DALLE ARCHITETTURE AI SISTEMI OPERATIVI

Danilo Croce

Ottobre 2024

COS'È UN SISTEMA OPERATIVO?

- Un **moderno calcolatore** è tipicamente formato da:
  - uno o più processori;
  - memoria centrale;
  - dischi;
    stampanti e altre periferiche di I/O.
- I dettagli di basso livello sono **molto complessi**.
- Gestire tutte queste componenti richiede uno strato intermedio software: il **Sistema Operativo**.

COMPONENTI DI UN COMPUTER MODERNO (1)

- Uno o più processori
- Memoria principale
- Dischi o unità flash
- Stampanti
- Tastiera
- Mouse
- Display
- Interfacce di rete
- Dispositivi di I/O

COMPONENTI DI UN COMPUTER MODERNO (2)

- Doppia modalità supportate dall'hardware:
  - **modalità kernel** (o supervisor);
  - **modalità utente**.

THE OPERATING SYSTEM AS AN EXTENDED MACHINE

- Idea di **astrazione**
  - Il sistema operativo si pone tra l'Hardware e le Applicazioni
- Visione **top-down**
  - Il SO fornisce astrazioni ai programmi applicativi
- Vista **bottom-up**
  - Il SO gestisce parti di un sistema complesso
  - e fornisce un'allocazione ordinata e controllata delle risorse

IL SISTEMA OPERATIVO COME GESTORE DELLE RISORSE

- Da un moderno sistema operativo ci aspettiamo che **gestisca**:
  - **più programmi in esecuzione**;
  - **più utenti**.
- Necessita allocazione ordinata e controllata di
  - risorse quali: processori, memoria, unità di I/O,...
- Non solo hardware: file, database,...
- **Multiplexing:**
  - nel **tempo**: CPU, stampante,...
  - nello **spazio**: memoria centrale, disco,...

BREVE STORIA DEI SISTEMI OPERATIVI

GENERAZIONI DI SISTEMI OPERATIVI

- The first generation (1945-55): Vacuum tubes
- The second generation (1955-65): Transistors and batch systems
- The third generation (1965-1980): ICs and multiprogramming
- The fourth generation (1980-present): Personal computers
- The fifth generation (1990-present): Mobile computers

PRIMA GENERAZIONE (1945-55): VALVOLE TERMOIONICHE

- **Sviluppo storico**:
  - **John Atanasoff & Clifford Berry**: Costruirono il primo computer digitale alla Iowa State University (300 valvole).
  - **Konrad Zuse**: Costruì il **Z3** a Berlino con relè elettromeccanici.
  - **Colossus**: Progettato a **Bletchley Park**, Inghilterra.
  - **Mark I**: Costruito da **Howard Aiken** ad Harvard.
  - **ENIAC**: Creato da **William Mauchley** e **J. Presper Eckert** all'Università della Pennsylvania.
- **Caratteristiche**:
  - Varie tecnologie (binari, valvole, relè).
  - Computazioni molto lente (secondi per operazione).

The Colossus Mark 2 computer:

Cryptanalysis of the Lorenz Cipher

METODOLOGIA E LIMITI DEI PRIMI COMPUTER

- **Gestione e programmazione**:
  - Svolta interamente dagli ingegneri stessi.
  - Programmazione manuale tramite linguaggio macchina o cablaggi complessi.
- **Limitazioni**:
  - Assenza di linguaggi di programmazione (nemmeno assembly).
  - Nessun sistema operativo.
  - Alta manutenzione (soprattutto per la sostituzione delle valvole bruciate).
- **Miglioramento negli anni '50**:
  - Introduzione delle **schede perforate** per la programmazione, semplificando l'inserimento dei codici.

SECOND GENERATION: TRANSISTORS AND BATCH SYSTEMS

- **Anni ’50**: I computer diventano più affidabili grazie ai **transistor**, consentendo la vendita a grandi enti (università, industrie, governi).
  - **Organizzazione**: Ruoli distinti tra progettisti, costruttori, programmatori e operatori.
  - **Mainframe**: Grandi macchine, chiuse in sale condizionate e gestite da personale professionale.
- **Introduzione del Sistema Batch**
- **Funzionamento**:
  - Job raccolti su **schede perforate** e trasferiti su nastri magnetici con un computer ausiliario (es. IBM 1401).
  - **Macchina principale (es. IBM 7094)** usata per il calcolo, mentre l’output veniva gestito offline.

SECOND GENERATION: TRANSISTORS AND BATCH SYSTEMS (1)

- Un primo sistema batch (manuale?)
  - (a) I programmatori portano le schede al 1401.
  - (b) Il 1401 legge un lotto di lavori su nastro…

SECOND GENERATION: TRANSISTORS AND BATCH SYSTEMS (2)

- (c) L'operatore porta il nastro di input al 7094.
- (d) Il 7094 esegue i calcoli.
- (e) L'operatore porta il nastro di output al 1401.
- (f) Il 1401 stampa l'output.

SECOND GENERATION: TRANSISTORS AND BATCH SYSTEMS (3)

- **$JOB**: Specifica tempo massimo di esecuzione, numero utente e nome del programmatore.
- **$FORTRAN**: Carica il compilatore FORTRAN.
- **Programma**: Codice sorgente da compilare.
- **$LOAD**: Carica il programma compilato.
- **$RUN**: Esegue il programma con i dati forniti.
- **$END**: Termina il job.

Queste schede di controllo sono i precursori dei moderni interpreti via shell o a riga di comando.

