per ottenere l'uso di risorse. Se la **risorsa richiesta non è disponibile il processo viene posto in condizione di attesa**.

Se un processo in attesa non cambia più il suo stato, cioè se le risorse sono trattenute da altri processi essi pure in attesa, si ha una **situazione di deadlock** (blocco critico in stallo).

### Blocco critico

Per blocco critico si intende una situazione nella quale uno o più processi rimangono **indefinitamente bloccati** a causa della impossibilità del verificarsi delle condizioni necessarie per il loro proseguimento

# Risorse riusabili e risorse consumabili

## Deadlock

### Risorse riusabili

Dopo il loro uso da parte di un processo possono essere usate da altri processi

### Risorse consumabili

- Sono segnali o messaggi scambiati tra processi
- Cessano di esistere non appena acquisite da un processo
- Sono potenzialmente in numero infinito

# Modello di sistema

## Deadlock

Il sistema è costituito da risorse.

Tipi di risorse  $R_1, R_2, \dots, R_m$

Ogni tipo di risorsa  $R_i$  ha  $W_i$  istanze.

Ogni processo utilizza una risorsa come segue:

- **Richiesta**
- **Utilizzo**
- **Rilascio**

# Esempio di deadlock

## Deadlock

Due processi A e B necessitano entrambi delle risorse  $R_1$  (disco) e  $R_2$  (stampante)

- A richiede e ottiene R1
- B richiede e ottiene R2
- A richiede R2 e viene bloccato
- B richiede R1 e viene bloccato

Deadlock provocato da un **uso scorretto delle primitive di sincronizzazione**

Deadlock provocato da un **accesso non ordinato alle risorse**

# Grafo di allocazione delle risorse

## Deadlock

$G = \{V, E\}$ , dove V è l'insieme dei vertiici ed E quello dei lati.

L'insieme V è suddiviso in due tipi (**grafo bipartito**): nell'insieme dei processi e nell'insieme delle risorse.

Quando una richiesta è **soddisfatta** il lato di richiesta viene trasformato in lato di assegnazione. Quando il processo rilascia la risorsa, il lato viene eliminato.

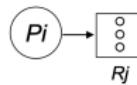
- Processo



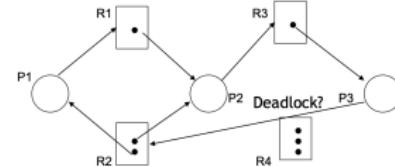
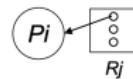
- Tipo di risorsa con 4 istanze



- Pi richiede un'istanza di Rj



- Pi è in possesso di un'istanza di Rj



- Se il grafo non contiene cicli, non c'è deadlock
- Se il grafo contiene un ciclo ci può essere deadlock (condizione necessaria); il SO può effettuare una verifica dei processi coinvolti nel ciclo per verificare se c'è deadlock
- Se è presente una sola istanza per ogni tipo di risorsa e c'è un ciclo allora c'è deadlock (necessaria e sufficiente)
- Se, a partire dallo stato in figura, P3 richiede un'istanza di R2?

# Condizioni per il deadlock

## Deadlock

È **necessario**, ma **non sufficiente** che si verifichino quattro condizioni per il deadlock

1. **Mutua esclusione**: ogni risorsa risulta assegnata esattamente ad un processo oppure è disponibile
2. **Processo e attesa (hold and wait)**: un processo che detiene almeno una risorsa può richiederne altre che sono detenute da altri processi
3. **Assenza di revoca**: le risorse precedentemente assegnate possono essere rilasciate solo volontariamente
4. **Attesa circolare**: è presente una situazione di attesa circolare tra un gruppo di processi: ogni processo è in attesa di una risorsa posseduta dal processo che esegue nella lista, e l'ultimo processo è in attesa di una risorsa detenuta dal primo processo considerato

