Sistemi Operativi - Teoria

Andrea Comar

January 22, 2025

Contents

Part I
Hardware e Architettura degli elaboratori

Part II
Introduzione ai Sistemi Operativi

1 Definizione

Un sistema operativo è un programma che agisce come intermediario tra utente programmatore e hardware. Coordina l'uso dell'hardware tra i vari programmi e utenti. Il suo obbiettivo è quello di realizzare una *macchina astratta* che implementi funzionalità di alto livello.

- **assegnatore di risorse**: alloca le risorse in modo efficiente
- **Programma di controllo**: controlla l'esecuzione dei programmi per evitare errori

Le componenti di un sistema di calcolo sono:

- **Hardware**: fornisce le risorse computazionali di base CPU, memoria, I/O
- **Sistema Operativo**: controlla e coordina le risorse hardware SO, driver, utility
- **Programmi di sistema**: programmi indipendenti dall'applicazione che forniscono servizi al SO (compilatori, editor,...)
- **Programmi applicativi**: programmi che definiscono il modo in cui le risorse del sistema sono usate per risolvere problemi computazionali dell'utente
- **Utenti**: persone, macchine, altri calcolatori.

2 Storia

2.1 Esordi

I primi calcolatori erano molto ingombranti e funzionavano unicamente da console, con un solo utente alla volta. Utilizzavano schede di collegamento poi IØsu nastro perforato o schede perforate

2.2 II Generazione: Transistor e sistemi batch

2.3 III Generazione: Sistemi multiprogrammati

La CPU era limitata dai sistemi a nastro e dalla *logica uniprogrammata*, ovvero da

2.4 Sistemi time-sharing

Part III
Componenti, servizi e strutture dei S.O

3 Componenti comuni dei sistemi

3.1 Gestione dei processi

processo: programma in esecuzione, il quale necessita di determinate risorse (tempo cpu, ram, file, dispositivi I/O ecc...)

Per quanto riguarda la gestione dei processi il sistema operativo è responsabile di:

- creazione e cancellazione;
- sospensione e resume;
- fornire meccanismi di sincronizzazione e comunicazione processi
- evitare, prevenire i deadlock

3.2 Gestione della memoria principale

Per quanto riguarda la memoria possiamo dire che:

- è un grande array di *words* identificate da un indirizzo preciso.
- è un deposito di dati rapidamente accessibile
- è un deposito *volatile*, ovvero perde il contenuto in caso di *system failure.*

Il sistema operativo è responsabile di:

- tenere traccia delle locazioni di memoria allocate e libere
- decidere quale processo deve essere caricato in memoria quando c'è spazio limitato
- allocare e deallocare spazio di memoria come richiesto

3.3 Gestione della memoria secondaria

La memoria secondaria è una memoria di supporto a quella principale, capiente e soprattutto non volatile. Il sistema operativo si occupa della *gestione dello spazio libero*, dell'*allocazione dello spazio* e dello *scheduling* dei vari dischi di memoria.

3.4 Gestione del sistema di I/O

Ovvero:

- sistemi di caching,buffering,spooling
- interfaccia generale dei gestori di dispositivi(device driver)
- i driver per ogni specifico dispositivo hw (controller)

3.5 Gestion dei File

file: è una collezione di informazioni correlate, definite dal creatore. Comunemente rapppresentano dati e programmi (sia sorgenti che *oggetti*(= eseguibili)) Il sistema operativo è responsabile di:

- creazione e cancellazione di file
- creazione e cancellazione di directory
- supporto di primitive per manipolazione di file e directory
- allocazione dei file nella memoria secondaria e salvataggio dati su supporti non volatili

3.6 Sistemi di protezione

Protezione: è il metodo per controllare l'accesso alle risorse del sistema da parte di utenti, programmi e processi. Un buon meccanismo di protezione deve:

- distinguere tra uso autorizzato e non
- fornire un modo per specificare i controlli da imporre
- forzare gli utenti e i processi a sottostare ai controlli richiesti

3.7 Networking(Sistemi distribuiti)

Sistema distribuito: è una collezione di processori che non condividono memoria o clock. Ogni processore ha una *memoria propria*. I processori del sistema sono connessi attraverso una *rete di comunicazione*. Fornisce agli utenti l'accesso a diverse risorse di sistema. La condivisione di una risorsa condivisa permette di avere dei miglioramenti prestazionali in termini computazionali, di quantità di dati e di affidabilità.