TERZA GENERAZIONE (1965-80): IC E MULTIPROGRAMMAZIONE

- **Innovazioni Tecnologiche**:
  - Introduzione dei **circuiti integrati (IC)**: miglior rapporto prezzo/prestazioni.
  - Introduzione di **multiprogrammazione** per ridurre il tempo inattivo della CPU.
- **IBM System/360**:
  - Serie di computer **compatibili** con stesso insieme di istruzioni.
  - Gestione sia di calcoli scientifici che commerciali.
  - Successo enorme, seguito dai modelli successivi (IBM 370, 4300, 3080, 3090).
- **OS/360**:
  - Sistema operativo complesso e universale, ma difficile da mantenere.
  - Problemi di gestione e aggiornamento continui.
  - Diffusione di tecniche chiave come multiprogrammazione.
- **Impatto**:
  - Modello di **"famiglia di computer"** adottato anche dai concorrenti.
  - Discendenti delle serie System/360 sono ancora utilizzati per grandi basi di dati e server.

THIRD GENERATION: ICS AND MULTIPROGRAMMING

- Problema delle architetture batch: CPU, inattiva in attesa di I/O.
  - Per calcoli scientifici, tempo sprecato insignificante.
  - Per dati commerciali, attesa I/O 80-90%.
- **Memoria partizionata per job multipli.**
  - **Spooling** (*simultaneous peripheral operation on line*) per caricare nuovi job senza interruzioni.
- **Problema**: Tempi lunghi tra l'inserimento di un job e l'output.
  - **Soluzione**: **Timesharing** per risposta rapida.
    - CPU assegnata a job utenti attivi.
  - Protezione hardware necessaria per vero *timesharing*.
- MULTICS, sistema per multiplexing e servizi informatici.
  - Non ebbe mai veramente successo ma i suoi principi permangono nel tempo

TIMESHARING AND MULTICS

- "*I remarked to Dennis that easily half the code I was writing in Multics was error recovery code*".

—Tom Van Vleck, 1973

- Multics (uno dei primi sistemi operative ad introdurre l'idea di Timesharing) introdusse molte caratteristiche innovative che oggi ritroviamo nei sistemi operative moderni.
  - Era purtroppo anche estremamente complesso.

UNIX – A SIMPLER OPERATING SYSTEM

- Unix è un sistema operativo **multiutente** e con **multiprogrammazione**.
- Storia
  - MULTICS (MULTIplexed Information and Computing Service, 1965).
  - Ken Thompson (Bell Laboratories -1969).
    - PDP-7, scritto in ASSEMBLER.
    - UNICS (UNiplexed Information and Computing Service).
  - Ken Thompson, Dennis Ritchie (1970-1974).
    - UNIX, PDP-11.
    - Ritchie sviluppa il linguaggio C (partendo dal linguaggio B).
    - La terza versione di UNIX è scritta in C.
    - Un articolo su UNIX viene publicato nel 1974 (ACM Turing Award 1984).

UNIX – A SIMPLER OPERATING SYSTEM

- Ken Thompson crea versione monoutente di MULTICS su PDP-7.
- Questa versione evolve in sistema operativo UNIX®.
- UNIX® diventa popolare in ambito accademico e aziendale.
- Emergenza di diverse versioni UNIX, System V e BSD.
  - Sebbene inizialmente condiviso e libero, portò a molte versioni diverse e al CAOS
- IEEE sviluppa standard POSIX per garantire compatibilità inter-sistemi.

I PROGETTI DI STANDARDIZZAZIONE

- Diversi progetti di standardizzazione mirano a consolidare i vari aspetti di UNIX con lo scopo di ottenere un'interfaccia standard per programmare in UNIX.
- I più importanti sono:
  - POSIX (Portable Operating System): fusione di System V e BSD (1984).
  - VIBM, DEC, Hewlett-Packard creano OSF (Open Software Foundation) e il loro sistema UNIX è OSF/1 (1988).
  - X/OPEN definisce la Single UNIX specification (1993) e il sistema relativo ha il marchio UNIX 95.
  - "Open group (fusione di Open Software Foundation e X/OPEN; http://www.opengroup.com 1996).
    - Definizione della seconda versione della Single UNIX specification (1997) col marchio UNIX 98.

A SUCCESS STORY

UNA VARIANTE DEL SISTEMA UNIX

- Sebbene ci siano molte versioni di UNIX, le compagnie più importanti forniscono una versione basata su UNIX System V Release 4 (SVR4).
  - es. **Solaris 2.x** è l'implementazione di UNIX più diffusa e di più **largo successo da un punto di vista commerciale**.
- Questi sistemi sono molto grossi e complicati (al contrario dell'idea originaria di Thompson) e in alcuni casi anche costosi.
- Cosi, Tanenbaum sviluppa **MINIX** (1987), un piccolo sistema UNIX (11800 righe di codice C e 800 righe di codice Assembler) compatibile con gli standard POSIX.
  - MINIX è un sistema sviluppato a scopo didattico basato sul modello a micro-kernel (www.cs.vu.nl/~ast/minix.html).
  - Ancora usato nei processori moderni

DA MINIX A LINUX (1990S)

Hello everybody out there using minix -

I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) A T clones. This has been brewing since april, and is starting to get ready. I'd like any feedback on things people like/dislike in minix, as my OS resembles it somewhat (same physical layout of the file-system (due to practical reasons) among other things).
I've currently ported bash(1.08) and g c c(1.40), and things seem to work. This implies that I'll get something practical within a few months, and I'd like to know what features most people would want. Any suggestions are welcome, but I won't promise I'll implement them :-)

Linus (torvalds@kruuna.helsinki.fi)

P.S. Yes - it's free of any minix code, and it has a multi-threaded fs. It is NOT portable (uses 386 task switching etc), and it probably never will support anything other than A T-harddisks, as that's all I have :-(

E NEL FRATTEMPO MICROSOFT E APPLE?