# Strategie per ignorare il deadlock

## Deadlock

1. Ignorare il problema: è meglio accettare deadlock occasionali piuttosto che dover utilizzare una sola risorsa alla volta
2. Rilevare la presenza di deadlock e risolverlo: approccio denominato *detection and recovery*. Periodicamente il SO controlla un grafo di allocazione delle risorse, se c'è un ciclo (deadlock) si terminano uno o più processi nel ciclo fino a quando non vi è più deadlock.
3. Prevenzione dinamica: allocazione attenta delle risorse, vincolare i processi in modo che il deadlock risulti strutturalmente impossibile mediante decisioni prese a tempo di esecuzione.

### Algoritmo del banchiere

- L'algoritmo richiede di conoscere a priori il massimo numero di risorse richeiste da ciascun processo e presuppone, come caso peggiore, che ogni processo possa richiedere contemporaneamente il numero massimo di risorse dichiarato e mantenerle tutte durante l'esecuzione
- L'algoritmo identifica le situazioni rischiose e nega l'assegnazione di risorse disponibili quando la loro attribuzione potrebbe portare ad un deadlock
- Un costo quadratico viene pagato ad ogni richiesta di allocazione di una risorsa libera
- Appropriato per i sistemi di elaborazione con carico statico noto

4. Prevenzione strutturale (statica)

- Occorre negare una delle 4 condizioni di Havender
- Negare la mutua esclusione?
- Negare il possesso e attesa?
- Negare la preemption?
- Normalmente si evita l'instaurarsi di una condizione di attesa circolare imponendo un ordine sulla richiesta delle risorse

# Deadlock nei sistemi reali

## Deadlock

Nei sistemi interattivi, il deadlock è in genere ignorato nel codice utente, il deadlock è sempre prevenuto staticamente nei sistemi in tempo reale.

Il deadlock è prevenuto all'interno del kernel del SO; i programmatore del SO seguono un ordine concordato nell'acquisire le risorse.

Il deadlock viene talvolta prevenuto dinamicamente o con tecniche di *detection and recovery* nei sistemi gestionali con carico transazionale stabile e in alcuni sistemi di calcolo

# Sommario

1. Introduzione

2. Evoluzione dei sistemi operativi

3. Appofondimento delle interruzioni

4. Servizi e organizzazione del SO

5. Virtualizzazione

6. Processi

7. Thread e concorrenza

8. Scheduling

9. Sincronizzazione dei processi

10. Sincronizzazione dei processi in ambiente globale mediante semafori

11. Deadlock

12. Sistema di protezione

13. Gestione della memoria principale e virtuale

# Introduzione

## Sistema di protezione

Un processo potrebbe tentare di modificare il programma o i dati di un altro processo o di parte del SO stesso, o di accedere direttamente ai controllori di dispositivi.

All'**hardware** è affidato il **compito di rilevazione degli errori** come possibili errori di programmazione o comportamenti deliberatamente intrusivi.

Gli **errori** vengono segnalati e affidati alla **gestione del SO** tramite il meccanismo delle eccezioni/trap

Abbiamo già visto come il sistema di protezione richieda l'esistenza di **più modi di funzionamento della CPU**:

- **Supervisor mode:** monitor mode
- **User mode**

Il passaggio dal modo **user** a quello **supervisor** avviene tramite l'interruzione (esterna, asincrona o interna, sincrona o da trap).

Il passaggio dal modo **supervisor** al modo **user** avviene tramite un'istruzione speciale di cambiamento di modo eseguita dal SO prima della cessione del controllo ad un processo utente

# Principi di protezione

## Sistema di protezione

Principio guida, **principio del minimo privilegio**.