3.8 Interprete dei comandi

Il sistema operativo riceve molti comandi attraverso dei *control statement*, i quali servono a controllare il comportamento del sistema nei casi visti precedentemente (processi, I/O, memorie, file system, ...). Il programma che legge e interpreta i comandi di controllo ha diversi nomi in base a sistema operativo e tipologia. La sua funzione è di ricevere un comando, eseguirlo e ripetere

- inteprete schede di controllo (sistemi batch)
- interprete della linea di comando (DOS, Windows)
- shell (UNIX)
- interfaccia grafica: Finder (MacOS), Explorer (Windows), gnome-session (Unix)

4 Servizi dei Sistemi operativi

I sistemi operativi si occupano di diversi servizi:

- esecuzione programmi: caricamento in memoria ed esecuzione
- operazioni di I/O, il Sistema operativo determina come condurle in quanto non possono farlo gli utenti
- manipolazione del file system
- comunicazione tra processi in esecuzione, mediante *memoria condivisa* o *passaggio di messaggi*
- indivudazione e risoluzione di errori nelle vaarie parti hw (cpu, ram ...) oppure nei programmi degli utenti

Vi sono poi delle funzionalità addizionali atte a migliorare l'efficienza del sistema:

- allocazione risorse a più utenti o processi
- accounting, ovvero tracciare chi e come usa le risorse (per statistica o rendicontazione)
- protezione, assicurando che tutti gli accessi alle risorse siano controllati

5 Chiamate di Sistema (System Calls)

5.1 Definizione

Le chiamate di sistema formano l'interfaccia tra un programm in esecuzione e il sistema operativo. Generalmente sono disponibili o come istruzioni assembler

speciali, oppure alcuni linguaggi specifici per OS permettono direttamente di effettuarne. Per passare parametri tra un programma e il sistema operativo possiamo:

- utilizzare i *registri*
- passare, mediante i registrim un indirizzo a una tabella di memoria contenente i parametri
- utilizzare lo *stack*, mediante push dei parametri da parte del programma e pop da parte del sistema operativo

5.2 Tipi di chiamate di sistema

- **Controllo dei processi**: creazione, terminazione di processi
- **Gestione dei file**:
- **Gestione dei dispositivi**: richiesta lettura, scrittura, apertura, chiusura
- **Informazioni sul sistema**: informazione sul sistema come data, hw e sw installato, ecc
- **Comunicazione**: creazione, cancellazione di comunicazioni tra processi

6 Programmi di sistema

I programmi di sistema forniscono un ambiente per lo sviluppo e l'esecuzione dei programmi. Si dividono in: gestione file, modifiche file, informazioni sullo stato del sistema e dell'utente, supporto dei linguaggi di programmazione, caricamente ed esecuzione dei programmi, comunicazioni, altri programmi come compilatori, elaboratori testi, interpreti di comandi, ecc.
Gran parte di ciò che un utente vede di un sistema operaitivo è definito dai programmi di sistema e non dalle reali chiamate di sistema.

7 Struttura dei Sistemi Operativi

7.1 approccio semplice

- MS-DOS: massime funzionalità nel minore spazio, NON diviso in moduli, interfaccie e livelli funzionali non ben separati
- UNIX originale: strutturazione debole, distinzione principale tra programmi di sistema e *kernel*

Kernel: consiste in tutto ciò che sta tra system call e hardware.
implementa il file system, lo scheduling della cpu, la gestione della memoria e altre funzioni del sistema operativo.

7.2 approccio stratificato

Il S.O. viene suddiviso in diversi strati detti livelli, partendo dall'hardware (livello 0) all'interfaccia utente. Secondo la *modularità* ogni livello utilizza operazioni e servizi degli strati inferiori, sul quale è costruito.

THE OS: fu il primo esempio di stratificazione, in 6 livelli: hardware, CPU scheduling, memory management, operator-console device driver, buffering for input e output devices, user programs.

8 Macchine virtuali

8.1 definizione

macchina virtuale: fornisce un'interfaccia identica all'hardware sottostante a diversi ambienti di esecuzione.
A ogni processo guest viene fornito una copia virtuale dell'host.
Il gestore della macchina virtuale (VMM o hypervisor) impiega le risorse del calcolatore fisico per creare macchine virtuali

- scheduling CPU crea illusione che ogni processo abbia la sua CPU dedicata
- gestione ram crea illusione di ram dedicata
- stampanti virtuali mediante spooling
- lo spazio del disco può essere impiegato per "dischi virtuali"

Fornisce protezione delle risorse del sistema in quanto ogni VM è isolata, a discapito della condivisione diretta delle risorse. Ottima per la simulazione di altri sistemi operativi, in quanto limita possibilità di far danni. L'implementazione è complessa in quanto la duplicazione non è facile. L'approccio è seguito in molti sistemi.

8.2 Exokernel

exokernel: (è un approccio alla virtualizzazione che cerca di minimizzare il livello di astrazione tra il software e l'hardware.) ogni macchina virtuale vede solo un sottoinsieme delle risorse complessive della macchina, ogni vm con proprio OS richiede le risorse all'exokernel ne traccia l'utilizzo. Questo approccio semplifica l'uso delle risorse allocate in quanto l'exokernel deve unicamente tenere **separati** i domini di allocazione

9 meccanismi e politiche

I kernel tradizionali (monolitici) sono poco flessibili.

- **meccanismi**: determinano **come** deve essere fatto (es. assegnazione dell'esecuzione a un processo)
- **politiche**: determinano **cosa** deve essere fatto (es. scegliere quale processo attivare)

Questa separazione permette flessibilità in caso di cambiamenti di politiche. (estremizzazione: kernel fornisce meccanismi, politiche implementate in user space).