- Anni 70: Sviluppo dei circuiti LSI e nascita dei personal computer.
  - (Large Scale Integration, integrazione su larga scala), chip contenenti migliaia di transistor su un centimetro quadrato di silicio
- Intel introduce l'8080, necessità di un sistema operativo.
  - Gary Kildall crea CP/M e fondò Digital Research.
  - Digital Research adatta CP/M per microcomputer (o PC anti-litteram)
- Anni 80: IBM cerca sistema operativo per il PC, incontra Digital Research.
  - Kildall rifiuta incontro con IBM, opportunità mancata.
- Microsoft acquisisce DOS da Seattle Computer Products.
  - Nasce MS-DOS e domina il mercato dei PC IBM.

E NEL FRATTEMPO MICROSOFT E APPLE? (2)

- Anni 80/90:
  - GUI introdotta da Doug Engelbart e adottata da Xerox PARC.
  - Steve Jobs ispirato da GUI, sviluppa Apple Macintosh.
  - Microsoft crea Windows come ambiente grafico su MS-DOS.
- Evoluzioni:
  - Apple consolida il suo sistema operativo basato su UNIX ma *architecture-specific*
  - Windows NT influenzato da VAX VMS e sfida legale con DEC.
  - Windows 2000 e successori consolidano la posizione di Microsoft.
  - Vista e Windows 8 incontrano resistenza, successo con Windows 10.

DA ALLORA, LINUX (E ANDROID) HANNO PROSPERATO

- Soprattutto in ambienti
  - Server
  - Mobile

MA ANCHE MINIX

This is Google's cache of ftp://ftp.supermicro.com/CDR-X11-UP_1.10_for_Intel_X11_UP_platform/Intel/ME/Other_Licenses/Minix3_License.txt. It is a snapshot of the page as it appeared on Mar 23, 2017 15:18:02 GMT.

```
License Copyright (c) 1987, 1997, 2006, Vrije Universiteit, Amsterdam, The Netherlands
All rights reserved. Redistribution and use of the MINIX 3 operating system in source and
binary forms, with or without modification, are permitted provided that the following
conditions are met: * Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer. * Redistributions in binary
form must reproduce the above copyright notice, this list of conditions and the following
disclaimer in the documentation and/or other materials provided with the distribution. *
Neither the name of the Vrije Universiteit nor the names of the software authors or
contributors may be used to endorse or promote products derived from this software
```

- MINIX 3 è stato adottato da Intel per il suo motore di gestione ed è ora presente in desktop, server e laptop.

UNO SGUARDO ALL'HARDWARE

UNO SGUARDO DI INSIEME

- Architettura (semplificata) di un calcolatore

IL PROCESSORE

- La CPU è il cervello del computer, esegue istruzioni dalla memoria.
  - Il ciclo base della CPU: **preleva** (*fetch*), **decodifica** (*decode*), **esegue** (*execute*) istruzioni.
  - I programmi vengono eseguiti in questo ciclo.

- Le CPU eseguono un set specifico di istruzioni.
  - Registri interni memorizzano dati importanti e risultati.
  - I set di istruzioni includono funzione per il caricamento/salvataggio dati dalla memoria.
- Registri speciali come il **program counter** indicano l'istruzione successiva.
- Lo **stack pointer** punta alla cima dello stack in memoria.
  - Lo stack contiene frame di procedure con parametri e variabili locali.
- Il **registro Program Status Word (PSW)** contiene informazioni sullo stato del programma.
  - Il PSW è fondamentale per chiamate di sistema e I/O.

IL PROCESSORE

- Il sistema operativo gestisce il *multiplexing* temporale della CPU.
  - Durante il *multiplexing*, il sistema operativo salva e ripristina i registri.
  - Ciò permette al sistema operativo di eseguire programmi in modo efficiente.
- Progettazioni avanzate: **pipeline**
  - non del tutto trasparenti al SO

(a)

(b)

PIÙ DI UN PROCESSORE

- *Multithreading* (o *hyperthreading*):
  - tiene all'interno della CPU lo stato di due thread ma non c'è una esecuzione parallela vera e propria;
  - il S.O. deve tenerne conto.
- Multiprocessori, vantaggi:
  - throughput;
  - economia di scala;
  - affidabilità;
- Multicore
- GPU
- Impatto fondamentale delle ***cache***

LA MEMORIA DI UN CALCOLATORE

LA MEMORIA DI UN CALCOLATORE (2)

- Problemi del sistema di cache:
  - Quando inserire un nuovo elemento nella cache?
  - In quale riga della cache inserire il nuovo elemento?
  - Quale elemento rimuovere dalla cache quando è necessario uno slot.
  - Dove mettere un elemento appena eliminato nella memoria più grande.

NONVOLATILE STORAGE

Spesso (erroneamente) indicato anche come disco. Nessuna parte in movimento, dati in memoria elettronica (flash). Molto più veloce dei dischi magnetici.

DISPOSITIVI DI I/O

- Si individuano due componenti:
  - **il controller**: più semplice da usare per il SO;
  - il **dispositivo** in sé: interfaccia elementare ma complicata da pilotare.
    - esempio: dischi SATA.
- Ogni controller ha bisogno di un *driver* per il S.O.
- Il **driver interagisce con il *controller*** attraverso le porte di I/O:
  - istruzioni tipo IN / OUT;
  - mappatura in memoria.

DISPOSITIVI DI I/O (2)

- Per eseguire l'I/O:
  - Il processo esegue la chiamata di sistema
  - Il kernel effettua una chiamata al driver
  - Il driver avvia l'I/O

- e interroga il dispositivo per vedere se ha finito (è occupato in attesa)
- o chiede al dispositivo di generare un interrupt quando ha finito (e restituisce qualche informazion)
- più avanzato: fa uso di hardware speciale
  - DMA

IL DMA

- DMA è l'acronimo di "*Direct Memory Access*" ed è un dispositivo hardware speciale.
- Consente ai componenti di accedere direttamente alla memoria del computer senza coinvolgere la CPU.
- Migliora l'efficienza ed aumenta le prestazioni nelle operazioni di input/output (I/O) ad alta velocità.
- Utilizzato per il trasferimento veloce di dati tra memoria e dispositivi periferici.
- Riduce il carico sulla CPU durante le operazioni di I/O, consentendole di concentrarsi su altri compiti critici.