- Ai programmi, agli utenti e ai sistemi devono essere concessi solo i **privilegi sufficienti** per svolgere i loro compiti
- **Permessi** impostati correttamente possono limitare i danni se l'entità presenta un bug o viene abusata
- Possono essere statici o dinamici **cambio di dominio, escalation di privilegi**.
- La compartimentazione è un concetto derivato per quanto riguarda l'accesso ai dati
- **Audit trail** - registrazione di tutte le attività orientate alla protezione, importante per capire cosa è successo e perché, e per individuare le cose che non dovrebbero succedere

# Difesa in profondità

## Sistema di protezione

Abbiamo **più livelli di protezione**, introdotti a livello di progettazione.  
Ogni dispositivo di sicurezza è in qualche misura vulnerabile e può essere difettoso.

Si assume che i livelli siano indipendenti, il fallimento di un livello non deve essere collagato ai fallimenti degli altri

# Misure applicabili ai diversi livelli

## Sistema di protezione

- Organizzazione e aspetti umani: formazione, consapevolezza, gestione del rischio
- Sicurezza fisica: protezione fisica, serrature, dispositivi di tracking
- Perimetro: firewall
- Rete: segmentazione rete, IPSec
- Endpoint
- Applicazioni
- Dati

# Anelli di protezione

## Sistema di protezione

Componenti ordinati in base alla qualità di privilegi e protetti l'uno dall'altro

- **Gate utilizzati per il passaggio da un livello all'altro**
- **Gli hypervisor** hanno introdotto la necessità di un ulteriore anello
- I processori ARMv7 hanno aggiunto l'anello **TrustZone(TZ)** per proteggere le funzioni crittografiche con accesso tramite la nuova istruzione **Secure Monitor Call (SMC)**

# Dominio di protezione

## Sistema di protezione

Gli anelli di protezione separano le funzioni in domini e le ordinano in modo gerarchico.

I computer possono essere trattati come processi e oggetti: **oggetti hardware** (come dispositivi) e **oggetti software** (come i file, i programmi, i semafori).

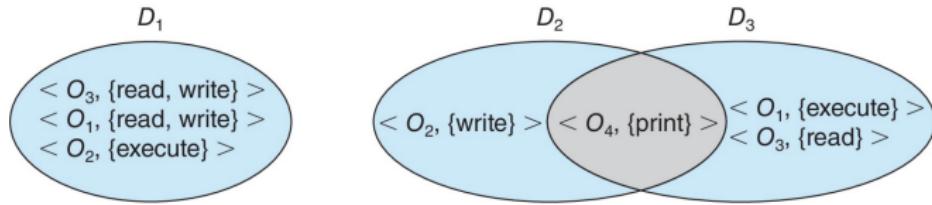
Un processo, ad esempio, dovrebbe avere **accesso solo agli oggetti** di cui **ha bisogno per portare a termine il suo compito**, il principio del **need-to-know**.

L'implementazione può avvenire tramite processi che operano in un **dominio di protezione**, che specifica le risorse a cui un processo può accedere

# Struttura del dominio

## Sistema di protezione

- Access-right = *<object-name, rights-set>*  
dove Access-right, il diritto di accesso, è un sottoinsieme di tutte le operazioni valide che possono essere eseguite sull'oggetto
- Dominio = un insieme di diritti di accesso



# Implementazione del dominio - UNIX

## Sistema di protezione

### Cambio di dominio tramite il *file system*

- Ad ogni file è associato un bit di dominio
- Quando il file viene eseguito e setuid è attivo allora l'ID utente viene impostato su quello del proprietario del file in esecuzione
- Cambio di dominio tramite password
- Comutazione di dominio tramite comandi (*sudo*)

# Implementazione del dominio - ID app Android

## Sistema di protezione

In Android gli ID utente distinti sono forniti su base applicativa. Quando una app viene installata il deamon installd le assegna un ID utente e un ID gruppo distinti, insieme a una directory di dati privata la cui proprietà è concessa solo a questa combinazione di ID utente e ID gruppo.

