Architettura di un sistema operativo

Struttura di un S.O.

- Sistemi monolitici
- Sistemi “a struttura semplice”
- Sistemi a livelli
- Virtual Machine
- Sistemi basati su kernel
- Sistemi con microkernel
- Sistemi con exokernel

Sistemi monolitici

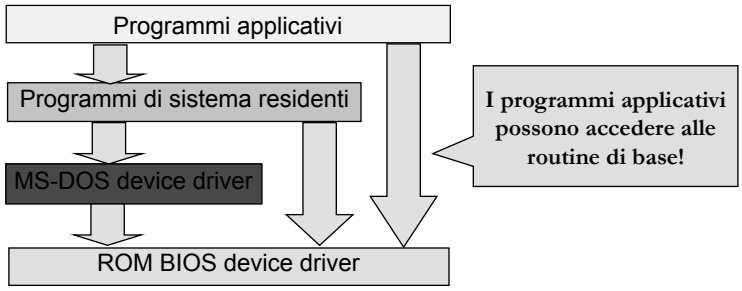
- No gerarchia
  - Unico strato SW tra utente e HW
    - Componenti tutti allo stesso livello
    - Insieme di procedure che possono chiamarsi vicendevolmente
- Svantaggi
  - Codice dipendente dall'architettura HW era distribuito su tutto il S.O.
  - Test e debug difficile

Sistemi a struttura semplice

- Minima organizzazione gerarchica
  - Definizione dei livelli della gerarchia molto flessibile
  - Strutturazione mira a ridurre costi di sviluppo e manutenzione
- Es.: MS-DOS, UNIX originale

MS-DOS

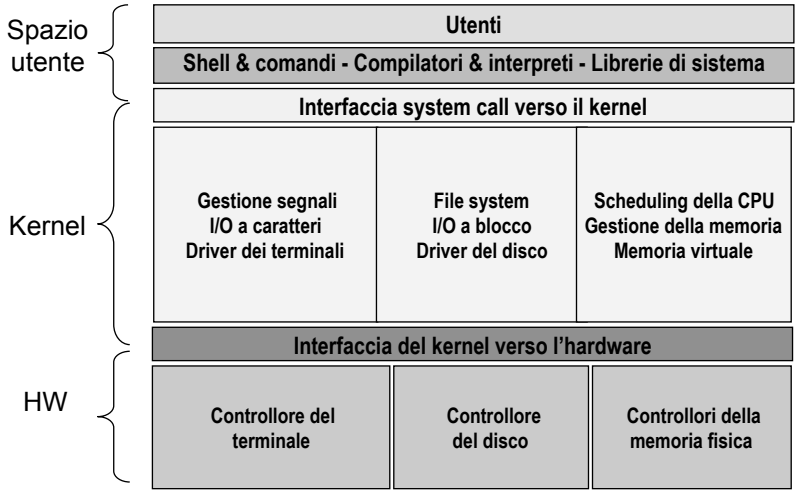
- Pensato per fornire il maggior numero di funzionalità nel minimo spazio
  - Non suddiviso in moduli
  - Possiede un minimo di struttura, ma le interfacce e i livelli di funzionalità non sono ben definiti
  - Non prevede dual mode (perché Intel 8088 non lo forniva)



UNIX (originale)

- Struttura base limitata a causa delle limitate funzionalità HW
  - Programmi di sistema
  - Kernel
    - Tutto ciò che sta tra il livello dell'interfaccia delle system call e l'HW
    - Fornisce
      - File system
      - Scheduling della CPU
      - Gestione della memoria
      - Altre funzioni

UNIX (originale)