BUSES

- L'evoluzione dei computer ha portato all'aggiunta di bus supplementari per gestire l'aumento del traffico dati.
  - Un sistema x86 moderno ha diversi bus con funzioni e velocità di trasferimento diverse.
  - **la CPU comunica con la memoria attraverso un bus veloce DDR4, con una periferica grafica esterna sul bus PCIe** (*Peripheral Component Interconnect Express*)
  - con tutti gli altri dispositivi attraverso un hub su un bus DMI (Direct Media Interface).
- Il bus PCIe è il principale e più veloce bus di comunicazione nei computer attuali.
  - **PCIe utilizza un'architettura a connessioni punto a punto dedicate**, migliorando l'efficienza rispetto ai bus condivisi.

BUSES

- Dispositivi *legacy* possono essere collegati a un processore hub separato.
- USB (Universal Serial Bus) è stato sviluppato per connettere dispositivi lenti al computer, ma le versioni moderne possono raggiungere velocità fino a 40 Gbps.
- USB utilizza un connettore a 4-11 conduttori per alimentazione e comunicazione.
- USB consente il collegamento immediato dei dispositivi senza necessità di riavvio del sistema.

AVVIO DEL SISTEMA (BOOT)

- La memoria flash della scheda madre contiene il firmware (il BIOS).
- Dopo aver premuto il pulsante di accensione, la CPU esegue il BIOS che
  - Inizializza RAM e altre risorse
  - Esegue la scansione dei bus PCI/PCIe e inizializza i dispositivi
  - Imposta il firmware *runtime* per i servizi critici (ad esempio, I/O a basso livello) che il sistema deve utilizzare dopo l'avvio.
- Il BIOS cerca la posizione della tabella delle partizioni sul secondo settore del dispositivo di avvio
  - Contiene le posizioni di altre partizioni
- Il BIOS è in grado di leggere semplici file system (ad esempio, FAT-32) e avvia il primo programma di *bootloader* (dalla partizione indicata dal boot manager).
  - Il *bootloader* può caricare altri programmi di *bootloading*.
- Alla fine viene caricato l'OS

INTRODUZIONE AI CONCETTI BASE DEI S.O.

Danilo Croce

Ottobre 2024

RIPARTIAMO DALLA DOMANDA DI IERI COS'È UN SISTEMA OPERATIVO?

- Un **moderno calcolatore** è tipicamente formato da:
  - uno o più processori;
  - memoria centrale;
  - dischi;
    stampanti e altre periferiche di I/O.
- I dettagli di basso livello sono **molto complessi**.
- Gestire tutte queste componenti richiede uno strato intermedio software: il **Sistema Operativo**.

LO ZOO DEI SISTEMI OPERATIVI

1. **Sistemi Operativi per Mainframe**: computer di grandi dimensioni usati in grandi aziende e istituzioni.
   - **Funzioni principali**:
     - **Batch processing**: esecuzione di lavori senza interazione dell'utente (es. elaborazione reclami).
     - **Elaborazione di transazioni**: gestione di numerose richieste simultanee (es. prenotazioni aeree).
     - **Timesharing**: molti utenti eseguono lavori contemporaneamente (es. interrogazioni database).
   - **Esempi**: **Z/OS** (successore di OS/360), usato per applicazioni mission-critical (banche, e-commerce).
2. **Sistemi Operativi per Server**: **Utilizzati su server** che servono più utenti attraverso la rete.
   - Forniscono servizi come: **File sharing**, **database**, **stampa**, **hosting web**.
   - Usati da provider di servizi internet e aziende per supportare infrastrutture critiche.
   - **Esempi**: **Linux**, **FreeBSD**, **Windows Server**, **Solaris**.

LO ZOO DEI SISTEMI OPERATIVI

3. **Sistemi Operativi per Personal Computer**: **Supportano un singolo utente** con multiprogrammazione e, spesso, architetture multiprocessore.
   - Utilizzati per compiti quotidiani: **videoscrittura**, **navigazione web**, **fogli di calcolo**.
   - **Esempi**: **Windows**, **macOS**, **Linux**, **FreeBSD**.
   - **Caratteristiche**: facilità d'uso, interfaccia grafica, supporto per applicazioni di produttività.
4. **Sistemi Operativi per Smartphone e Tablet**: Usati in **dispositivi mobili** come **smartphone** e **tablet**.
   - Mercato dominato da:
     - **Android** (Google) e **iOS** (Apple).
   - Supportano **CPU multicore**, **GPS**, **fotocamere** e numerose **app di terze parti** (oltre 3 milioni su Google Play Store).

LO ZOO DEI SISTEMI OPERATIVI

6. **Sistemi Operativi per IoT e Embedded**: Usati in **dispositivi connessi** (es. frigoriferi, lavatrici, telecamere di sicurezza).
   - Tipicamente **leggeri**, con funzioni specifiche e limitate.
   - Esempi: **Embedded Linux**, **QNX**, **TinyOS**.
   - **Caratteristiche**: footprint ridotto (es. **RIOT** può girare su meno di 10 KB).
7. **Sistemi Operativi Real-Time**: Progettati per rispettare **scadenze rigide** nei processi industriali e militari.
   - **Hard real-time**: risposte immediate e precise (es. robotica industriale).
   - **Soft real-time**: occasionali ritardi tollerabili (es. streaming multimediale).
   - **Esempi**: **eCos**, **VxWorks**.
8. **Sistemi Operativi per Smart Card**: Utilizzati in **smart card** per pagamenti, autenticazioni e altro. Alimentate tramite contatti o induzione, con capacità limitate.
   - **Java-oriented**: alcune eseguono applet Java per multiprogrammazione.
   - **Esempi**: Sistemi proprietari, **JavaCard** per applet multiprogrammate.

