Appunti per l'Esame di Stato
Ingegnere Junior - Settore Informazione

Paolo Pietrelli

18 luglio 2025

Indice

Elenco delle figure

Listings

Capitolo 1

Sistemi Operativi

Un **sistema operativo (SO)** è un software di sistema che gestisce le risorse hardware e software di un computer e fornisce servizi comuni per i programmi del computer e per l'utente. È l'interfaccia tra l'hardware e l'utente/applicazioni. La sua importanza risiede nell'astrazione dell'hardware, nella gestione efficiente delle risorse e nell'esecuzione controllata dei programmi.

1.1 Struttura e Organizzazione del Sistema Operativo

Un sistema operativo è un'entità complessa, ma può essere scomposto in componenti modulari che cooperano per fornire un ambiente funzionale per l'esecuzione dei programmi.

1.1.1 Componenti Principali

I principali componenti di un sistema operativo includono:

- **Kernel**: Il cuore del SO, responsabile della gestione dei processi (creazione, scheduling, terminazione, comunicazione interprocesso), della memoria (allocazione, protezione, gestione della memoria virtuale), dei file system (gestione dei file e delle directory, allocazione dello spazio su disco) e dell'I/O (gestione dei dispositivi di input/output, driver).
- **Gestore dei Processi (Process Management)**: Si occupa della creazione, terminazione, sospensione e ripristino dei processi, e della gestione dei loro stati (pronto, in esecuzione, in attesa).
- **Gestore della Memoria (Memory Management)**: Responsabile dell'allocazione e deallocazione della memoria ai processi, della gestione della memoria virtuale (paginazione, segmentazione) e della protezione della memoria per evitare interferenze tra i processi.
- **File System Management**: Organizza e gestisce i dati su dispositivi di archiviazione, controllando l'accesso e la protezione dei file e delle directory.
- **Gestore I/O (I/O Management)**: Fornisce un'interfaccia standardizzata per interagire con i dispositivi hardware (stampanti, tastiere, dischi) tramite driver specifici.
- **Network Management**: Gestisce le comunicazioni di rete e i protocolli di comunicazione.
- **Security and Protection**: Implementa meccanismi per proteggere le risorse del sistema e i dati degli utenti da accessi non autorizzati o malfunzionamenti.

- **Interfaccia Utente (User Interface)**: Può essere una GUI (Graphical User Interface) con elementi visivi o una CLI (Command Line Interface) basata su testo, permettendo all'utente di interagire con il sistema.

1.1.2 Modelli di Sistemi Operativi

I sistemi operativi possono essere strutturati secondo diversi modelli architetturali:

- **Monolitici**: Tutti i servizi del SO risiedono nello stesso spazio di indirizzamento (kernel space).
  - **Vantaggi**: Alta performance grazie al minimo overhead di comunicazione.
  - **Svantaggi**: Difficili da debuggare, poco flessibili, un crash di un componente può bloccare l'intero sistema.
  - **Esempio**: Linux, Unix (tradizionali).
- **Layered (a Strati)**: Il SO è diviso in strati, ognuno dei quali offre servizi allo strato superiore e utilizza servizi dallo strato inferiore.
  - **Vantaggi**: Modularità, facilità di debug e manutenzione.
  - **Svantaggi**: Performance ridotte a causa dell'overhead di comunicazione tra strati.
  - **Esempio**: THE (Dijkstra).
- **Microkernel**: Solo i servizi essenziali (gestione processi, gestione memoria base, comunicazione interprocesso) risiedono nel kernel (microkernel). Altri servizi (file system, driver, network) sono implementati come processi utente (server).
  - **Vantaggi**: Modularità, robustezza (un crash di un server non blocca il sistema), flessibilità.
  - **Svantaggi**: Performance potenzialmente più basse a causa di più cambi di contesto.
  - **Esempio**: Mach (base per macOS), QNX.
- **Modulari (o Ibridi)**: Un approccio intermedio che combina le migliori caratteristiche dei modelli monolitici e microkernel. Permettono il caricamento dinamico dei moduli kernel (es. driver) senza richiedere un riavvio completo del sistema.
  - **Esempio**: Versioni moderne di Linux, Windows.

1.2 Scheduling della CPU

Lo **scheduling della CPU** è l'attività di selezionare quale processo, tra quelli pronti per l'esecuzione, deve essere assegnato alla CPU in un dato momento. Ha un impatto cruciale sulle performance complessive del sistema.

1.2.1 Principali Problematiche dello Scheduling

Lo scheduling deve affrontare diverse sfide e problematiche per bilanciare l'efficienza e l'equità:

- **Ottimizzazione degli obiettivi**: Bilanciare metriche contrastanti come massimizzare il throughput (numero di processi completati per unità di tempo), minimizzare il tempo di risposta (tempo tra richiesta e prima risposta), minimizzare il tempo di attesa e garantire l'equità tra i processi.