# Matrice di accesso

## Sistema di protezione

- Visualizzazione delle proteizone come matrice
- Le righe rappresentano i domini
- Le colonne rappresentano gli oggetti
- $\text{Access}(i,j)$  è l'insieme delle operazioni che un processo in esecuzione nel Dominio può invocare sull'Oggetto

Se un processo nel dominio  $D_i$  tenta di fare "op" sull'oggetto  $O_j$ , allora "op" deve essere nella matrice di accesso

| object<br>domain | $F_1$ | $F_2$ | $F_3$   | printer |
|------------------|-------|-------|---------|---------|
| $D_1$            | read  |       | read    |         |
| $D_2$            |       |       |         | print   |
| $D_3$            |       | read  | execute |         |

Con la matrice di accesso si separa il meccanismo dalla politica

- **Meccanismo**

- Il SO fornisce una matrice di accesso e delle regole
- Si assicura che la matrice sia manipolata solo da agenti autorizzati e che le regole siano rigorosamente applicate

- **Politica**

- L'utente detta la politica
- Chi può accedere a quale oggetto e in quale modalità

# Implementazione della matrice di accesso

## Sistema di protezione

in generale, una matrice sparsa

- **Opzione 1 - Tabella globale**

- memorizzazione di triple ordinate  
 $< \text{Dominio}, \text{Oggetto}, \text{Set di diritti} >$  nella tabella
- ma la tabella potrebbe essere troppo grande per la memoria principale

- **Opzione 2 - Liste di accesso per gli oggetti**

- ogni colonna è implementata come lista di accesso per un oggetto
- l'elenco risultante per oggetto consiste in coppie ordinate  
 $< \text{Dominio}, \text{Set di diritti} >$  che definiscono tutti i domini con un insieme non vuoto di diritti di accesso per l'oggetto

- **Opzione 3 - Lista delle capability per i domini**
  - la lista per le capability per il dominio è un elenco di oggetti insieme alle operazioni consentite su di essi
  - oggetto rappresentato dal suo nome o indirizzo, chiamato **capability**
  - Elenco di capability associate al dominio ma mai direttamente accessibili dal dominio
- **Opzione 4 - Lock key**
  - Compromesso tra liste di accesso e liste di capability
  - Ogni oggetto ha un elenco di pattern e bit unici, chiamati lock
  - Ogni dominio ha un elenco di pattern di bit unici, chiamati chiavi

# Confronto tra implementazioni

## Sistema di protezione

Molti compromessi da considerare

- la tabella globale è semplice, ma può essere molto grande
- le access list corrispondono alle esigenze degli utenti: determinare l'insieme dei diritti di accesso per un dominio non localizzato è difficile
- le liste di capability sono utili per localizzare le informazioni per un determinato processo
- lock-key efficace e flessibile, le chiavi possono essere passate liberamente da un dominio all'altro

# Revoca dei diritti di accesso

## Sistema di protezione

Varie opzioni per rimuovere il diritto di accesso di un dominio ad un oggetto:

- Immediata o ritardata
- Selettivo o generale
- Parziale o totale
- Temporaneo o permanente

*Access list* - Eliminare i diritti di accesso dall'access list è **facile e immediato**, basta ricercare una voce nella lista ed eliminarla

*Elenco delle capability* - Schema necessario per individuare le capability nel sistema prima di poterle revocare

- Riacquisizione - cancellazione periodica
- Back-pointer - insieme di puntatori da ogni oggetto a tutte le capability di quell'oggetto
- Indirezione - la capacità punta alla voce della tabella globale che punta all'oggetto
- Chiavi - bit univoci associati alla capability, generati quando la capability viene creata

# Role-based Access Control

## Sistema di protezione

La protezione può essere applicata anche a risorse diverse dai file.

Oracle Solaris 10 fornisce un **controllo degli accessi basato sui ruoli (RBAC)** per implementare il minimo privilegio.

Il **privilegio** è il diritto di eseguire una chiamata di sistema o di utilizzare un'opzione all'interno di una chiamata di sistema.

