Sistemi Operativi

Giacomo Simonetto e Diego Chiesurin

Secondo Semestre 2024-25

Sommario

Appunti del corso di Sistemi Operativi della facoltà di Ingegneria Informatica dell'Università di Padova.

Indice

1 Introduzione e struttura dei sistemi operativi

1.1 Introduzione e compiti di un sistema operativo

Il sistema operativo è un software che agisce da intermediario tra l'utente e l'hardware del computer, deve fornire servizi e caratteristiche presenti a tutti i suoi utenti attraverso l'interfacciamento di hardware e applicativi e deve gestire le varie risorse hardware a disposizione.

Funzioni del sistema operativo / problemi che deve risolvere

- fornire all'utente applicativi (programmi e comandi) per far eseguire operazioni all'elaboratore
- coordinare l'esecuzione dei programmi dell'utente e dei processi del sistema (es. multithreading)
- gestire le risorse (memoria, I/O, ...) efficientemente e prevenire errori dal loro uso improprio

Architettura di un sistema di calcolo

Il sistema di calcolo è l'insieme di hardware, sistema operativo, programmi di sistema e applicativi con la possibilità di accedere alle risorse hardware.

Struttura a cipolla

La struttura del sistema di calcolo è descritta con il modello a cipolla, composto da una serie di gusci concentrici, che racchiudono l'hardware, posto al centro, e i diversi strati di software che lo gestiscono/utilizzano. I gusci rappresentano programmi, che operano a livelli diversi di interazione uomo-macchina, ogni strato sfrutta gli strati sottostanti e fornisce funzioni per quelli superiori. L'utente interagisce solo con lo strato più esterno detto shell, mentre lo strato più interno a diretto contatto con l'hardware è detto kernel.

I SO dei dispositivi mobili comprendono anche un middleware, ovvero uno strumento che si posiziona tra hardware e software per semplificare la programmazione ed aumentare la portabilità siccome gli hardware sono estremamente diversificati.

1.2 Caricamento del sistema operativo

Caricamento con BIOS

Il bootstrap consiste in tutte le operazioni e programmi che vengono eseguiti dall'accensione allo spegnimento di un computer. In fase di accensione il SO viene caricato attraverso una procedura che utilizza il BIOS (ormai non più usata). Il BIOS (Basic Input Output System) o firmware è salvato nella ROM ed è costituito da routine software che forniscono funzioni base eseguibili sull'hardware. La prima procedura prevede che

1. il BIOS esegue il caricamento del bootloader dalla partizione MBR (Master Boot Record) del disco
2. il bootloader inizializza (e controlla) tutte le componenti del sistema e procede al caricamento del kernel del SO e ne lancia l'esecuzione
3. il kernel avvia i programmi di sistema, che garantiscono i servizi fondamentali per l'esecuzione dei programmi utente

Caricamento con UEFI

Nei computer moderni è ancora presente il BIOS, ma la procedura UEFI (Unified Extensible Firmware Interface) ne controlla l'esecuzione attraverso una interfaccia grafica, e ne estende le funzionalità con la possibilità di indirizzare più memoria ed eseguire compiti più complessi come strumenti per la diagnostica e il ripristino dei dati, servizi di crittografia e funzionalità per la gestione dei consumi. La procedura UEFI prevede che:

1. l'UEFI carica il bootloader dalla cartella EFI nella partizione del disco dedicata al sistema operativo (non più da una partizione apposita)
2. il bootloader carica il kernel che a sua volta caricherà il resto del SO

1.3 Servizi offerti dal sistema operativo

Servizi user oriented

- **interfaccia utente**: strumento con cui l'utente interagisce con il computer, a riga di comando (Command Line Interface) e/o grafica (Graphical User Interface)
- **esecuzione di programmi**: caricare ed eseguire comandi correttamente, su richiesta dell'utente
- **operazioni di I/O**: interazione con le periferiche connesse
- **gestione del filesystem**: creare, leggere, scrivere, cancellare files e muoversi tra le directory
- **comunicazioni**: gestione dello scambio di informazioni tra processi in esecuzione
- **rilevamento di errori**: individuare e agire opportunamente se si verificano errori

Servizi per il funzionamento corretto ed efficiente del sistema

- **gestione delle risorse**: assegnazione delle risorse e monitoraggio del loro utilizzo da parte del sistema e degli utenti
- **protezione**: garantire un accesso controllato alle risorse
- **sicurezza**: fornire mezzi di autenticazione e proteggere le risorse da accessi illegali

Interfaccia CLI

Interfaccia utente a riga di comando che permette all'utente di impartire comandi al sistema. Il sistema può offrire più shell, ovvero ambienti CLI diversi (bash, zsh) e incorpora un interprete dei comandi che esegue i comandi richiesti dall'utente. I comandi possono far parte built-in dell'interprete, oppure possono essere customizzati dall'utente.

Interfaccia GUI

L'interfaccia grafica permette una più intuitiva iterazione con l'utente tramite mouse, tastiera e monitor. I files, le cartelle e i programmi sono rappresentati da icone con cui è possibile interagire con il mouse. Le prime interfacce grafiche sono state sviluppate dalla Xerox PARC di Palo Alto e sono state adottate subito dalla Apple.

Chiamate a sistema o system call

Le chiamate a sistema sono gli strumenti che permettono di accedere ai servizi del sistema operativo, sono realizzate in linguaggi ad alto livello (C, C++) e sono richiamate attraverso le API (Application Programming Interface). Esempi di API sono: Win64 API per Windows, POSIX API per Unix, Linux e macOS e Java API per Java Virtual Machine.

Quando viene effettuata una chiamata al sistema, avviene il cambio di mode bit (da user mode a kernel mode) e quando la routine del sistema termina, restituisce il risultato al programma chiamante ritorna al bit mode iniziale (da kernel mode a user mode).

Il passaggio dei parametri alle chiamate a sistema può avvenire in tre modi: salvandoli nei registri, salvandoli in un'area di memoria e passare l'indirizzo di memoria in un registro (preferita da Linux ¿5 e Solari) oppure tramite lo stack

Esempi di alcune chiamate a sistema in Linux:

- `fork`, `exit`, `wait/waitpid`, `exec execve`, `signal`, `kill` per controllo dei processi
- `open`, `read`, `write`, `close` per gestione dei files
- `ioctl`, `read`, `write` per gestione dei dispositivi
- `getpid`, `ps`, `alarm`, `sleep` per recupero informazioni
- `pipe`, `shmget`, `mmap` per comunicazione tra processi
- `alloc`, `free` per assegnazione e rilascio della memoria

Programmi di sistema e servizi

I programmi di sistema sono una interfaccia conveniente e semplificativa delle system call. Vengono invocati da terminale e sono lo strumento con cui l'utente interagisce con il sistema. Si dividono in due categorie:

- **servizi in background** (servizi, sottosistemi o daemon) che vengono avviati in fase di boot e permettono il corretto funzionamento del sistema, vengono esegui in modalità utente
- **programmi applicativi** non fanno parte del sistema, ma sono scaricati dall'utente

Compilazione, ABI, linker e loader

Il risultato della **compilazione** è specifico di ogni sistema, ovvero non è universale per tutti i SO, su Unix si utilizza il formato ELF (Executable and Linkable Format), in Windows si usa il formato PE (Portable Executable) e in macOS si usa Mach-O.