- **Contesto Switching (Cambio di Contesto)**: L'overhead di tempo necessario per salvare lo stato di un processo in esecuzione e caricare lo stato del prossimo processo da eseguire. Questo tempo è "sprecato" e non contribuisce all'esecuzione del lavoro utile.
- **Starvation (Inedia)**: Un processo a bassa priorità potrebbe non essere mai eseguito se processi a priorità più alta arrivano continuamente e monopolizzano la CPU.
- **Deadlock**: Sebbene sia una problematica più ampia della gestione della concorrenza, situazioni di deadlock possono emergere in sistemi con scheduling se le risorse non sono gestite correttamente, bloccando indefinitamente i processi.
- **Dipendenza dall'I/O**: Processi che trascorrono molto tempo in attesa di operazioni di I/O (I/O-bound) possono rendere inefficiente lo scheduling se la CPU rimane inattiva mentre attende il completamento di tali operazioni.

1.2.2 Esempi di Algoritmi di Scheduling

Diversi algoritmi sono stati sviluppati per affrontare le problematiche dello scheduling, ognuno con i propri compromessi:

- **First-Come, First-Served (FCFS)**:
  - **Descrizione**: Non preemptive. I processi vengono eseguiti nell'ordine in cui arrivano.
  - **Vantaggi**: Semplice da implementare.
  - **Svantaggi**: "Effetto convoglio", dove un processo lungo blocca tutti gli altri, aumentando il tempo medio di attesa.
- **Shortest-Job-First (SJF)**:
  - **Descrizione**: Può essere preemptive (Shortest-Remaining-Time-First, SRTF) o non preemptive. Il processo con il tempo di esecuzione stimato più breve viene eseguito per primo.
  - **Vantaggi**: Ottimale per minimizzare il tempo medio di attesa.
  - **Svantaggi**: Difficile conoscere a priori la durata esatta di un job; può portare a starvation per processi lunghi.
- **Priority Scheduling**:
  - **Descrizione**: Può essere preemptive o non preemptive. Ai processi viene assegnata una priorità numerica o concettuale, e viene eseguito quello con la priorità più alta.
  - **Vantaggi**: Prioritizza lavori critici o importanti.
  - **Svantaggi**: Può portare a starvation per processi a bassa priorità; una soluzione è l'aging (aumentare la priorità di un processo che aspetta da troppo tempo).
- **Round Robin (RR)**:
  - **Descrizione**: Preemptive. Ogni processo ottiene una piccola porzione di tempo di CPU (quantum). Se non finisce entro il quantum, viene preempted e messo in coda per il prossimo turno.
  - **Vantaggi**: Equo, garantisce un buon tempo di risposta per processi interattivi.

  - **Svantaggi**: L'overhead del context switching aumenta se il quantum è troppo piccolo; le performance degradano se il quantum è troppo grande (tende a FCFS).

- **Multilevel Queue Scheduling**:

  - **Descrizione**: I processi sono divisi in diverse code, ognuna con il proprio algoritmo di scheduling (es. una coda per processi foreground con RR, una per processi background con FCFS).

- **Multilevel Feedback Queue Scheduling**:

  - **Descrizione**: Permette ai processi di muoversi tra le code in base al loro comportamento (es. un processo che usa molto la CPU scende di priorità, un processo che aspetta molto sale). È uno degli schedulatori più generali e complessi.

Capitolo 2

Basi di Dati

Le **Basi di Dati (Database)** sono collezioni organizzate di dati che permettono un'efficiente memorizzazione, recupero e gestione delle informazioni. Sono fondamentali per la maggior parte delle applicazioni software moderne.

2.1 Modello Concettuale Entità-Relazione (ER)

Il **Modello Entità-Relazione (ER)** è uno strumento concettuale di alto livello utilizzato nella fase iniziale della progettazione di database. Permette di rappresentare il mondo reale in termini di "entità" (oggetti o concetti di interesse) e "relazioni" (associazioni tra le entità). L'obiettivo è fornire una rappresentazione intuitiva e facilmente comprensibile della struttura dei dati prima di tradurla in un modello logico.