Sistemi a livelli

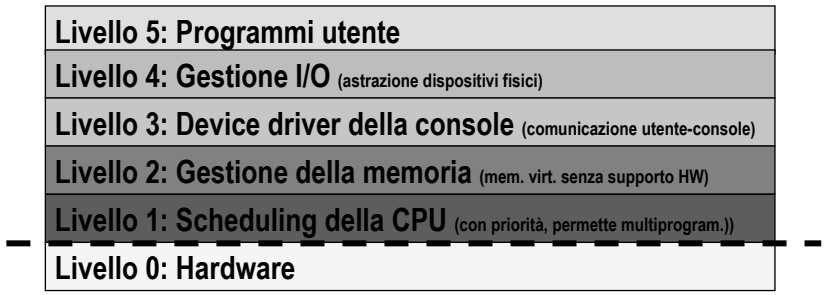
- Servizi organizzati per livelli gerarchici
  - Interfaccia utente (livello più alto) → … → HW (più basso)
  - Ogni livello:
    - può usare solo funzioni fornite dai livelli inferiori
    - definisce precisamente il tipo di servizio e l'interfaccia verso il livello superiore nascondendone l'implementazione

Sistemi a livelli

- Vantaggi:
  - Modularità: facilita messa a punto e manutenzione del sistema
- Svantaggi:
  - Difficile definire appropriatamente gli strati
  - Minor efficienza: ogni strato aggiunge overhead alle system call
  - Minor portabilità: funzionalità dipendenti dall'architettura sono sparse sui vari livelli
- Es.: THE, MULTICS, OS/2

THE (Dijkstra 1968)

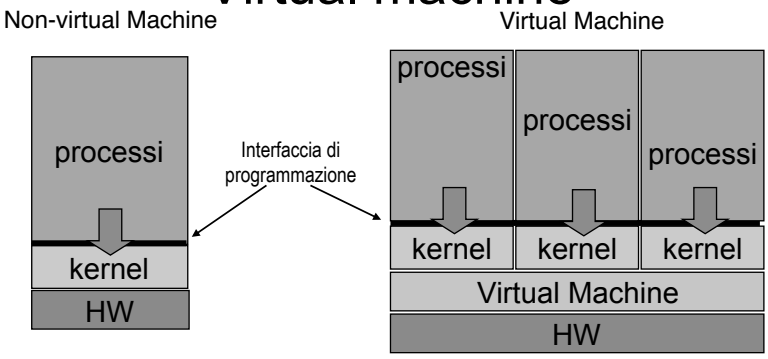
- Sistema operativo accademico per sistemi batch
- Primo esempio di sistema a livelli
  - Insieme di processi cooperanti sincronizzati tramite semafori



Virtual machine

- Estremizzazione dell'approccio a livelli (IBM VM 1972)
  - Pensato per offrire un sistema timesharing "multiplo"
- HW e S.O. trattati come hardware
  - Una VM fornisce un'interfaccia identica all'HW sottostante
  - Il S.O. esegue sopra la VM
  - La VM dà l'illusione di processi multipli, ciascuno in esecuzione sul proprio HW
- Possibilità di presenza di più S.O.

Virtual machine



- Concetto chiave: separazione di
  - Multiprogrammazione (Virtual Machine)
  - Presentazione (S.O.)

Virtual machine

- Vantaggi
  - Protezione completa del sistema: ogni VM è isolata dalle altre
  - Più di un S.O. sulla stessa macchina host
  - Ottimizzazione delle risorse
    - La stessa macchina può ospitare quello che senza VM doveva essere eseguito su macchine separate
  - Ottime per lo sviluppo di S.O.
  - Buona portabilità

Virtual machine

- Svantaggi
  - Problemi di prestazioni
  - Necessità di gestire dual mode virtuale
    - Il sistema di gestione delle VM esegue in kernel mode, ma la VM esegue in user mode
  - No condivisione: ogni VM è isolata dalle altre
    - Soluzione:
      - condivisione un volume del file system
      - Definire una rete virtuale tra VM via SW

Virtual machine

- Concetto usato ancora oggi, anche se in contesti diversi e con certi vincoli
- Esempi:
  - Esecuzione di programmi MS-DOS in Windows
    - Emulazione di 8086 (1MB memoria)
  - Esecuzione “contemporanea” di Linux e Windows (vMware, VirtualBox, …)
  - Java Virtual Machine (JVM)