I dettagli di come un file compilato agisce sul sistema operativo (chiamate a sistema, passaggio dei parametri, formato degli eseguibili, gestione della memoria, ...) sono contenute nelle **ABI** (Application Binary Interface).

Il **linker** è lo strumento che unisce i diversi file oggetto rilocabili ottenuti dalla compilazione di un determinato programma e dalle relative librerie statiche. Nei moderni sistemi si predilige il linking dinamico delle librerie dinamiche (DLL in Windows) che vengono caricate e condivise dai vari programmi in base alle necessità.

Il **loader** è lo strumento che carica il file eseguibile e le necessarie dipendenze dalla memoria secondaria alla memoria primaria.

1.4 Progettazione e sviluppo dei SO

Linguaggi con cui è scritto un sistema operativo

Il sistemi vengono sviluppati con un mix di linguaggi per favorirne la portabilità, la leggibilità e la facilità nella manutenzione: Si usa assembly per i driver dei dispositivi, il C per il kernel e altri linguaggi di alto livello come C, C++, shell script, Python per programmi di sistema e applicativi.

Sistemi monolitici

Nei sistemi monolitici si ha che tutte le funzioni di gestione delle risorse sono realizzate nel kernel e i diversi moduli sono molto legati tra di loro. Lo svantaggio è che, siccome tutti lavorano nella stessa area di memoria, un errore di un modulo può comportare il blocco dell'intero sistema. Il vantaggio è che un sistema monolitico è molto efficiente.
Esempi di sistemi monolitici:

- **MS-DOS**:
  era nato con l'idea di offrire il maggior numero di funzionalità nel minor spazio possibile. Non è suddiviso in moduli e non c'è una netta separazione tra interfacce e funzionalità. Gli applicativi accedono direttamente alle routine di sistema del BIOS, senza nessuna protezione, in quanto l'Intel 8088 (hardware su cui girava MS-DOS) non aveva protezione hardware tramite i bit mode.
- **UNIX**:
  è un sistema scarsamente stratificato (date le limitate funzionalità hardware di quando è stato progettato) ed è diviso in due parti: i programmi di sistema e il kernel (che incorpora tutto ciò che si trova tra le system call e l'hardware).

Approccio stratificato

L'approccio stratificato consiste nell'organizzare i vari moduli del sistema su più livelli in cui ogni livello può interagire con quelli sottostanti e fornisce funzionalità e servizi per quelli più esterni. Il livello più interno 0 è l'hardware, quello più esterno è l'interfaccia utente. Il vantaggio è nella semplicità di realizzazione e manutenzione, gli svantaggi sono la difficile caratterizzazione degli strati (uno strato può usare solo funzionalità di quello immediatamente inferiore) e la scarsa efficienza in quanto è necessario attraversare più strati per ogni system call.

Microkernel

I sistemi microkernel sono composto da un kernel minimalista che si occupa solo di gestire processi, memoria e comunicazione, mentre lascia il resto (applicativi, filesystem e device driver) al lato utente. In questo modo l'aggiunta di nuove funzionalità (dal lato utente) non modifica il kernel, è più facile da mantenere portare su nuove architetture, è più sicuro e affidabile in quanto poco codice viene eseguito in modalità kernel. Lo svantaggio è il possibile rallentamento causato dall'overhead tra modalità utente e modalità kernel. Esempi di microkernel sono Windows NT, Mach (incluso in Darwin), GNU Hurd.

Kernel modulari

I kernel modulari o Loadable Kernel Modules (LKMs) si organizzano in moduli che implementano componenti base con relative interfacce, possono comunicare tra di loro e possono essere caricati in memoria solo quando richiesto. Rispetto all'architettura a strati, quella a moduli è più flessibile perché non si ha l'obbligo di attraversare i diversi strati, sono facili da mantenere e da evolvere. Esempi sono Linux (periferiche e filesystem) e Solaris.

Sistemi ibridi

I sistemi ibridi combinano diversi approcci allo scopo di migliorare performance, manutenibilità e usabilità. Alcuni esempi sono:

- **Linux e Solaris** sono costituiti da kernel monolitici, ma prevedono la possibilità di nuovi moduli da caricare in runtime
- **Windows** è di base monolitico, ma conserva alcuni comportamenti di sistemi microkernel come il supporto a sottosistemi separati (personalità) eseguibili in modalità utente

macOS

```
|
+- Darwin: core open source di macOS
|  +- XNU: kernel ibrido "X is Not Unix"
|  |  +- Mach: microkernel per gestione di memoria, scheduling thread, RPC e IPC
|  |  +- BSD: sottosistema Unix-derived per gestione di filesystem, rete e API POSIX
|  |  +- IOKit: moduli dinamici per la gestione dei driver (estensioni kernel)
|  +- librerie C di basso livello: libc, dyld, ecc.
|  +- utility di sistema base: shell, tool UNIX, ecc.
+- framework grafico Cocoa: GUI, AppKit, UIKit
+- applicazioni di sistema: Finder, Safari, ecc.
+- servizi Apple proprietari: Spotlight, iCloud, ecc.
```

- Da Mac OS X (2001), il SO è costruito secondo la struttura sopra, con un kernel ibrido dotato di microkernel, sottosistema Unix e moduli dinamici per gestione dei driver.
- Le chiamate di procedura remota (RPC, Remote Procedure Calls) sono un meccanismo di comunicazione usato in informatica per permettere a un programma di eseguire una funzione o procedura che risiede su un altro computer o processo, come se fosse una chiamata locale.
- La comunicazione tra processi (IPC, Inter-Process Communication) è un insieme di meccanismi che permettono di gestire lo scambio di dati e informazioni tra due processi concorrenti (pipe, named pipe, message queue, shared memory, semafori, socket e signal).
- Le API POSIX (Portable Operating System Interface) sono un insieme di specifiche standard per le interfacce di programmazione dei sistemi operativi, progettate per garantire la portabilità delle applicazioni tra diversi sistemi Unix-like e altri sistemi compatibili. Grazie a POSIX, gli sviluppatori possono creare software che funziona su Linux, macOS, BSD e altri sistemi conformi, senza dover riscrivere codice specifico per ogni piattaforma.
- Il framework grafico Cocoa è l'insieme degli strumenti per creare interfacce grafiche nativamente in macOS. L'interfaccia utente predefinita di sistema si chiama Aqua che viene mantenuta sempre aggiornata e determina il look e il feel del sistema.