2.1.1 Componenti Principali del Modello ER

- **Entità**: Rappresentano "cose" o "oggetti" del mondo reale su cui si vogliono memorizzare informazioni. Possono essere concrete (es. Persona, Prodotto) o astratte (es. Corso, Ordine). Nel diagramma ER, le entità sono generalmente rappresentate con un rettangolo.
- **Attributi**: Sono le proprietà o caratteristiche che descrivono un'entità o una relazione. Ad esempio, un'entità "Studente" può avere attributi come "Nome", "Cognome", "Matricola", "DataNascita". Nel diagramma ER, gli attributi sono spesso rappresentati con un ovale.
  - **Attributi Semplici/Composti**: Un attributo semplice non può essere scomposto (es. "Età"), mentre uno composto è formato da più attributi (es. "Indirizzo" composto da "Via", "Civico", "Città").
  - **Attributi Mono-valore/Multi-valore**: Mono-valore ha un singolo valore per istanza (es. "DataNascita"), multi-valore può avere più valori (es. "NumeriDiTelefono").
  - **Attributi Derivati**: Il loro valore può essere calcolato da altri attributi (es. "Età" derivata da "DataNascita").
  - **Chiave (Key Attribute)**: Un attributo (o un insieme di attributi) che identifica in modo univoco ogni istanza di un'entità. Viene tipicamente sottolineato nel diagramma ER.
- **Relazioni**: Rappresentano associazioni logiche tra due o più entità. Ad esempio, un "Docente" "insegna" a un "Corso". Nel diagramma ER, le relazioni sono rappresentate con un rombo.

- **Cardinalità delle Relazioni**: Definisce il numero di istanze di un'entità che possono essere associate a un'istanza dell'altra entità nella relazione. Le cardinalità più comuni sono:
  - **Uno a Uno (1:1)**: Una istanza di entità A è associata a una e una sola istanza di entità B, e viceversa (es. "Persona" - "ha" - "Patente").
  - **Uno a Molti (1:N)**: Una istanza di entità A è associata a zero o molte istanze di entità B, ma un'istanza di B è associata a una e una sola istanza di A (es. "Dipartimento" - "comprende" - "Docente").
  - **Molti a Molti (N:M)**: Una istanza di entità A è associata a zero o molte istanze di entità B, e viceversa (es. "Studente" - "frequenta" - "Corso").
- **Partecipazione (o Dipendenza)**: Indica se l'esistenza di un'istanza di un'entità dipende dalla sua partecipazione a una relazione.
  - **Totale (o Obbligatoria)**: Ogni istanza dell'entità deve partecipare alla relazione (indicata da una doppia linea).
  - **Parziale (o Opzionale)**: Un'istanza dell'entità può partecipare o meno alla relazione (indicata da una singola linea).

2.2 Progettazione Logica: Normalizzazione e Forme Normali

La **normalizzazione** è un processo sistematico di organizzazione dei dati in un database relazionale. Il suo scopo è ridurre la ridondanza dei dati, eliminare le anomalie di aggiornamento (inserimento, cancellazione, modifica) e migliorare l'integrità e la coerenza dei dati. La normalizzazione si basa su una serie di regole chiamate "forme normali".

2.2.1 Obiettivi della Normalizzazione

- **Riduzione della Ridondanza**: Evitare la duplicazione inutile dei dati, che spreca spazio e può portare a incongruenze.
- **Miglioramento dell'Integrità dei Dati**: Assicurare che i dati siano accurati e consistenti.
- **Prevenzione delle Anomalie**:
  - **Anomalia di Inserimento**: Impossibilità di inserire un'informazione a meno che non si inseriscano anche altre informazioni non correlate.
  - **Anomalia di Cancellazione**: La cancellazione di un dato comporta la perdita accidentale di altre informazioni non desiderate.
  - **Anomalia di Aggiornamento**: La modifica di un dato ripetuto richiede l'aggiornamento di più occorrenze, con rischio di inconsistenza se non tutte vengono aggiornate.
- **Flessibilità e Manutenibilità**: Rendere il database più facile da modificare ed estendere.

2.2.2 Principali Forme Normali

Le forme normali sono una serie di regole progressive; per essere in una forma normale N, una relazione deve soddisfare i requisiti della forma normale N-1. Le più comuni e rilevanti per la maggior parte delle applicazioni sono la Prima, Seconda e Terza Forma Normale.

Prima Forma Normale (1NF)

Una relazione è in 1NF se e solo se:

- Tutti gli attributi sono **atomici** (indivisibili). Non ci sono attributi con valori multipli o attributi composti che non sono stati scomposti.
- Ogni record (riga) nella relazione è **unico**. Questo implica che deve esistere una chiave primaria.

Esempio di Violazione: Una colonna "NumeriDiTelefono" che contiene più numeri per un'unica riga.

Seconda Forma Normale (2NF)

Una relazione è in 2NF se e solo se:

- È in **1NF**.
- Tutti gli attributi non-chiave dipendono **completamente** dalla chiave primaria. Non ci sono dipendenze parziali, il che significa che nessun attributo non-chiave dipende solo da una parte di una chiave primaria composta.