Sistemi basati su kernel

- Due soli livelli: Servizi kernel e servizi non-kernel
  - Alcune funzionalità fuori dal kernel (es.: File system)
  - Es.: Implementazioni “moderne” di UNIX
- Vantaggi
  - Vantaggi dei sistemi a livelli, ma senza averne troppi
- Svantaggi
  - Non così generale come un sistema a livelli
  - Nessuna regola organizzativa per parti del S.O. fuori dal kernel
  - Kernel complesso tende a diventare monolitico

Microkernel

- Come i kernel, soltanto più semplici
  - Microkernel include soltanto i servizi essenziali (gestione memoria, IPC, sincronizzazione tra processi, …)
  - Altre funzioni convenzionali (scheduler, file system, rete, …) fuori dal kernel (spazio utente)
- Esempi
  - Mach, WindowsNT, Chorus, L4, Amoeba
- Tendenza degli anni ’80
- Poco usati oggi

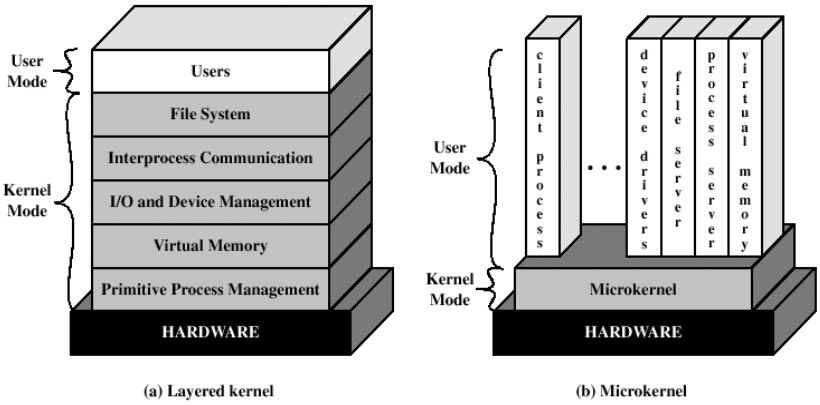
Microkernel

- Tutti i servizi del S.O. sono realizzati come processi utente (client)
- Il client chiama un processo servitore (server) per usufruire di un servizio
- Il server, dopo l’esecuzione, restituisce il risultato al client
- Il kernel si occupa solo della gestione della comunicazione tra client e server
- Il modello si presta bene per S.O. distribuiti
- S.O. moderni realizzano tipicamente alcuni servizi in questo modo

Microkernel

- Vantaggi
  - Vantaggi dei sistemi a kernel
  - Codice per servizi essenziali altamente ottimizzato
  - Facilmente estendibile
    - Aggiunta di servizi non richiede modifica del kernel
  - Maggior sicurezza e affidabilità …
    - … visto che la maggior parte dei servizi eseguono in modalità utente
- Svantaggi
  - Tendenza a diventare "kernel"
  - Potenziali basse prestazioni

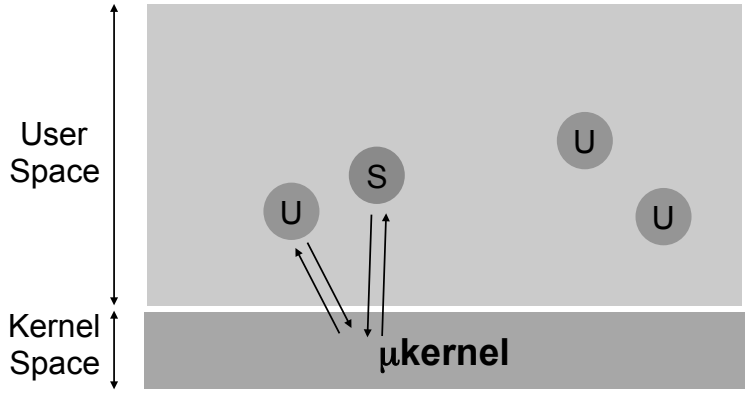
Architettura di base



Operazioni

- I servizi esterni al microkernel sono implementati come processi server
- L'interazione avviene attraverso lo scambio di messaggi che transitano nel microkernel
  - microkernel è essenzialmente un "gestore di messaggi" tra moduli esterni
  - In pratica, una architettura client/server su una singola macchina

Schema operativo



Problemi

- Ogni operazione richiede un passaggio attraverso il kernel in termini di scambio di messaggi
  - Potenzialmente inefficiente
- Problema chiave: bilanciamento tra funzioni kernel e non kernel
  - Quanto “micro”?