iOS

iOS si basa sullo stesso core di macOS chiamato Darwin integrato con:

- il framework grafico Cocoa Touch apposito per i dispositivi touch
- i media service per le applicazioni multimediali (grafica, video, audio)
- i core services per supporto al cloud computing e ai database

Android

Android è un sistema sviluppato dalla Open Handset Alliance guidata da Google, insieme a Asus, Htc. Intel, Motorola, Qualcomm, T-Mobile, Samsung e Nvidia. È un sistema a struttura stratificata basato su un kernel Linux modificato. I vari programmi e servizi vengono eseguiti sull'ART (Android Run Time), ovvero su macchina virtuale. Le app sono sviluppate in Java e tradotte in eseguibili per la ART virtual machine. È disponibile anche l'interfaccia JNI (Java Native Interface) per bypassare la ART e avere accesso diretto all'hardware. Nel sistema è incluso uno strato di astrazione dell'hardware HAL (Hardware Abstaction Layer).

Windows Subsystem for Linux - WSL

Grazie all'architettura ibrida di Windows è possibile eseguire sottosistemi in modalità utente in grado di emulare altri SO. Nel caso del WSL, viene avviata una istanza di Linux in grado di eseguire le applicazioni native di Linux grazie alla presenza dei servizi kernel LXCore e LXSS che traducono le system call Linux in system call Windows (es. fork in CreateProcess) e in mancanza di una corrispondenza esatta, fornisce una funzionalità equivalente.

Debugging

Il debugging è l'attività di individuazione e risoluzione dei bachi (bug) e di performace tuning. Il SO genera diversi file con i dati runtime dei processi attivi e dell'intero sistema:

- **file di log**: informazioni sugli errori durante l'esecuzione di un processo
- **core dump**: immagine della memoria di un processo al momento della sua terminazione anomala
- **crash dump**: immagine completa della memoria nell'istante in cui si verifica un crash, ovvero un guasto nel kernel, spesso si ricorrono a tecniche particolari (salvare l'immagine in un'area di memoria dedicata) per evitare di compromettere il filesystem in caso di kernel in stato inconsistente

Performance tuning

I problemi che coinvolgono le prestazioni del sistema sono considerati bachi e il performance tuning si occupa di individuarli e risolverli per ottimizzare le prestazioni ed eliminare i colli di bottiglia. Ciò viene fatto tramite strumenti per monitorare gli eventi in rilievo, le risorse in uso (es. comando top di Unix) e le system call più frequenti.

1.5 Gestione delle periferiche e delle risorse

Interfacciamento tramite driver, controller e interrupt

L'interfacciamento tra hardware e programmi applicativi avviene secondo il seguente schema:

applicativi ↔ kernel ↔ kernel I/O subsystem ↔ drivers ↔ controllers ↔ risorsa/dispositivo

- **controller**: componente hardware specifico per ogni risorsa che gestisce un buffer per l'input e l'output dei dati dalla risorsa all'elaboratore
- **driver**: componente software specifico di ogni risorsa che permette al kernel di interfacciarsi con un determinato controller e fornisce le indicazioni sullo scambio dei dati tra elaboratore e risorsa

Device driver

Un device driver (o controllore di periferica), è un componente software di basso livello utilizzato per comunicare con le periferiche connesse al computer.

- **driver di sistemi embedded**: tutto il software è un unico programma compilato e caricato in ROM, il driver non è altro che una routine del programma che si interfaccia con l'hardware da pilotare
- **driver in kernel monolitici** (es. Linux): i driver sono moduli compilati insieme al kernel, se si vuole collegare una periferica per cui non è presente un driver, è necessario aggiungerlo e ricompilare il kernel (dalle nuove versioni non è più richiesta la ricompilazione), questo porta al vantaggio che i driver già presenti nel kernel sono testati dai programmatori del sistema operativo offrendo maggiore efficienza e affidabilità
- **driver in kernel ibridi** (es. Windows): i driver sono file binari caricati dinamicamente dal kernel al momento del bisogno, questa scelta permette una maggiore compatibilità e flessibilità nella gestione delle periferiche in quanto è sufficiente che il produttore fornisca il driver per la propria periferica e si è certi di poterla utilizzare, lo svantaggio è che i driver potrebbero non essere ottimizzati ed efficienti, non essendo parte del kernel

Interrupt

La comunicazione tra controller e driver/kernel avviene tramite gli interrupt. Esiste un vettore con puntatori alle procedure da eseguire quando si riceve un interrupt (chiamate Interrupt Service Routines) e che gestiscono i flussi di dati tra risorse e sistema.

Input/Output

- **gestione sincrona**: la CPU aspetta fino a quando il dato non è pronto in quanto è la CPU che gestisce i trasferimenti
- **gestione asincrona**: la CPU delega al DMA lo spostamento dei dati e nel frattempo può eseguire altre operazioni

DMA, RDMA e GPU Direct Storage

Il DMA (Direct Memory Access) è un componente hardware che esegue il trasferimento di blocchi di dati dalla memoria secondaria o dal buffer locale di qualche dispositivo alla memoria principale. Viene gestito tramite interrupt e funziona come segue:

1. vengono specificati l'indirizzo, la dimensione dei dati sorgente e l'indirizzo di destinazione
2. il DMA si occupa del trasferimento, inviando un interrupt per ogni blocco trasferito, lasciando il processore libero di eseguire altri processi

Alcune varianti del DMA applicate a casi particolari:

- **RDMA o Remote Direct Memory Access** è un sistema che permette il trasferimento dei dati tra le RAM di diversi elaboratori collegati insieme, in ambito multicomputer con parallelizzazione
- **GPU Direct Storage** è un protocollo che prevede il trasferimento dei dati dalla memoria secondaria direttamente alla GPU, senza passare per la RAM, introdotto in Windows11 e inizialmente sviluppato per XBox.

1.6 Strutture dati del kernel

Strutture dati classiche

- **array**: semplice struttura dati in cui ogni elemento è direttamente accessibile tramite un indice. La memoria principale è costruita come un array in cui si accede alle celle tramite un indirizzo di memoria
- **liste concatenate**: possono contenere dati di diversa natura e dimensione e sono talvolta utilizzate direttamente dagli algoritmi del kernel, ad esempio nell'allocazione concatenata di file
- **stack**: struttura LIFO (Last In First Out) utilizzata ad esempio per le chiamate a funzione (RDA) e il cambio di contesto
- **code**: struttura FIFO (First In First Out) utilizzata per gestire i documenti inviati ad una stampante o i processi in attesa di ottenere l'accesso alla CPU
- **alberi**: strutture basate sulla relazione causale padre-figlio, in genere si utilizzano gli alberi binari di ricerca ad esempio in Linux per l'algoritmo di scheduling della CPU

Tabelle e funzioni hash

Una funzione hash è una funzione che trasforma ogni valore di chiave in un indirizzo o bucket.