Agli utenti vengono assegnati **ruoli** che garantiscono l'accesso a privilegi e programmi

# Controllo dell'accesso obbligatorio (MAC)

## Sistema di protezione

Il SO tradizionalmente hanno un controllo di accesso discrezionale (DAC) per limitare l'accesso ai file e ad altri oggetti, la **discrezionalità è un punto debole**

Una forma più forte è il controllo obbligatorio degli accessi (MAC), che nemmeno l'utente root può aggirare.

La base sono le etichette assegnate agli oggetti e ai soggetti (compresi i processi), quando un soggetto richiede l'accesso ad un oggetto, la politica controlla se un determinato soggetto con etichetta è autorizzato o meno a eseguire l'azione sull'oggetto

# Sistemi basati sulle capability

## Sistema di protezione

- Hydra e CAP sono stati i primi sistemi basati sulle capability
- Ora incluse in Linux, Android e altre, basate su POSIX.1e
  - essenzialmente suddivide i poteri root in aree distinte, ognuna rappresentata da una bit e da una bitmap
  - il controllo a grana finne sulle operazioni privilegiate può essere ottenuto impostando o mascherando la bitmap

# Altri metodi di miglioramento della protezione

## Sistema di protezione

### Protezione dell'integrità del sistema (SIP)

- Introdotto da Apple in macOS 10.11
- Limita l'accesso ai file e alle risorse di sistema, anche da parte di root
- Utilizza attributi di file estesi per contrassegnare un binario per limitare le modifiche, disabilitare il debug e lo scrutinio

### Filtraggio delle chiamate di sistema

- Come un firewall, per le chiamate di sistema
- Può anche essere più profondo
- Linux implementa tramite SECCOMP-BPF

### Sandboxing

- Esecuzione del processo in un ambiente limitato
- Imporre un insieme di restrizioni inammovibili all'inizio dell'avvio del processo
- Il processore non è in grado di accedere a nessun risorsa oltre a quelle consentite
- Apple è stata una delle prime ad adottare questo sistema, a partire dalla funzione "seabelt" di macOS 10.5

### Firma del codice da parte del programmatore

# Protezione basata sul linguaggio di programmazione

## Sistema di protezione

La specifica della protezione in un linguaggio di programmazione consente di descrivere ad alto livello le politiche per l'allocazione e l'uso delle risorse. L'implementazione del linguaggio può fornire software per l'applicazione della protezione quando il controllo automatico supportato dall'Hw non è disponibile

# Protezione in Java 2

## Sistema di protezione

La protezione è gestita dalla macchina virtuale Java (JVM).

A una classe viene assegnato un dominio di protezione quando viene caricata dalla JVM.

Se viene invocato un metodo di libreria che esegue un'operazione privilegiata, lo stack viene ispezionato per assicurarsi che l'operazione possa essere eseguita dalla libreria

# Sommario

1. Introduzione

2. Evoluzione dei sistemi operativi

3. Appofondimento delle interruzioni

4. Servizi e organizzazione del SO

5. Virtualizzazione

6. Processi

7. Thread e concorrenza

8. Scheduling

9. Sincronizzazione dei processi

10. Sincronizzazione dei processi in ambiente globale mediante semafori

11. Deadlock

12. Sistema di protezione

13. Gestione della memoria principale e virtuale

# Introduzione

## Gestione della memoria principale e virtuale

Il programma deve essere portato (da disco) in memoria e collocato all'interno di un processo per poter essere eseguito.

La memoria principale e i registri sono l'unica memoria a cui la CPU può accedere direttamente.

L'accesso ai registri avviene in un ciclo di clock della CPU.