Esempio di Violazione: In una tabella '(IDCorso, IDStudente, NomeCorso, Voto)', se '(IDCorso, IDStudente)' è la chiave primaria, e 'NomeCorso' dipende solo da 'IDCorso' (e non da 'IDStudente'), allora 'NomeCorso' è parzialmente dipendente e viola la 2NF.

Terza Forma Normale (3NF)

Una relazione è in 3NF se e solo se:

- È in **2NF**.
- Non contiene **dipendenze transitive**. Ovvero, nessun attributo non-chiave dipende da un altro attributo non-chiave (anziché dipendere direttamente dalla chiave primaria).

Esempio di Violazione: In una tabella '(IDImpiegato, NomeImpiegato, Dipartimento, CapoDipartimento)', se 'IDImpiegato' è la chiave primaria e 'CapoDipartimento' dipende da 'Dipartimento' (che a sua volta dipende da 'IDImpiegato'), si ha una dipendenza transitiva.

2.2.3 Linguaggio SQL

Il **Structured Query Language (SQL)** è il linguaggio standard per la gestione dei sistemi di gestione di database relazionali (RDBMS). Permette di definire, manipolare e controllare i dati.

Categorie di Comandi SQL

- **Data Definition Language (DDL)**: Utilizzato per definire e modificare la struttura del database.
  - **CREATE**: Crea database, tabelle, viste, indici, ecc. (es. 'CREATE TABLE Studenti (...)').
  - **ALTER**: Modifica la struttura di oggetti esistenti (es. 'ALTER TABLE Studenti ADD COLUMN Età INT').
  - **DROP**: Cancella oggetti dal database (es. 'DROP TABLE Studenti').
- **Data Manipulation Language (DML)**: Utilizzato per manipolare i dati all'interno delle tabelle.
  - **SELECT**: Recupera dati da una o più tabelle. È la query più usata.
  - **INSERT**: Aggiunge nuove righe a una tabella.
  - **UPDATE**: Modifica righe esistenti in una tabella.
  - **DELETE**: Rimuove righe da una tabella.
- **Data Control Language (DCL)**: Utilizzato per gestire i permessi di accesso ai dati.
  - **GRANT**: Concede privilegi agli utenti.
  - **REVOKE**: Rimuove privilegi dagli utenti.
- **Transaction Control Language (TCL)**: Utilizzato per gestire le transazioni (gruppi di operazioni che devono essere eseguite atomicamente).
  - **COMMIT**: Salva le modifiche di una transazione.
  - **ROLLBACK**: Annulla le modifiche di una transazione.

Elementi Comuni delle Query SQL (SELECT)

La query `SELECT` è la più potente e versatile, permettendo di interrogare il database.

- **SELECT**: Specifica le colonne da recuperare.
  - `SELECT colonna1, colonna2`
  - `SELECT *` (tutte le colonne)
  - `SELECT DISTINCT colonna` (solo valori unici)
- **FROM**: Specifica la tabella (o le tabelle) da cui recuperare i dati.
- **WHERE**: Filtra le righe in base a una condizione specificata.
  - `WHERE condizione` (es. 'WHERE Età $>$ 18')
  - Operatori: '=', '$>$', '$<$', '$>=$', '$<=$', '$<>$', 'LIKE' (per pattern matching), 'IN', 'BETWEEN', 'IS NULL'.
- **JOIN**: Combina righe da due o più tabelle basandosi su una colonna correlata.
  - **INNER JOIN**: Restituisce solo le righe che hanno corrispondenze in entrambe le tabelle.

  - **LEFT (OUTER) JOIN**: Restituisce tutte le righe dalla tabella sinistra e le righe corrispondenti dalla tabella destra (con NULL se non ci sono corrispondenze).
  - **RIGHT (OUTER) JOIN**: Simile al LEFT JOIN, ma per la tabella destra.
  - **FULL (OUTER) JOIN**: Restituisce tutte le righe quando c'è una corrispondenza in una delle due tabelle.

- **GROUP BY**: Raggruppa le righe che hanno gli stessi valori in una o più colonne, spesso usato con funzioni di aggregazione.
- **HAVING**: Filtra i gruppi creati da 'GROUP BY' in base a una condizione. Si usa con le funzioni di aggregazione.
- **ORDER BY**: Ordina il set di risultati in base a una o più colonne (ASC per ascendente, DESC per discendente).
- **Funzioni di Aggregazione**: Calcolano un singolo valore da un insieme di valori (es. 'COUNT()', 'SUM()', 'AVG()', 'MAX()', 'MIN()').