COSA HANNO IN COMUNE?

- Extended Machine
  - Estensione della funzionalità hardware
  - Astrazione dell'hardware
  - Nascondere i dettagli al programmatore
- Resource Manager
  - Protegge l'uso simultaneo/non sicuro delle risorse
  - Condivisione equa delle risorse
  - Resource accounting/limiting

CONCETTI "BASE" DI UN SISTEMA OPERATIVO

- Il sistema operativo offre funzionalità attraverso le chiamate di sistema
- Gruppi di chiamate di sistema implementano servizi, ad esempio
  - File System Service
  - Process Management Service
- I processi sono astrazioni a livello utente per eseguire un programma per conto dell'utente.
- Ogni processo ha il proprio spazio di indirizzamento
- I dati coinvolti nell'elaborazione vengono recuperati e memorizzati in file.
- I file persistono rispetto ai processi

COSA È UN PROCESSO?

- Concetto chiave in tutti i sistemi operativi
- **Definizione: un programma in esecuzione**
- Il processo è associato
  - a uno spazio di indirizzi
  - un insieme di risorse
    - registri
    - file aperti
    - allarmi
    - …
- Il processo può essere pensato come un contenitore
  - Contiene tutte le informazioni necessarie per l'esecuzione del programma

IL LAYOT DI UN PROCESSO

- Layout dipende da:
  - Architettura della macchina
  - Il Sistema operativo
  - Il programma
- Very basic layout:
  - Stack: Active call data
  - Data: Program variables
  - Text: Program code

IL CICLO DI VITA DI UN PROCESSO

- Le informazioni sui processi sono conservate nella tabella dei processi del sistema operativo.

Un processo sospeso consiste in una voce della tabella dei processi (registri salvati e altre informazioni necessarie per riavviare il processo) e nel suo spazio degli indirizzi.

- Gestione dei processi:
  - Operazioni come la **creazione**, la **terminazione**, la **pausa** e la **ripresa** di un processo.
- Un processo può creare un altro processo
  - Conosciuto come processo “figlio”
  - Crea una gerarchia (o "albero") di processi

ALBERO DEI PROCESSI

- Un albero di processi. Il processo A ha creato due processi figli, B e C. Il processo B ha creato tre processi figli, D, E e F.

CHI "POSSIEDE" UN PROCESSO?

- I processi sono "di proprietà" di un utente, identificato da un UID.
  - Ogni processo ha tipicamente l'UID dell'utente che lo ha avviato.
  - Su UNIX, un processo figlio ha lo stesso UID del suo processo padre.
  - Gli utenti possono essere membri di gruppi, identificati da un GUID.

Un processo (superuser/root/administrator) è speciale: ha più privilegi !

FILE

- **File**: astrazione di un dispositivo di memorizzazione (eventualmente) reale (ad esempio, un disco).
- È possibile leggere e scrivere dati da/su file fornendo una posizione e una quantità di dati da trasferire.
- I file vengono collezionati in directory (o cartelle)
  - Una directory conserva un identificatore per ogni file che contiene.
  - Una directory è un file a sé stante
  - La filosofia UNIX: "*Everything is a file*".

FILE

- Le directory e i file formano una gerarchia:
  - La gerarchia inizia dalla "directory principale" or "directory radice" (**root**):
    - `/`
  - È possibile accedere ai file tramite percorsi assoluti (*absolute path*):
    - `/home/ast/todo-list`
  - … o percorsi relativi a partire dalla directory di lavoro corrente:
    - `../courses/slides1.pdf`
  - Altri filesystem possono essere montati (da **mount**) nella root:
    - `/mnt/windows`

DIRITTI DI ACCESSO

- I file sono "protetti" da tuple a tre bit per il **proprietario** (***owner***), il **gruppo** (***group***) e gli **altri utenti (*other users*)**.
- Le tuple contengono un bit (r)ead, (w)rite e un bit e(x)ecute (ma sono disponibili più bit)
- Example:

```
-rwxr-x--x myuser my group 14492 Dec 4 18:04 my file
```

  - Owner is allowed to execute, modify, read the file
  - Group is allowed to read and execute the file
  - Other users are only allowed to execute the file

UN ESEMPIO DI ORGANIZZAZIONE DI UN FILE SYSTEM

Un file system per un dipartimento universitario.

FILE E RISORSE

a) Prima del *mount*, i file dell'unità USB non sono accessibili.

b) Dopo aver eseguito l'operazione di *mount*, fanno parte della gerarchia dei file.