- le funzioni di hash non sono iniettive per cui possono verificarsi **collisioni** se due chiavi distinte vengono mappate sullo stesso indirizzo, trovare funzioni che riducono al minimo le collisioni è molto complesso e molto dispendioso, per gestire le collisioni si utilizzano per ogni bucket delle catene o liste di trabocco (liste concatenate) per memorizzare le chiavi che collidono
- la **capacità** di un bucket è il numero di chiavi che può contenere (in caso di collisioni)
- l'**overflow** si verifica quando si tenta di inserire una chiave all'interno di un bucket già pieno
- l'**area di overflow** è un'area di memoria utilizzata per memorizzare i dati che hanno generato overflow
- l'**area primaria** è l'area di memoria con i bucket indirizzabili dalla tabella di hash

Per convertire una stringa alfanumerica s si sceglie una determinata funzione che mappa ogni carattere in un intero distinto s_i e, dopo aver scelto una base b, si associa il corrispettivo peso ad ogni carattere. Infine si sommano i risultati ottenuti:

$$k(s) = \sum_{i=0}^{n-1} s_i \cdot b^i$$

1.7 Protezione e sicurezza

Protezione delle risorse

Il SO deve controllare che i programmi accedano solo alle risorse (file, segmenti di memoria, CPU, ...) di cui hanno ottenuto l'apposita autorizzazione. Inoltre deve verificare che solo gli utenti autorizzati accedano alle risorse di cui hanno accesso.

Mode bit

Ogni operazione è associata ad un livello di protezione detto mode bit. Il mode bit è gestito via hardware e indica lo stato di protezione entro cui una determinata operazione può essere eseguita.

- **user mode** è uno stato di privilegio caratterizzato da un numero relativamente basso di privilegi verso la memoria, l'hardware e altre risorse
- **kernel mode** è lo stato di privilegio massimo riservato all'esecuzione del kernel. Il codice in linguaggio macchina eseguito in tale modalità ha accesso illimitato alla memoria, all'hardware e alle altre risorse

Se ci si trova in uno stato con alto livello di protezione (user mode) e si desidera eseguire un'operazione che richiede un livello inferiore (kernel mode), è necessario passare ad un mode bit inferiore tramite un processo di **trap**. Una volta completata l'operazione si effettua un processo di **return** per ripristinare il livello di protezione superiore. Ogni system call effettua il passaggio in modalità kernel; il ritorno dalla chiamata riporta il sistema in modalità utente.

Sicurezza - Protezione da attacchi informatici

Il SO deve implementare dei sistemi di difesa per contrastare attacchi interni ed esterni classificati in:

- **denial of service**: malfunzionamento dovuto ad un attacco informatico in cui si esauriscono deliberatamente le risorse di un sistema di calcolo che fornisce un servizio, fino a renderlo non più in grado di erogare il servizio
- **trojan**: programmi che hanno una funzione conosciuta legittima e una funzione dannosa nascosta
- **worm**: malware (software usato per disturbare le operazioni svolte da un computer, rubare informazioni sensibili, accedere a sistemi informatici privati, o mostrare pubblicità indesiderata) in grado di autoreplicarsi
- **virus**: porzioni di codice dannoso che si legano ad altri programmi del sistema per diffondersi, un virus può anche essere un malware

Gli attacchi sono svolti da:

- **hacker**: pirata informatico che si impegna nell'affrontare sfide intellettuali per aggirare o superare creativa mente le limitazioni di accesso ai sistemi per esplorare, divertirsi, apprendere, senza creare danni reali
- **cracker**: chi si ingegna per eludere blocchi imposti da qualsiasi software al fine di trarne guadagno

1.8 Virtualizzazione ed emulazione

Virtualizzazione in generale

La virtualizzazione/emulazione è una tecnica che permette di eseguire un sistema operativo come applicazione all'interno di un altro SO attraverso una macchina virtuale (VM). La VM crea un ambiente virtuale isolato che riproduce tipicamente il comportamento di una macchina fisica grazie all'assegnazione di risorse hardware (porzioni di disco rigido, RAM e CPU) gestite dall'hypervisor o Virtual Machine Manager (VMM).

Tra i vantaggi vi è il fatto di poter offrire contemporaneamente ed efficientemente a più utenti diversi ambienti operativi separati, attivabili su richiesta, senza sporcare il sistema fisico reale con partizionamenti del disco rigido oppure ricorrendo ad ambienti clusterizzati su sistemi server. Lo svantaggio è che, essendo le macchine virtuali isolate, non possono condividere le risorse nativamente, a meno che non si utilizzino determinati software come Virtio.

La virtualizzazione (per le CPU che la supportano) può essere vista come una terza modalità del mode bit con più privilegi dei processi utente in quanto lavora attivamente con le risorse assegnate, ma meno del kernel del SO ospitante in quanto deve sempre sottostare al kernel.

Virtualizzazione vs emulazione

L'emulazione è il processo per cui vengono riprodotte risorse hardware simulate che non corrispondo necessariamente all'architettura reale dell'elaboratore (es. emulazione x86 su ARM). Questo processo è più costoso in quanto le istruzioni da eseguire sono interpretate, ma permette di avere a disposizione molte architetture differenti.

La virtualizzazione consiste nel riservare parte delle risorse dell'hardware reale per l'ambiente virtualizzato. In questo modo non è necessario tradurre le istruzioni da una architettura ad un'altra, rendendo l'esecuzione più rapida, ma è possibile eseguire solo codice compilato per l'architettura reale in possesso.

1.9 Sistemi embedded

I sistemi embedded sono sistemi ubiquitari, ovvero presenti ovunque (elettrodomestici, auto, robot aziendali), sono eseguiti su sistemi rudimentali e hanno le seguenti caratteristiche:

- sistemi standard che girano su elaboratori general-purpose con implementate specifiche funzionalità
- sistemi special-purpose che girano su microprocessori
- Application Specific Integrated Circuit (ASIC) senza un vero SO

Le applicazioni prevedono dei requisiti stringenti sul tempo con cui determinate operazioni devono essere portate a termine, ovvero sono sistemi hard real-time.