La **cache** si colloca tra la memoria principale e i registri della CPU

# Protezione della memoria con registri limite

## Gestione della memoria principale e virtuale

Una possibile implementazione: prima di mettere un processo in esecuzione, il SO ne confina lo spazio logico di memoria mediante i registri limite

# Protezione della memoria in Hw con registri limite

## Gestione della memoria principale e virtuale

La CPU deve controllare ogni accesso alla memoria generato in modalità utente per assicurarsi che sia compreso tra la base e il limite per quell'utente.

Le istruzioni per caricare i registri base e limite sono privilegiate

# Binding degli indirizzi

## Gestione della memoria principale e virtuale

I programmi su disco, pronti per essere portati in memoria per essere eseguiti, formano una coda di ingresso.

Scomodo avere l'indirizzo fisico del primo processo utente sempre a 0.

Indirizzi rappresentati in modi diversi in fasi diverse della vita di un programma

- Gli indirizzi del codice sorgente di solito sono simbolici
- Gli indirizzi del codice compilato **si legato (bind)** a indirizzi rilocabili
- Il linker o il loader legheranno gli indirizzi rilocabili agli indirizzi assoluti

# Binding di istruzioni e dati alla memoria

## Gestione della memoria principale e virtuale

Il binding delle istruzioni e dei dati agli indirizzi di memoria può avvenire in tre fasi diverse

- **A tempo di compilazione:** se la posizione in memoria è nota a priori, è possibile generare **codice assoluto**, è necessario ricompilare il codice se la posizione iniziale cambia
- **A tempo di caricamento:** deve essere gerato **codice rilocabile** se la posizione in memoria non è nota al momento della compilazione
- **A tempo di esecuzione:** binding ritardato fino al tempo di esecuzione se il processo può essere spostato durante la sua esecuzione da un segmento di memoria ad un altro

# Spazio degli indirizzi logici e fisici

## Gestione della memoria principale e virtuale

Il concetto di uno spazio di indirizzo logico che viene legato ad uno **spazio di indirizzamento fisico** separato è centrale nella gestione della memoria

- **Indirizzo logico:** generato dalla CPU, indicato anche come **indirizzo virtuale**
- **Indirizzo fisico:** indirizzo visto dall'unità di memoria

Lo **spazio degli indirizzi logici** è l'insieme di tutti gli indirizzi logici generati da un programma.

Lo **spazio degli indirizzi fisici** è l'insieme di tutti gli indirizzi fisici generati da un programma

# Unità di gestione della memoria (MMU)

## Gestione della memoria principale e virtuale

Dispositivo hardware che a tempo di esecuzione mappa gli indirizzi virtuali in quelli fisici.

Molti metodi possibili per il mapping

Si consideri uno schema semplice, che è una generalizzazione dello schema dei registri di base.

Il registro di base ora si chiama **registro di rilocazione**.

Il valore nel registro di rilocazione viene aggiunto a ogni indirizzo generato da un processo utente nel momento in cui viene inviato alla memoria.

# Caricamento dinamico

## Gestione della memoria principale e virtuale

- Non è necessario che l'intero programma sia in memoria per poter essere eseguito
- Una routine non viene caricata fino a quando non viene chiamata
- Migliore utilizzo dello spazio di memoria, una routine non utilizzata non viene mai caricata
- Utile quando sono necessarie grandi quantità di codice per gestire casi poco frequenti

# Linking dinamico

Gestione della memoria principale e virtuale

- **Linking statico** - librerie di sistema e codice di programma combinati dal loader nel file eseguibile del programma
- **Linking dinamico** - linking rimandato al momento dell'esecuzione

Il SO controlla se la routine si trova nell'indirizzo di memoria del processo.

Il collegamento dinamico è particolarmente utile per le librerie.

Sistema noto anche come **librerie convise** (shared libraries o DLL)

# Allocazione contigua

## Gestione della memoria principale e virtuale

La memoria principale deve supportare sia il sistema operativo che i processi utente.

Risorsa limitata, deve essere allocata in modo efficiente.