ACCESSO AI FILE: UN ESEMPIO

```
herbertb@sleet:~/tmp$ pwd
/home/herbertb/tmp
herbertb@sleet:~/tmp$ mkdir os
Herb ertb@sleet:~/tmp$ cat > os/hello.sh
#!/bin/sh
echo "hello world!"
herbertb@sleet:~/tmp$ chmod 744 os/hello.sh
herbertb@sleet:~/tmp$ os/hello.sh
hello world!
herbertb@sleet:~/tmp$ chmod 644 os          ← What happens here?
herbertb@sleet:~/tmp$ cat > os/qq.txt       ← Will this work?
herbertb@sleet:~/tmp$ ls os/                ← Will this work?
herbertb@sleet:~/tmp$ os/hello.sh           ← Will this work?
herbertb@sleet:~/tmp$ rm os/hello.sh        ← Will this work?
```

"EVERYTHING IS A FILE":

- Hardware devices are abstracted as files:
  Block special files, esempio i dischi:
  `brw-rw---- 1 root root 8, 2 Dec 4 18:04 /dev/sda2`
- Character special files, ad esempio le porte seriali:
  `crw-rw---- 1 root root 4,64 Dec 4 18:04 /dev/ttyS0`
- Altri file speciali:
  symbolic links
  named/anonymous FIFOs (sockets/pipes)
- "Everything is a file *descriptor*"

FILE E PIPE

- Pipes: pseudo file che consentono ai processi di comunicare su un canale FIFO
- Deve essere impostato in anticipo
- Sembra un file "normale" per leggere e scrivere da/sui processi in esecuzione.

Due processi collegati da una pipe

TERMINI IMPORTANTI

- Path
- Folder / map / directory
- Root directory
- Working directory
- File descriptor
- Mounting
- Block/character special files
- Pipe

LE CHIAMATE DI SISTEMA (O SYSTEM CALL)

CHIAMATE DI SISTEMA

Le chiamate di sistema sono l'**interfaccia che il sistema operativo offre alle applicazioni per richiedere servizi**.

Problema:

- Il meccanismo delle chiamate di sistema è **altamente specifico** del sistema operativo e dell'hardware.
- La **necessità di efficienza esaspera** questo problema

Soluzione:

- Incapsulare le chiamate di sistema nella libreria C (`libc`)
- Tipicamente esporta **una chiamata di libreria per ogni chiamata di sistema**
- UNIX `libc` si basa sulla libreria C POSIX
- Si noti che esistono molte librerie C UNIX... :-/

I 10 PASSI PER EFFETTUARE LA CHIAMATA DI SISTEMA

`read(fd, buffer, nbytes)`

- **Preparazione dei parametri**: il programma chiamante prepara i parametri (i.e., `fd`, `buffer`, `nbytes`)
  - solitamente memorizzandoli nei registri (es, `RDI`, `RSI`, `RDX`) o nello stack (se di più)
- **Chiamata alla procedura di libreria**: Il programma effettua la chiamata alla procedura di libreria (step 4)
- **Collocazione del numero di chiamata di sistema**: colloca il numero della chiamata di sistema in un registro, come RAX (passaggio 5).
  - Questo numero identifica quale funzione del kernel deve essere eseguita (es. `read`, `write`, `open`, ecc.).
  - Il kernel può consultare una tabella di chiamate di sistema, dove ogni numero è associato a un gestore specifico.
- **Passaggio a modalità kernel**: si esegue un'istruzione "**trap**" (ad esempio, SYSCALL in x86-64)
  - passare dalla modalità utente a quella kernel (passaggio 6).
  - L'istruzione "trap" è simile a una chiamata di procedura ma cambia la modalità in modalità kernel
  - può saltare solo a indirizzi specifici o indici di una tabella di memoria, a differenza della chiamata di procedura.

I 10 PASSI PER EFFETTUARE LA CHIAMATA DI SISTEMA

`read(fd, buffer, nbytes)`

- **Esecuzione del gestore di chiamate di sistema**: Il gestore di chiamate di sistema specifico viene eseguito (passaggio 8).
- **Ritorno alla procedura di libreria utente**: dopo, il controllo può essere restituito alla procedura di libreria utente, all'istruzione successiva all'istruzione "trap" (passaggio 9).
- **Possibilità di blocco**: La chiamata di sistema può bloccare il chiamante, ad esempio, se l'input desiderato non è disponibile.
  - Il sistema operativo può quindi eseguire altri processi
- **Ripresa dopo il blocco**: Quando l'input o le condizioni desiderate sono disponibili, il processo bloccato viene ripreso,
  - Tornando alla procedura di libreria utente e procedendo all'istruzione successiva (passo 10).

CHIAMATE DI SISTEMA PER LA GESTIONE DEI PROCESSI

Call	Description
`pid fork( )`	Creare un processo figlio identico al genitore
`pid waitpid( pid, &statloc, options)`	Attendere che un processo figlio termini
`s = execve(name, argv, environp)`	Sostituire l'immagine centrale di un processo
`exit(status)`	Terminare l'esecuzione del processo e restituire lo stato

- Alcune delle principali chiamate di sistema POSIX.
- Il codice di ritorno s è -1 se si è verificato un errore. I codici di ritorno sono i seguenti: `pid` è l'id di un processo

CHIAMATE DI SISTEMA PER LA GESTIONE DEI PROCESSI

Call	Description
`fd = open(file, how, ...)`	Apre un file per la lettura, la scrittura o entrambe le operazioni.
`s = close(fd)`	Chiude un file aperto
`n = read(fd, buffer, nbytes)`	Legge dati da un file in un buffer
`n = write(fd, buffer, nbytes)`	Scrive dati da un buffer in un file
`Position = lseek(fd, offset, whence)`	Sposta il puntatore del file
`s = stat(name, &buf)`	Ottiene informazioni sullo stato di un file

- Il codice di ritorno s è -1 se si è verificato un errore. I codici di ritorno sono i seguenti: `fd` è un descrittore di file, `n` è un conteggio di byte, `position` è un offset all'interno del file.