1.10 Sistemi operativi open source

I sistemi operativi open source sono sistemi che vengono distribuiti sia in formato binario compilato, sia in codice sorgente. Non è necessario effettuare il reverse engineering per comprendere il funzionamento del sistema. Si forma una comunità di programmatori e aziende che contribuiscono allo sviluppo, al debugging, all'assistenza e al supporto gratuito agli utenti. Il codice più sicuro e i bug sono scoperti e risolti velocemente.

Richard Stallman, GNU e Linux

Richard Stallman nel 1984 da origine al progetto GNU (GNU is Not Unix), ovvero l'idea di ricreare un intero sistema operativo analogo a UNIX, ma libero, con licenza GLP (General Public Licence). La GPL indica che ogni componente software può essere utilizzata, modificata e distribuita mantenendo sempre la licenza GPL, ovvero non è possibile utilizzarlo in codice proprietario. Ad oggi il progetto GNU ha numeri strumenti compatibili con Unix, ma il kernel Hurt (GNU) è ancora in fase di sviluppo. Per i sistemi operativi liberi si utilizza il kernel Linux sviluppato da Linus Torvald usando strumenti di GNU, ma che non fa parte del progetto GNU.

Open source e software libero

Il software libero (GLP) indica che può essere utilizzato, modificato e distribuito liberamente, sempre sotto licenza GPL, ovvero non può mai diventare software proprietario. Il software open source è analogo al software libero, solo che può essere anche utilizzato in progetti proprietari.

Esempi di sistemi operativi open source / liberi

- **Linux**: open source con alcune distribuzioni libere e altre solo open source
- **Windows**: proprietario e closed-source
- **macOS**: ibrido in quanto basato su kernel Darwin open-source basato su UNIX BSD, con l'aggiunta di componenti proprietarie
- **UNIX BSD**: derivato da Unix, non è open source in quanto è richiesta la licenza, ma era inizialmente distribuito insieme al sorgente, esistono diverse distribuzioni (FreeBSD, NetBSD, ...)
- **MINIX**: sistema operativo sviluppato da Andrew Tanenbaum in supporto al suo corso universitario, ma impiegato a sua insaputa all'interno dell'Intel Management Engine, data la sua leggerezza

2 Ripasso Hardware

2.1 CPU

Struttura della CPU

La CPU o Control Proceccing Unit è l'unità di calcolo e controllo dell'elaboratore. È composta da:

- **Control Unit o CU**: unità di controllo che comprende il Program Counter e Instruction Register
- **Arithmetic Logic Unit o ALU**: unità che esegue operazioni matematiche e logiche
- **registri**: aree di memoria interne al chip, utilizzate nell'esecuzione delle istruzioni, come Program Counter (PC), Instruction Register (IR), Memory Address Register (MAR), Memory Data Register (MDR) e altri registri per gli operandi
- **bus dei segnali di controllo**: trasferimento unidirezionale di segnali di controllo dal processore alla memoria (read/write)
- **bus degli indirizzi**: trasferimento unidirezionale degli indirizzi dal processore alla memoria
- **bus dei dati**: trasferimento bidirezionale dei dati tra processore e memoria

Funzionamento

Alcune nomenclature sul funzionamento:

- **fetch-decode-execute**: ciclo di valutazione delle istruzioni in una CPU
- **data path**: processo che comprende il recupero dei dati dai registri, l'esecuzione dell'operazione da eseguire da parte della ALU e memorizzazione del risultato sul nuovo registro; il ciclo di data path corrisponde al ciclo di clock nelle architetture non parallelizzate
- **durata istruzione ISA**: ogni istruzione viene eseguita in un multiplo di cicli di data path

Prestazioni

- **tempo di esecuzione** dato dal T_{clock} tempo di un ciclo di clock o di data path, da N_i numero di istruzioni di tipo i e da CPI_i numero di cicli di clock per istruzioni di tipo: i

$$T_{\text{exec}} = T_{\text{clock}} \cdot \sum_{i=0}^{n} N_i \cdot CPI_i$$

- **MIPS o Mega Instructions Per Second** indica il numero di milioni di istruzioni eseguibili in un secondo non è molto preciso in quanto non tiene conto delle istruzioni offerte, della variazione di CPI per le diverse istruzioni e della velocità del buffer, si calcola come segue:

$$M_{\text{ega}}I_{\text{nstructions}}P_{\text{er}}S_{\text{econd}} = \frac{\text{freq. di clock}}{10^6 \cdot CPI}$$

- **MFLOPS o Mega Floating Point operations Per Second** indica il numero di operazioni in virgola mobile a 32 bit che vengono eseguite in un secondo
- **LINPACK benchmark** misura le operazioni in virgola mobile a 64 bit
- **Speedup** misura l'aumento di prestazioni in percentuale

$$\text{SpeedUp} = \frac{\text{Prest.}A - \text{Prest.}B}{\text{Prest.}B} = \frac{T_B - T_A}{T_A} \qquad \text{con} \quad \text{Prestazione} = 1/T_{\text{esecuzione}}$$

- **Legge di Amdhal** indica il miglioramento in funzione del ruolo del componente nel sistema con p tempo di utilizzo della parte migliorata e a accelerazione dovuta al miglioramento

$$T_{\text{finale}} = \frac{p}{a}\, T_{\text{iniziale}} + (1-p)\, T_{\text{iniziale}}$$

In generale per migliorare le prestazioni di una CPU è possibile agire:

- riducendo il numero di cicli per istruzioni ISA
- aumentando la frequenza di clock
- introducendo il parallelismo (a livello di istruzioni con pipeline e superscalari o a livello di core/CPU)

2.2 Memoria

Gerarchia delle memorie

La memoria si classifica in una struttura gerarchica in ordine decrescente di prestazioni e costo:

1. registri (volatile)
2. cache su più livelli (volatile)
3. memoria principale (volatile), es. RAM
4. memoria a stato solido es. DRAM, NVRAM, SSD, Flash
5. disco rigido
6. nastri magnetici

Gestione della memoria

Il SO è responsabile delle seguenti attività connesse alla gestione della memoria centrale:

1. tener traccia di quali parti della memoria sono attualmente usate e da chi (controllare l'accesso alla memoria e garantire la versione più recente di ogni file sul disco alle applicazioni che lo richiedono)
2. decidere quali (parti di) processi caricare in memoria quando vi è spazio disponibile
3. allocare e deallocare lo spazio di memoria secondo necessità

Caching

Il concetto di caching consiste nel creare una copia di dati su un supporto di memoria più veloce, ad esempio dal disco alla RAM, o dalla RAM alla cache. Quando il computer deve accedere alla memoria, per prima cosa controlla i supporti più veloci (registri) e, in assenza del dato, si passa al livello inferiore (cache, RAM, HDD/SSD). Per una migliore prestazione si cerca di avere almeno tra l'80 % e il 99 % degli accessi dalla cache.