Esempio

- L3 (L4) microkernel [Liedtke, 1992-1995]
  - Definisce un insieme di primitive minime
    - Gestione “primitiva” della memoria
    - Thread & IPC
    - I/O & gestione interrupt
  - Complessità
    - 12KB codice (Linux 2.6 ha 2.5 milioni di linee di codice)
    - 7 system call
  - Piattaforme
    - Ix86, Alpha, ARM, UltraSparc, …

L4: Gestione memoria

- Solo gestione dello spazio di indirizzamento
  - Realizza solo il mapping tra pagina virtuale e frame fisico
  - Tre primitive
    - Grant: un processo fornisce pagine ad un altro processo (spazio di indirizzamento)
    - Map: un processo condivide con un altro il suo spazio di indirizzamento
    - Flush: un processo richiede spazio di indirizzamento precedentemente ceduto con grant o map
  - Spazio di indirizzamento costruito ricorsivamente con queste tre primitive
    - Inizialmente, un unico spazio detenuto da un processo "base"
- Gestione della memoria e paginazione gestite da processi fuori dal kernel

L4: Thread

- Thread = unità di esecuzione all'interno di uno spazio di indirizzamento
- Thread possiede
  - Set di registri
  - Spazio di indirizzamento
  - Gestore di page fault
  - Parametri di scheduling
  - …
- microkernel mantiene l'associazione tra thread e spazio di indirizzamento

L4 mkernel : IPC

- microkernel fornisce supporto per la comunicazione tra spazi di indirizzamento
  - Basata su scambio di messaggi
- Altre astrazioni di IPC costruite sopra lo schema a messaggi
- Messaggi usati anche per I/O e interrupt
  - Porte di I/O = viste come parte spazio di indirizzamento
  - Interrupt = messaggi dall'HW
  - Kernel riconosce interrupt ma non li gestisce
    - Avverte il processo, che ottiene un grant dello spazio corrispondente

L4: Prestazioni

- Es: L4Linux (linux su L4)
  - Prestazioni tipiche di system call
    - 2/3 volte più lento di Linux nativo
    - 5/20 volte più veloce di altre implementazioni Linux su microkernel
- Vantaggi?
  - Flessibilità
  - Estendibilità

L4: Approfondimenti

- J. Liedtke, “On µ-kernel construction”, Proceedings of ACM Symposium on Operating System Principles, 1995.
- A. Au, G. Heiser, “L4 user manual”, 1999.
- http://en.wikipedia.org/wiki/L4_microkernel_family

Exokernel

- S.O. estendibile sviluppato al MIT
- Idea: portare all’estremo l’idea di microkernel
  - In pratica, nessuna funzione classica è implementata nello spazio del kernel
  - L’unica funzione del kernel consiste nel “multiplexare” le risorse HW tra processi
- Exokernel può essere visto come l’estensione dell’architettura RISC al livello del sistema operativo

Exokernel

- Funzioni base di S.O. (generico)
  - HW multiplexing
  - Astrazione dell'HW
    - Impedisce l'applicazione di ottimizzazioni possibili in casi specifici (es. RTOS, SO embedded, ...)
    - Scoraggia modifiche a implementazioni esistenti
    - Nuove astrazioni possono essere costruite solo sopra quelle esistenti
    - Costosa in termini di performance
- Exokernel elimina l'astrazione adottando una gestione delle risorse application-level