CHIAMATE DI SISTEMA PER LA GESTIONE DEL FILE SYSTEM

Call	Description
`s = mkdir(name, mode)`	Crea una nuova directory
`s = rmdir(name)`	Rimuove una directory vuota
`s = link(name1 , name2)`	Crea una nuova voce, nome2, che punta a nome1
`s = unlink(name)`	Rimuove una voce della directory
`s = mount(special, name, flag)`	Monta un file system
`s = umount(special)`	Smonta un file system

ALTRE CHIAMATE DI SISTEMA

Call	Description
`s = chdir(dirname)`	Cambia la directory di lavoro
`s = chmod(name,mode)`	Modifica i bit di protezione di un file
`s = kill(pid,signal)`	Invia un segnale a un processo (NON UCCIDE)
`s = time(&seconds)`	Ottiene il tempo trascorso dal 1° gennaio 1970

UN ESEMPIO

```
#define TRUE 1

while (TRUE) {                                  /* repeat forever */
        type_prompt( );                         /* display prompt on the screen */
        read_command(command, parameters);      /* read input from terminal */

        if (fork( ) != 0) {                     /* fork off child process */
            /* Parent code. */
            waitpid(-1, &status, 0);            /* wait for child to exit */
        } else {
            /* Child code. */
            execve(command, parameters, 0);     /* execute command */
        }
}
```

Si assume che TRUE sia definito come 1.

API DI WINDOWS

UNIX	Win32	Descrizione
fork	CreateProcess	Crea un nuovo processo
waitpid	WaitForSingleObject	Può attendere l'uscita di un processo
execve	(none)	Createprocess = fork + execve
exit	ExitProcess	Termina l'esecuzione
open	createFile	Crea un file o apre un file esistente
close	CloseHandle	Chiude un file
read	ReadFile	Legge dati da un file
Write	WriteFile	Scrive dati in un file
lseek	SetFilePointer	Sposta il puntatore del file
stat	GetFileAttributesEx	Ottiene vari attributi del file
mkdir	CreateDirectory	Crea una nuova directory

- Le chiamate API Win32 che corrispondono grosso modo alle chiamate UNIX

API DI WINDOWS (2)

- Le chiamate API Win32 che corrispondono grosso modo alle chiamate UNIX

UNIX	Win32	Description
Iseek	SetFilePointer	Move the tile pointer
stat	GetFileAttributesEx	Get various file attributes
mkdir	CreateDirectory	Create a new directory
rmdjr	RemoveDirectory	Remove an empty directory
link	(none)	Win32 does not support links
unlink	DeleteFile	Destroy an existing file
mount	(none)	Win32 does not support mount
umount	(none)	Win32 does not support mount, so no umount
chdir	SetCurrentDirectory	Change the current working directory
chmod	(none)	Win32 does not support security (although NT does)
kill	(none)	Win32 does not support signals
time	GetLocamme	Get the current time

STRUTTURA DI UN SISTEMA OPERATIVO

STRUTTURA DI UN SISTEMA OPERATIVO: MONOLITICO

- Il programma principale invoca le chiamate di sistema richieste
- Il kernel è un blocco monolitico con:
  - Procedure di servizio che eseguono le chiamate di Sistema
  - Procedure di utilità che aiutano a implementare le procedure di servizio

ALCUNE CONSIDERAZIONI

- **Sistemi Operativi Monolitici**: un approccio al design «tutto in uno»
  - il kernel è un'unica unità grande e interconnessa.
  - tutte le funzioni del sistema operativo, come la gestione dei processi, la gestione della memoria e la gestione dei dispositivi di I/O, sono strettamente integrate in un unico spazio di indirizzamento.
- **Flessibilità vs Complessità:**
  - offre una certa flessibilità in termini di prestazioni e design.
  - tuttavia, dato che tutto è strettamente interconnesso, un bug o un errore in una parte del sistema può causare problemi in altre parti, potenzialmente portando a crash sistemici.
- **Compilazione e Collegamento:**
  - tutte le funzioni e procedure del sistema operativo devono essere compilate e collegate in un unico eseguibile del kernel.

ALTRE CONSIDERAZIONI

- **Mancanza di occultamento:**
  - tutte le procedure possono, in teoria, accedere a qualsiasi altra procedura o variabile all'interno del kernel.
  - Non c'è un vero e proprio "occultamento" o isolamento tra le diverse parti del sistema.
- **Utilizzo di «trap»:**
  - meccanismo attraverso il quale un programma può richiedere i servizi del sistema operativo.
    - avviene attraverso interruzioni software che trasferiscono il controllo al sistema operativo.
- **Una «struttura a tre strati»**
  - una suddivisione del sistema in livelli, spesso user mode, kernel mode e hardware, con il «trap» che agisce come meccanismo di comunicazione tra questi livelli.

ALTRE CONSIDERAZIONI

- **Estensioni caricabili:**
  - Molti sistemi operativi **permettono di caricare dinamicamente componenti aggiuntivi**, come driver di dispositivi o file system, senza dover riavviare o ricompilare l'intero sistema operativo.
  - Questi componenti possono essere caricati e scaricati a seconda delle necessità, offrendo una certa modularità anche in un sistema monolitico.
- **Librerie Condivise e DLL:**
  - Sia UNIX che Windows supportano l'idea di librerie di codice che possono essere condivise tra più programmi.
    - In UNIX, queste sono chiamate "librerie condivise",
    - In Windows sono chiamate "Dynamic Link Libraries" (DLL).
  - contengono codice che può essere eseguito da più programmi contemporaneamente, riducendo la necessità di avere copie multiple del medesimo codice in memoria.

STRUTTURA DI UN SISTEMA OPERATIVO: MONOLITICO

- L'organizzazione stratificata dei sistemi operativi è una generalizzazione dell'approccio monolitico.
- Il sistema THE fu uno dei primi a implementare questa idea, con sei livelli gerarchici.
  - Questi livelli gestivano l'allocazione del processore, la memoria, la comunicazione, l'I/O, i dispositivi, e gli utenti.
- Il sistema MULTICS usava anelli concentrici per definire i privilegi, con livelli interni più privilegiati di quelli esterni.
- Vantaggi:
  - Protezione delle risorse e dati critici.
  - Separazione chiara dei compiti (es. valutazione degli studenti).

