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

Paolo Pietrelli

16 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.