Exokernel

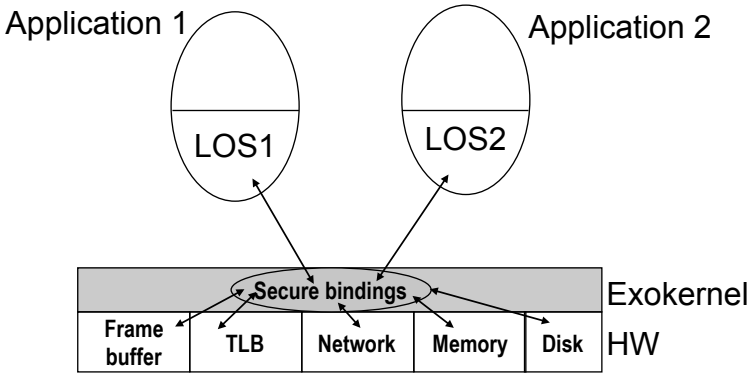
- Exokernel implementa solo l'HW multiplexing
- Tutte le astrazioni (API, politiche di gestione) devono essere fornite da librerie al livello utente
  - library operating systems (LOS)
  - a basso livello possibile per motivi di efficienza
- Separazione tra protezione e gestione
  - Exokernel: protegge risorse e ne permette la condivisione
  - Applicazioni: gestiscono le proprie risorse
    - Un errore su un'applicazione ha conseguenze solo su questa

Exokernel

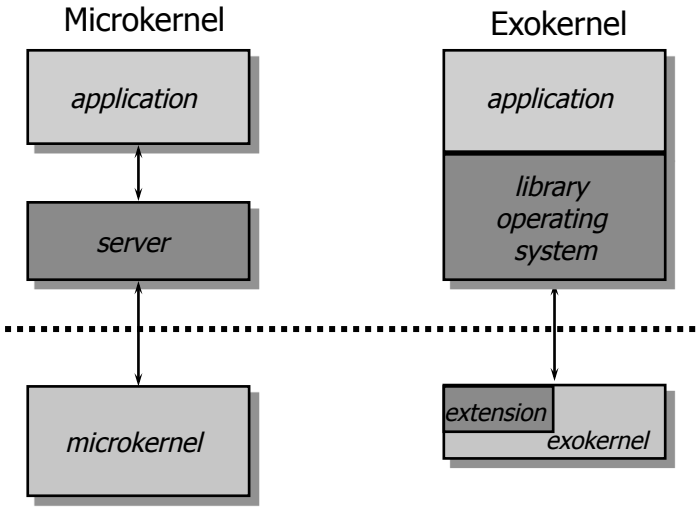
- Servizi forniti
  - Controllo del possesso delle risorse
  - Controllo dell'utilizzo delle risorse o dei binding point
  - Revoca dell'accesso alle risorse
- Modalità di fornitura "sicura" delle risorse
  - Secure binding: permette alle LOS di collegare le risorse alle applicazioni e di gestire gli eventi in modo sicuro
  - Visible resource revocation: permette alle LOS di partecipare attivamente alla revoca delle risorse
  - Abort protocols: permette all'exokernel di rompere i binding di applicazioni "non cooperative"

Exokernel

- Architettura base:



Microkernel vs. Exokernel



Exokernel: prestazioni

- Molto efficienti, ma:
  - Quanto deve essere "tecnico" un programmatore per scrivere codice che possa sfruttare le potenzialità dell'architettura?
    - Problema del modello di programmazione
  - Portabilità del S.O.?
    - Limitata, ma meno problematica
- Non implementato in nessun S.O. commerciale
  - Es.: Aegis, Xok
- Ottimi per applicazioni specifiche
  - Es: web server

Approfondimenti

- “On microkernel construction”
  - Liedtke, SOSP’95
- “The performance of u-kernel-based systems”
  - Hartig et al., SOSP’97
- “Exokernel: an OS for application-level resource management”
  - Engler et al., SOSP’95
- “Application performance and flexibility on exokernel systems”
  - Kaashoek et al., SOSP’97