STRUTTURA DI UN SISTEMA OPERATIVO: MONOLITICO

- **Kernel Unificato**: Tutte le funzionalità centralizzate in un unico kernel.
- **Interconnessione**: Ogni componente ha la capacità di richiamare qualsiasi altro componente.
- **Scalabilità**: Questa struttura può diventare complessa e meno gestibile con l'evoluzione del sistema.

ORIGINE DELLE MACCHINE VIRTUALI

- Le **macchine virtuali (VM)** permettono l'esecuzione di più sistemi operativi su un unico hardware fisico, simulando un ambiente separato per ciascuno.
- **Origini**: Il sistema **VM/370** di IBM (anni '70) è stato uno dei primi a implementare macchine virtuali, creando ambienti virtuali identici all'hardware fisico, capaci di eseguire diversi sistemi operativi contemporaneamente.
- **Vantaggi**:
  - **Isolamento**: Ogni VM opera indipendentemente.
  - **Flessibilità**: Possibilità di eseguire diversi sistemi operativi simultaneamente (es. OS/360, CMS).
  - **Gestione Semplificata**: Facilita la gestione e manutenzione separando multiprogrammazione e risorse hardware.
- **Uso Moderno**: Il successore **z/VM** viene utilizzato sui mainframe IBM serie Z per far girare più sistemi operativi completi, come **Linux**, in ambienti ad alta intensità di transazioni e dati.

STRUTTURA DI UN SISTEMA OPERATIVO: VIRTUALIZZAZIONE (1 DI 2)

- Inventato negli anni '70 per separare la multiprogrammazione dalla macchina estesa
- Oggi di nuovo interesse in diversi ambiti
- N interfacce di chiamata di sistema indipendenti dal sistema operativo

The structure of VM/370 with Conversational Monitor System (CMS)

STRUTTURA DI UN SISTEMA OPERATIVO: VIRTUALIZZAZIONE (2 DI 2)

- Virtual machine monitor (VMM) o Hypervisor emula l'hardware
  - **Type 1**: VMM viene eseguito sul "pezzo di ferro" (direttamente su HW, come Xen)
  - **Type 2**: VMM ospitato nel sistema operative (esempio: QEMU)
    - In teoria non c'è nessuna specializzazione da parte del Sistema operative sottostante (fig. b)
    - Nella pratica esistono dei moduli del Kernel usati per ottimizzare il processo di simulazione (fig. c)

(a) A type 1 hypervisor. (b) A pure type 2 hypervisor. (c) A practical type 2 hypervisor

CONTAINER

- I container possono eseguire più istanze di un sistema operativo su una singola macchina.
- Ogni container condivide il kernel del sistema operativo host e i file binari e le librerie.
  - Il container non contiene il sistema operativo completo e può quindi essere leggero
- Gli svantaggi dei container
  - Non è possibile eseguire un container con un sistema operativo completamente diverso da quello dell'host
  - A differenza delle macchine virtuali, non esiste un rigido partizionamento delle risorse.
  - I container sono isolati a livello di processo
    - Se un container altera la stabilità del kernel sottostante, ciò può influire sugli altri container.

STRUTTURA DI UN SISTEMA OPERATIVO: EXOKERNEL

- **Idea**: Separare il controllo delle risorse dalla macchina estesa
- Simile a un VMM/Hypervisor, ma:
  - Exokernel non emula l'hardware
  - Fornisce solo una condivisione sicura delle risorse a basso livello
- Ogni macchina virtuale a livello utente esegue il suo sistema operativo, ma è limitata a utilizzare solo le risorse assegnate.
- Rispetto ad altri approcci, l'exokernel elimina la necessità di mappature complesse, concentrandosi solo su quale macchina virtuale ha accesso a quali risorse.

STRUTTURA DI UN SISTEMA OPERATIVO: UNIKERNEL

- Gli unikernel sono sistemi minimi basati su LibOS, progettati per eseguire una singola applicazione su una macchina virtuale
  - Esempio WebServer
- Questi sistemi contengono solo la funzionalità necessaria per supportare l'applicazione specifica, come un server web, su una macchina virtuale.
- Gli unikernel sono altamente efficienti poiché non richiedono protezione tra il sistema operativo (LibOS)
  - esiste solo un'applicazione per macchina virtuale.
- Il concetto degli unikernel è stato recentemente riscoperto, offrendo una soluzione leggera ed efficiente per eseguire applicazioni isolate su macchine virtuali.

STRUTTURA DI UN SISTEMA OPERATIVO: MICROKERNEL-BASED CLIENT/SERVER (1 OF 3)

- Organizza le *service procedure* che vengono eseguiti in modo separato processes => **System Servers/Drivers**
- I processi di sistema comunicano attraverso il passaggio di messaggi
- Le chiamate di sistema si basano sullo stesso meccanismo di messaggistica
- Meccanismo di messaggistica implementato nel kernel minimale => **Microkernel**

STRUTTURA DI UN SISTEMA OPERATIVO: MICROKERNEL-BASED CLIENT/SERVER (2 OF 3)

Struttura semplificata del sistema MINIX

STRUTTURA DI UN SISTEMA OPERATIVO: MICROKERNEL-BASED CLIENT/SERVER (3 OF 3)

- **Pro**: è più facile aderire al Principle of Least Authority (POLA):
  - Trusted Computing Base (TCB) relativamente "piccolo"
  - Ogni processo del sistema operativo può fare solo ciò che è necessario per svolgere il proprio compito
  - La compromissione, ad esempio, del driver di stampa non influisce sul resto del sistema operativo.
- **Contra:**
  - Il passaggio di messaggi è più lento di una chiamata di funzione (come in un kernel monolitico)