Buffering

Il buffering è un processo per cui si utilizza un'area di memoria chiamata buffer per salvare temporaneamente i dati trasferiti tra dispositivi con velocità diverse (es. RAM e disco), i dati per dispositivi di output o le modifiche ai file prima del salvataggio su disco. In questo modo si migliora l'efficienza del trasferimento e salvataggio dei dati. Differisce dalla cache in quanto si trova nella RAM e funge da area in cui la CPU può memorizzare temporaneamente dati prima che essi vengano trasferiti o riprodotti.

Spooling

Lo spooling è una tecnica che consiste nel mettere in coda (spool) le richieste di I/O in un'area di memoria dedicata in modo che il sistema operativo le riesca a gestire in modo sequenziale. In questo modo è possibile gestire più attività di I/O contemporaneamente, migliorando l'efficienza del sistema.

I/O Interleaving

L'I/O Interleaving consiste nel trattare le operazioni di lettura o scrittura che coinvolgono più dispositivi di I/O in modo sequenziale, alternando le operazioni tra di essi anziché completarle su un dispositivo prima di passare al successivo. Questo permette al sistema di lavorare, ad esempio su settori di dischi diversi contemporaneamente, riducendo il tempo complessivo necessario per completare l'operazione. In generale, l'interleaving implica la disposizione non contigua dei dati per migliorare le prestazioni.

Memoria cache

La memoria cache è una piccola memoria molto veloce e costosa affiancata alla CPU per ridurre notevolmente il tempo impiegato per gli accessi alla RAM. In pratica vengono memorizzate delle copie delle pagine in RAM (dette linee di cache) in modo da poterci accedere velocemente senza sprecare cicli di clock per leggerle dalla RAM. Il processore tenta di leggere sempre i dati dalla cache, se il dato è effettivamente presente si ha un cache hit, altrimenti si ha un cache miss ed è necessario caricare in memoria la pagina con il dato richiesto.

Il principio di utilizzo della cache si basa su:

- **località spaziale**: ovvero se viene richiesto un dato, è probabile che verranno richiesti anche i dati limitrofi, infatti vengono caricate le intere pagine di RAM (delle linee di cache) e non i singoli dati
- **località temporale**: ovvero se in un istante si richiede un determinato dato, è probabile che verrà riutilizzato in un istante successivo, normalmente si usa la politica di rimpiazzo LRU

Spesso si utilizzano più livelli di cache per un migliore trade-off tra velocità e costi:

- **cache L1**: suddivisa in L1-I (istruzioni) e L1-D (dati) situata all'interno dei core (16KB to 128KB)
- **cache L2**: più lenta ed economica, cache condivisa tra i diversi core (256KB and 1MB)
- **cache L3**: più lenta ed economica, situata nella motherboard (2MB to 32MB)

Gestione del file system

Per gestione del file system significa gestione dell'organizzazione dei file nel disco, il SO è responsabile di:

1. creazione e cancellazione di file e directory
2. supporto alle funzioni elementari per la manipolazione di file e directory
3. associazione dei file ai dispositivi di memoria secondaria
4. backup di file su dispositivi stabili di memorizzazione

2.3 Concetti di sistemi e reti

Sistemi distribuiti

I sistemi distribuiti sono un insieme (eterogeneo) di calcolatori con memoria e clock indipendente (non condivisi) che sono connessi attraverso una rete di comunicazione di uno dei seguenti tipi (in base all'estensione):

1. Wide Area Network (WAN)
2. Metropolitan Area Network (MAN)
3. Local Area Network (LAN)
4. Personal Area Network (PAN)

La comunicazione avviene secondo un dato protocollo (TCP/IP è il più diffuso). Un sistema distribuito fornisce agli utenti l'accesso a varie risorse condivise di sistema in modo da consentire di accelerare l'elaborazione, aumentare la disponibilità di dati e migliorare l'affidabilità. In base al ruolo degli elaboratori nelle reti, queste vengono classificate in:

- **client-server**: i pc fungono da client che richiedono servizi al server
- **peer-to-peer (P2P)**: non esiste la distinzione tra client e server (es. Napster, eMule, servizi VoIP come Skype; bittorrent non è P2P puro perché necessita di server per connettersi alla rete)

Cloud computing

Il cloud computing indica l'insieme di piattaforme e tecnologie che permettono di archiviare file o di sviluppare/utilizzare programmi e applicazioni direttamente sui server di chi fornisce il servizio, anziché sul proprio dispositivo. Pertanto il cloud computing è una tecnica che permette la fruizione di risorse computazionali, di storage e di applicazioni come servizi di rete. Alcuni esempi di cloud computing sono Google Drive ed EC2 (Elastic Compute Cloud) di Amazon. Le tipologie di cloud computing sono:

1. **cloud pubblico**: disponibile attraverso internet a chiunque si abboni al servizio
2. **cloud privato**: gestito da un'azienda ad utilizzo interno
3. **cloud ibrido**: comprende componenti sia pubbliche che private
4. **SaaS** (Software as a Service): applicazioni fruibili via unternet (es. word processor, fogli di calcolo)
5. **PaaS** (Platform as a Service): ambiente software per usi applicativi via internet (server database)
6. **IaaS** (Infrastructure as a Service): server o storage accessibili via internet (spazio per backup)

2.4 Sistemi multiprocessore, multiscalare e parallelismo

Introduzione al parallelismo

Caratteristiche dei diversi tipi di parallelismo:

- **loosely coupled**: poche CPU indipendenti collegate a bassa velocità
- **strongly coupled**: tanti piccoli componenti (ALU, core, ...) collegati ad alta velocità
- **course grained**: il software parallelizzato è grande e non richiede interazioni con altri software
- **fine grained**: si parallelizzano piccole istruzioni che richiedono interazioni costanti
- **interconnessione statica**: se determinata a priori e non cambia nel tempo
- **interconnessione dinamica**: se si basa sulle necessità del momento, con apparecchi come switch

Sistemi superscalari - a parallelismo di istruzioni

Le architetture superscalari implementano un parallelismo a livello di istruzioni. Permettono di eseguire o iniziare a eseguire contemporaneamente più stati diversi (es. recupero operandi, decodifica, esecuzione, ...) di diverse istruzioni distribuite a diverse unità di elaborazione all'interno del singolo chip.

Sistemi Hyper-Threading o multithreading

Nei sistemi Hyper-Threading o multithreading, i singoli core dispongono di due unità di elaborazione in grado di eseguire contemporaneamente due thread in parallelo sullo stesso core.