10 Sistemi con Microkernel

microkernel: quando il kernel è ridotto all'osso e fornisce soltanto i meccanismi di

- comunicazione tra i processi
- minima gestione della memoria e dei processi
- gestione hardware basso livello (driver)

Tutto il resto viene gestito a livello di spazio utente. è meno efficiente del kernel monolitico, ma è molto flessibile, immediatamente scalabile in ambiente di rete.

11 Esempi

- Esempio stratificato microkernel: MacOS X
- Esempio stratificato non microkernel Windows Vista

Part IV
Processi e Thread

12 Concetto di processo

programma: entità statica che risiede in un file su disco.

processo: è la parte **dinamica** di un programma generata da un'esecuzione del programma stesso. Ogni processo ha il suo program counter di tipo logico.

job: è l'insieme dei processi (o il singolo processo) generati da una singola linea di comando che occorrono in pipeline.

multiprogrammazione: tecnica in cui più processi risiedono in memoria principale allo stesso tempo, in modo da tenere occupata la CPU.

time-sharing: tecnica in cui la CPU è condivisa tra più processi, in modo che ognuno abbia l'impressione di avere la CPU dedicata.

13 Operazioni sui processi

13.1 switch di contesto

Passaggio della cpu da un processo all'altro. Il tempo di context-switch porta a un certo overhead, in quanto include:

- salvataggio dello stato del processo corrente
- caricamento dello stato del processo successivo
- cambiamento dello spazio di indirizzamento

(appunti 2024-10-11) In un sistema ad alto parallelismo può rappresentare un collo di bottiglia, il tempo di di switch dipende dall'hardware delll'architettura. Uno switch di contesto può venire quando il processo termina oppure a seguito di una system call bloccante o prelazione.

13.2 Creazione di un processo:

Un processo può essere creato al boot del sistema, all'esecuzione di una system call (es. fork()), oppure su richiesta da parte dell'utente o all'inizio di un job batch. La generazione dei processi induce a una gerarchia detta albero di processi.

- **padre:** processo che crea un altro processo
- **figlio:** processo creato da un altro processo

L'esecuzione del processo padre e figlio sono concorrenti, il padre generalmente viene sospeso fino al completamento del padre. La condivisione delle risorse può essere totale (condividono le stesse risorse) oppure parziale (sottoinsieme) oppure assente (nessuna risorsa condivisa). Per quanto riguarda lo spazio indirizzi possono essere una duplicazione (es. fork()) oppure i figli caricano un programma (CreateProcess()).

13.3 Terminazione di un processo:

Un processo può terminare volontariamente, sia normalmente sia tramite una **exit**, ovvero un errore. L'output viene raccolto dal padre che era in attesa (wait()). Oppure può terminare involotariamente, ad esempio per errore fatale (superamento limiti, operazioni illegali). Altrimenti la terminazione può essere causata da un altro processo (kill()) o da parte del kernel. Il sistema operativo dealloca le risorse del processo terminato.

13.4 Gerarchia dei processi:

I processi figli e genitore possono rimanere collegati, creando delle famiglie chiamate **gruppi di processi**. I gruppi sono utili per la comunicazione. In UNIX tutti i processi discendono da **init**, il processo di sistema. Se un parent muore, il figlio viene ereditato da init. Un processo non può diseredare il padre.

13.5 Stati dei processi:

Durante l'esecuzione il processo un processo può trovarsi in diversi stati:

- **new**: il processo è in fase di creazione
- **ready**: il processo è pronto per l'esecuzione
- **running**: il processo è in esecuzione
- **waiting**: il processo è in attesa di un evento (es. I/O)
- **terminated**: il processo ha terminato l'esecuzione

Il passaggio da uno stato all'altro avviene in seguito a interruzioni, richieste di risorse non disponibili, selezione da parte di uno scheduler. (diagramma degli stati)

13.6 Process Control Block - PCB

Contiene le informazioni associate a un processo, tra cui stato, dati identificativi, program counter, registri cpu, informazioni per lo scheduling, informazione(...)

13.7 Gli scheduler

scheduler: è un componente del sistema operativo che si occupa di decidere quale processo passa da una coda all'altra. Esistono diversi tipi di scheduler:

- **scheduler di lungo termine**: decide quali processi devono finire in ready queue (detto anche job scheduler)
- **short-term scheduler**: decide quale processo in ready deve essere eseguito, assegnando la cpu. Poiché viene invocato molto frequentemente deve essere molto veloce.
- **medium-term scheduler**: decide quali processi devono essere rimossi dalla memoria principale per liberare spazio, abbassando il livello di multiprogrammazione. è detto anche swap scheduler.

13.8 Modelli di esecuzione dei processi:

Esecuzioni kernel separata dai processi utente, esecuzione kernel all'interno dei processi. (..)

14 Threads

15 Scheduling dei processi

Part V
Scheduling CPU

16 Introduzione

L'introduzione della multiprogrammazione ha portato alla necessità di gestire l'allocazione della CPU tra i vari processi. Ogni processo è caratterizzato da un **ciclo Burst CPU-I/O** che consiste in una sequenza di periodi di esecuzione di CPU e attesa di I/O.

Part VI
Esercizi