La **memoria principale** è solitamente suddivisa in **due partizioni**:

- Sistema operativo residente, di solito nella parte bassa della memoria
- Processi utente vengono tenuti nella parte alta della memoria
- Ogni processo è contenuto in una singola sezione contigua di memoria

I registri di rilocazione sono usati per proteggere i processi utente uno dall'altro, ed impedire modifiche al codice e ai dati del SO

- Il registro base contiene il valore dell'indirizzo fisico più piccolo
- Il registro limite contiene un intervallo di indirizzi logici
- La MMU mappa gli indirizzi logici in modo dinamico

# Partizionamento variabile

## Gestione della memoria principale e virtuale

### Allocazione di partizioni multiple

- Dimensioni variabili delle partizioni per l'efficienza
- **Buco (hole)** - blocco di memoria disponibile, buchi di varie dimensioni sono sparsi in tutta la memoria
- Il processo che esce libera la sua partizione, le partizioni libere adiacenti si combinano tra loro
- Il sistema SO mantiene informazioni su:
  1. partizioni allocate
  2. partizioni libere (buchi)

# Problema di allocazione dinamica dello storage

## Gestione della memoria principale e virtuale

Come può soddisfare una richiesta di dimensione  $n$  da una lista di buchi liberi?

- **First-fit:** alloca il **primo** buco sufficientemente grande
- **Best-fit:** alloca il buco **più piccolo** che sia sufficientemente grande; deve cercare nell'intero elenco
- **Worst-fit:** alloca il buco più grande, deve anche cercare nell'intero elenco

# Frammentazione

## Gestione della memoria principale e virtuale

- **Frammentazione esterna:** lo spazio di memoria totale esiste per soddisfare una richiesta, ma non è contiguo
- **Frammentazione interna:** la memoria allocata può essere leggermente più grande di quella richiesta; questa differenza di dimensione è memoria interna a una partizione, ma che non viene utilizzata

Ridurre la frammentazione esterna con la compattazione

Problema di I/O:

- Bloccare un job in memoria mentre è coinvolto nell'I/O
- Eseguire l'I/O solo nei buffer del sistema operativo

# Spazio di indirizzamento virtuale e spazio di indirizzamento fisico

## Gestione della memoria principale e virtuale

Il processore genera un indirizzo (virtuale) che non coincide con l'indirizzo fisico di accesso alla memoria

# Astrazione memoria virtuale

## Gestione della memoria principale e virtuale

Il processore accede per conto del processore ad uno spazio di indirizzamento **logico**

# Memoria virtuale: motivazioni

## Gestione della memoria principale e virtuale

Il codice deve essere in memoria per essere eseguito, ma raramente l'intero programma è utilizzato.

L'intero codice del programma non è necessario allo stesso tempo.

Si consideri la capacità di eseguire programmi parzialmente caricati

# Memoria virtuale

## Gestione della memoria principale e virtuale

**Memoria virtuale:** separazione dalla memoria logica dei programmi alla memoria fisica

- solo una parte del programma è necessaria in memoria per l'esecuzione
- lo spazio di indirizzamento logico può essere molto più grande dello spazio di indirizzamento fisico
- consente agli spazi di indirizzamento di essere condivisi da più processi
- la memoria virtuale può essere implementata con: **demand segmentation, demand paging** (paginazione su richiesta)

# Segmentazione

Gestione della memoria principale e virtuale

Traduzione di indirizzi con tabella dei segmenti

# Paginazione

## Gestione della memoria principale e virtuale

Lo spazio di indirizzamento fisico di un processo può essere non contiguo, al processo viene allocata memoria fisica appena questa è disponibile.

Divide la **memoria fisica** in blocchi di dimensione fissa "frame".

Divide la **memoria logica** in blocchi detti pagine.

Va predisposta una **tabella delle pagine** (*page table*) per tradurre gli indirizzi logici in quelli fisici **per ogni processo**.

Anche la memoria ausiliaria è divisa in pagine