Sistemi multicore

Un'architettura multicore si raggruppano diverse unità di calcolo (core) in un singolo chip. È più efficienti, perché le comunicazioni, sul singolo chip, sono più veloci e un chip multicore usa molta meno potenza di diversi chip single core. I diversi core nel singolo chip hanno un livello di cache condiviso (L2 o superiore).

Sistemi con coprocessori

Un'architettura con coprocessori consiste nell'affiancare un'altra unità di elaborazione esterna al processore per svolgere compiti specifici come elaborazione grafica (GPU), processi di rete (schede di rete), crittografia (criptoprocessori).

Sistemi multiprocessore

Un'architettura multiprocessore è costituita da più processori su una unica scheda madre che condividono la stessa memoria. Bisogna fare attenzione nella gestione del traffico dei dati tra i diversi processori e all'assegnazione delle risorse. L'elaborazione può essere:

- **asimmetrica**: ad ogni processore viene assegnato un compito specifico e un processore principale sovrintende all'intero sistema
- **simmetrica**: ogni processore è abilitato all'esecuzione di tutte le operazione del sistema

Sistemi multicomputer

Un'architettura multicomputer è una struttura costituita da più elaboratori (ciascuno indipendente con la propria CPU e memoria) collegati insieme, di solito condividono la stessa memoria secondaria.. La gestione del traffico dei dati tra i diversi processori è particolarmente ostico da gestire, però hanno il vantaggio di essere più facili ed economici da implementare rispetto ai sistemi multiprocessore. Esistono alcuni indici di classificazione dei multicomputer:

- **grado o fanout**: numero di interconnessioni di ogni unità, serve per calcolare il fault tollerance
- **diametro**: distanza massima tra due nodi qualsiasi della rete
- **dimensionalità**: numero di assi diversi su cui si sviluppa la rete
- **scalabilità**: proporzionalità tra il numero di processori e le prestazioni della rete

Alcuni esempi di interconnessioni:

- **stella**: tutti i nodi sono connessi ad un nodo centrale, diametro = 2, dimensione = 0
- **albero**: struttura ad albero, diametro = 2h, dimensione = 0
- **interconnessione completa**: tutti i nodi sono connessi tra di loro, diametro = 1, dimensione = 0
- **anello**: i nodi sono posizionati su una circonferenza: diametro = $2P/2$, dimensione = 1
- **griglia**: nodi posizionati in una griglia, diametro = $2(l-1)$, dimensione = 2
- **toroide2D**: griglia con interconnessioni tra nodi esterni, diametro = l, dimensione = 2
- **cubo**: nodi disposti su un cubo, diametro = 3, dimensione = 3
- **ipercubo**: nodi disposti su un ipercubo, buon compromesso di scalabilità in quanto il diametro aumenta logaritmicamente in base al numero di processori, diametro = 4, dimensione = 4
- **toroide3D**: nodi disposti su una griglia tridimensionale, in quanto è un buon compromesso tra diametro della rete e numero di connessioni, diametro = ..., dimensione = 3

Sistemi cluster

Un cluster è un particolare tipo di multicomputer con particolare attenzione alle interconnessioni e alla coordinazione delle diverse unità. Per usufruire delle potenzialità dei cluster in ambito High Performance Computer (HPC), è necessario che i processi siano scritti per essere parallelizzati. In base alla gestione delle interconnessioni si differenziano in:

- **clustering asimmetrico**: un calcolatore rimane nello stato di attesa attiva mentre gli altri eseguono le applicazioni, il calcolatore in stand by controlla gli altri nodi e si attiva nel caso di malfunzionamenti
- **clustering simmetrico**: tutti i nodi eseguono le applicazioni e si controllano reciprocamente, attraverso ad esempio il protocollo NUMA (Non Uniform Memory Acccess system)

Speedup per aumento delle CPU

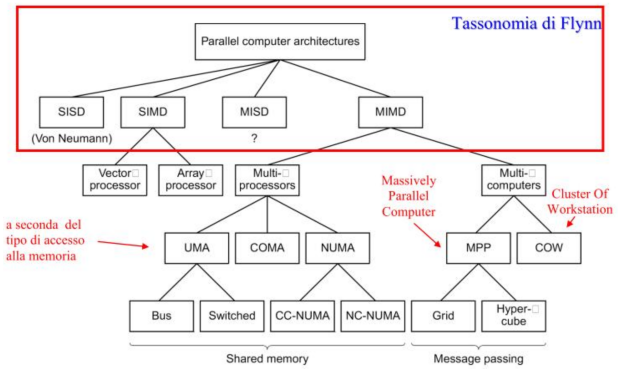
Il tempo di esecuzione di un frammento di codice (parzialmente o totalmente parallelizzato) in un'architettura in grado di eseguire operazioni parallele viene:

$$S_{\text{peedUp}} = \frac{T_{\text{seq}}}{T_{\text{par}}} = \frac{n}{1-(n-1)f} \qquad T_{\text{par}} = f \cdot T_{\text{seq}} \frac{(1-f)T_{\text{seq}}}{n}$$

T_{seq} = tempo exec. con 1 CPU
T_{par} = tempo exec. con n CPU
n = numero CPU
f = frazione sequenziale del codice

Classificazione tassonomica di Flynn (1972)

- **SISD**: single instruction sequence, single data sequence
- **SIMD**: single instruction sequence, multi data sequence
- **MISD**: multi instruction sequence, single data sequence
- **MIMD**: multi instruction sequence, multi data sequence



2.5 Coprocessori: GPU, TPU, FPGA

Evoluzione della GPU per general purpose calculation GP-GPU

All'inizio le GPU erano specifiche per eseguire calcoli relativi alla parte grafica (colore dei pixel, ...) usando apposite unità di elaborazione grafica come i pixel shader. Nel 2006 grazie all'architettura CUDA nella scheda grafica GeForce 8800 GTX di NVIDIA anche le schede grafiche sono equipaggiate con apposite ALI in modo da poter eseguire aritmetica a singola e doppia precisione, accesso in lettura e scrittura alla memoria centrale e gestione diretta delle memorie cache come la shared memory.

Calcolo ibrido GPU+CPU

Progettare cluster per il calcolo ad alte prestazioni basato sulla struttura ibrida CPU+GPU è più complesso, ma permette di rendere molto efficiente in termini di tempo l'esecuzione di molte applicazioni di carattere scientifico, matematico, fisico o ingegneristico. Infatti la CPU consiste di pochi core ottimizzati per l'elaborazione sequenziale, mentre una GPU ha un'architettura massicciamente parallela che consiste di migliaia di core molto efficiente progettati per trattare task multipli simultaneamente. Alla CPU si lascia la parte di codice in cui vengono gestite le risorse hardware e software del sistema, mentre alla GPU si lascia la parte puramente matematica di analisi dati.

Tensor Processing Unit - TPU

Google ha progettato e realizzato un processore proprietario denominato TPU e dedicato alle architetture delle reti neurali per l'apprendimento approfondito (Deep Learning Neural Networks). È stato chiamato così perché sfrutta la libreria TensorFlow sviluppata da Google per DeepLearning.

La TPU è un ASIC (Application Specific Integrated Circuit), ovvero un processore creato per svolgere specifici compiti in maniera estremamente ottimizzata. I processori TPU, progettati per l'apprendimento automatico, offrono prestazioni migliori per watt e numero di transistor rispetto alle CPU e GPU grazie all'alta specificità, a una tolleranza maggiore agli errori computazionali e a circuiti più semplici.

Field Programmable Gate Array - FPGA

I FPGA sono array di componenti hardware programmabili singolarmente detti Configurable Logic Blocks (CLB) interconnessi tra di loro. In questo modo si può ottenere implementazioni hardware di algoritmi e reti logiche aggiornabili senza dover però sostenere gli elevati costi di fabbricazione di un chip. Grazie all'implementazione hardware hanno un elevato speedup.

3 Concorrenza

3.1 Processi e Thread

Un processo è un'istanza attiva di un programma che comprende tutte le informazioni necessarie per il suo funzionamento, risiede quindi in RAM. Il programma è un insieme di codice e strutture eseguibile da un elaboratore ed è passivo. Un programma può avere più istanze di esecuzione, ovvero più processi.

Contesto di un processo o Process Control Block - PCB

Un processo è composto dal codice con le istruzioni da eseguire e dal contesto di esecuzione (o Process Control Block - PCB). Il PCB è una struttura dati contenente tutte le informazioni per il sistema operativo per gestire un processo durante il suo ciclo di vita, è salvato in RAM ed è composto da:

- identificatore (PID)
- stato del processo
- contesto della CPU (registri e program counter)
- informazioni sulla memoria
    - base e limite della memoria dedicata al processo
    - tabella delle pagine o dei segmenti
    - heap (memoria dinamica) e stack (variabili locali e RDA)
    - initialized e uninitialized data segment
- informazioni sulle risorse
- informazioni di scheduling
- informazioni di comunicazione tra processi

Multiprogrammazione e ruolo del SO nella gestione dei processi

La multiprogrammazione consiste nell'avere contemporaneamente più job (o processi) salvati in RAM in modo da poterli alternare tra loro nel caso in cui uno di essi sia in "pausa" mentre attende una risorsa. Il SO si deve occupare di:

1. creazione e cancellazione di processi (utente e di sistema)
2. sospensione e riattivazione di processi (attraverso scheduling della CPU e swapping)
3. fornire meccanismi per la sincronizzazione di processi, la comunicazione fra processi e la gestione del deadlock (evitare che processi si blocchino in attesa di risorse)

Stati di un processo

- **new**: quando un processo è stato appena avviato
- **ready**: quando ha i dati, ma è in coda di esecuzione
- **running**: quando è in esecuzione
- **waiting**: quando sta aspettando le risorse
- **terminated**: quando ha terminato l'esecuzione

Il passaggio di un processo dallo stato di ready a quello di running è gestito dall'algoritmo di scheduler dispatch. Quando un processo passa da running a ready, il PCB di tale processo viene salvato in RAM per poter essere ripristinato la volta successiva che torna in esecuzione. Lo stato terminated si verifica per terminazione normale e restituzione del controllo al SO, terminazione anomala (es. eccezioni, ...), uso scorretto di risorse o chiamata a istruzioni non valide (trap).

Scheduling dei processi pt. 1

Lo scheduling dei processi è un meccanismo messo in atto per massimizzare l'utilizzo della CPU, che consiste nel passare rapidamente dall'esecuzione di un processo al successivo, garantendo il time sharing. È svolto dal CPU scheduler (modulo del sistema operativo) che sfrutta tre code:

1. **job queue**: insieme di tutti i processi presenti nel sistema
2. **ready queue**: insieme dei processi ready, prossimi all'esecuzione, che risiedono in memoria centrale
3. **code dei dispositivi**: insieme dei processi in attesa di un dispositivo di I/O

Thread

Un thread (filo) è l'unità base di esecuzione all'interno della CPU. Ogni processo deve essere suddiviso in uno o più thread per essere eseguito. Per gestirne l'esecuzione ogni thread possiede il proprio TCB, simile al PCB ma specifico per i thread. Inoltre condivide codice, dati e risorse con gli altri thread dello stesso processo, ma ha istanze di esecuzione e program counter distinti. I thread vengono utilizzati nelle architetture multicore perché permettono il parallelismo e la concorrenza in quanto è possibile eseguire in parallelo più thread, poiché il sistema può assegnare thread diversi a diverse unità di calcolo.

3.2 Concorrenza vs Parallelismo

Tabella di confronto

Concorrenza	**Parallelismo**
Atto di eseguire più calcoli nello stesso intervallo di tempo	Atto di eseguire più calcoli simultaneamente
Task multipli vengono eseguiti negli stessi intervalli di tempo, senza ordine specifico particolare	Task multipli vengono eseguiti contemporaneamente in sistemi multi-CPU o multicore
L'esecuzione del task sembra contemporanea, ma non lo è	Viene permesso da strutture hardware apposite
L'effetto è dovuto al time-slicing della CPU, ovvero allo scheduler(context switching) che dedica l'unità di calcolo ai vari task per unità di tempo infinitesimali	Si ottiene trasformando un flusso di esecuzione sequenziale in uno parallelo

Tipi di Parallelismo

Ci sono due tipi di parallelismo:

- Data parallelism: la stessa operazione viene eseguita contemporaneamente sui dati, questi vengono suddivisi tra i vari thread in modo che non ci siano conflitti
- Task parallelism: parallelizzare del codice tra thread diversi

3.3 Diagramma Temporale

Tempi di esecuzione

Grafo di Precedenza

Sistemi di Processi

Massimo Grado di Parallelismo

Interferenza e Determinatezza

3.4 Risorse

3.5 Deadlock

Definizione di Deadlock

Definizione di Livelock

Gestione dei Deadlock

Ripristino dei Deadlock

3.6 Prevenzione del Deadlock

Allocazione Globale

Allocazione Gerarchica

Algoritmo del Banchiere

3.7 Grafo delle Risorse

Individuazione dei Deadlock

3.8 Race Condition e Sezioni Critiche

3.9 Semafori

Semafori Contatore

Semafori Binari - Lock Mutex

Semafori Privati

Busy Waiting

Deadlock e Starvation

3.10 Monitor

Problema dei 5 Filosofi

Allocazione Risorse

3.11 Memory Barrier

4 Scheduling CPU

5 Gestione Memoria Principale

6 Gestione File

7 Affidabilità