Genesi – 1

- **MS-DOS**
  - Mono-utente in modalità *command line*
  - **Non multi-programmato**
  - Inizialmente ispirato a CP/M
    - 1981 : **1.0** (8 KB) → PC IBM 8088 (16 *bit*)
    - 1986 : **3.0** (36 KB) → PC IBM/AT (i286 @ 8 MHz, ≤ 16 MB)
- **Windows** 1ª generazione
  - Modalità GUI solo come rivestimento di MS-DOS
  - Interfaccia **copia** del 1º modello Macintosh di Apple
    - 1990 - 1993 : **3.0**, **3.1**, **3.11** → i386 (32 *bit*)

GUI

- **GUI** (*Graphical User Interface*)
  - Introdotto dal modello **Macintosh** di Apple il 24 gennaio 1984
    - Vedi `http://www.apple-history.com/lisa.html`

  - Basato sul paradigma **WIMP** (dispregiativo!)
    - Finestre (*windows*), icone (*icons*), *m*enu e dispositivi di puntamento (*pointing*)
- Realizzabile
  - Sia come programma in spazio utente (GNU/Linux)
  - Che come parte del S/O (Windows)

Genesi – 2

- **Windows** 2ª generazione
  - Vero e proprio S/O **multiprogrammato** ma sempre **mono-utente** con FS su modello FAT
    - 1995 : Windows 95 (MS-DOS 7.0)
    - 1998 : Windows 98 (MS-DOS 7.1)
      - Nucleo a procedure **non rientranti**
        - Incapaci di consentire più esecuzioni simultanee
        - Ogni accesso a nucleo protetto da semaforo a mutua esclusione
        - Scarsissimi benefici di multiprogrammazione
      - ¼ dello spazio di indirizzamento di processo (4 GB totali) condiviso R/W con gli altri processi; ¼ condiviso R/W con il nucleo
        - Scarsissima integrità dei dati critici
    - 2000 : Windows Me (ancora MS-DOS)

Modeste modifiche

Genesi – 3

- **Windows** 3ª generazione
  - Progetto **NT**: abbandono della base MS-DOS
    - Architettura a 16 *bit*
  - Enfasi su sicurezza e affidabilità
  - FS di nuova concezione (**`NTFS`**)
    - 1993 : Windows NT 3.1 → fiasco commerciale per la mancanza di programmi di utilità
    - 1996 : Windows NT 4.0 → reintroduzione di interfaccia e programmi Windows 95
      - Scritto in C e C++ per massima portabilità al costo di grande complessità (16 M linee di codice!)
      - Molto superiore a Windows 95/98 ma privo di supporto per *plug-and-play* gestione batterie e emulatore MS-DOS

Genesi – 4

- **Windows** 3ª generazione (segue)
  - **Architettura di NT 3.1** a ***microkernel*** e modello ***client-server***
    - La maggior parte dei servizi è incapsulata in processi di sistema eseguiti in modo utente e offerti ai processi applicativi tramite scambio messaggi
  - **Elevata portabilità**
    - Dipendenze *hardware* localizzate nel nucleo
  - **Bassa velocità**
    - Più costosa l'esecuzione in modo privilegiato
  - **Architettura di NT 4.0** a **nucleo monolitico**
    - Servizi di sistema riposizionati entro il nucleo

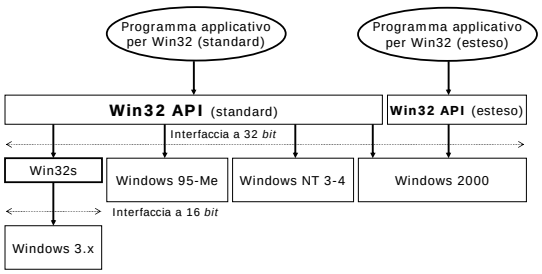
Genesi – 5

- **Windows** 3ª generazione (segue)
  - 1999 : Windows 2000 (alias di **NT 5.0**)
    - Il S/O esegue in **modo nucleo**
      - Lo spazio di indirizzamento dei processi è interamente privato e distinto dal modo nucleo
        - **Memoria virtuale**
    - Periferiche rimuovibili
      - *Plug-and-play*
    - Internazionalizzazione
      - **Unica** versione configurabile per lingua nazionale
    - Alcune migliorie a **`NTFS`**
    - MS-DOS completamente rimpiazzato da una *shell* di comandi che ne riproduce e estende le funzionalità
    - Enorme complessità: oltre 29 M linee di codice C[++]

Interfaccia di programmazione – 1

- Basato su principio speculare a quello adottato da UNIX e GNU/Linux
  - Interfaccia di sistema **non** pubblica
  - Procedure di libreria pubblicate in **Win32 API** (***Application Programming Interface***) a uso del programmatore ma controllata da Microsoft
  - Alcune procedure includono chiamate di sistema
  - Altre svolgono servizi di utilità eseguiti interamente in modo utente
    - Nessun sforzo di evitare ridondanza o rigore gerarchico

Interfaccia di programmazione – 2



Informazioni di configurazione

- Tutte le informazioni vitali di configurazione del sistema sono raccolte in una specie di FS detto `registry` salvato su disco in *file* speciali detti `hives`
  - *Directory* → *key*
  - *File* → *entry* = {nome, tipo, dati}
- 6 *directory* principali con prefisso `HKEY_`
  - Per esempio: `HKEY_LOCAL_MACHINE` con *entry* descrittive dell'*hardware* e delle sue periferiche (`HARDWARE`) dei programmi installati (`SOFTWARE`) e con informazioni utili per l'inizializzazione (`SYSTEM`)

Architettura di sistema – 1

- Sistema su 2 livelli gerarchici
  - **Nucleo monolitico** i cui componenti eseguono tutti in modo privilegiato
    - Dipendenze dalla scheda madre dello specifico elaboratore isolate in un livello detto `HAL` (*hardware abstraction layer*)
      - Insieme **standard** di servizi di accesso a registri, indirizzi di periferiche, vettore delle interruzioni, orologi, BIOS
      - Poi affiancato da un interfaccia di maggior potenza e velocità detto `DirectX`
  - **Sottosistemi d'ambiente** visti come processi che eseguono in modo normale

Architettura di sistema – 2

- Su `HAL` poggia un livello detto `kernel` che eleva il livello di astrazione dei servizi `HAL`
  - **Gestione della concorrenza**
    - Ordinamento, prerilascio, salvataggio e ripristino dei contesti
  - **Gestione degli "oggetti di controllo"** associati alle entità attive del sistema
    - Processi e servizi associati alle interruzioni
    - Oggetto ***Deferred Procedure Call***: la parte meno urgente di un servizio di interruzione, che esegue in modo nucleo e non è interrompibile da *thread* e APC
      - Gestione dei dispositivi, dell'orologio di sistema, della fine quanto
    - Oggetto ***Asynchronous Procedure Call***: la parte immediate di un servizio di interruzione, che esegue entro un *thread* sia in modo nucleo che normale
  - **Gestione degli "oggetti di ordinamento"** associati a entità passive come semafori, eventi, orologi
    - Usati dalle entità attive per sincronizzarsi tra loro

Architettura di sistema – 3

- Il livello `executive` (il più alto del S/O) è suddiviso in 10 aggregati di **procedure** funzionalmente correlate

`Object manager`: gestisce gli "oggetti" creati dal S/O allocando loro memoria virtuale e fissando le loro operazioni (1)
`I/O manager`: gestisce i dispositivi incluse le partizioni di disco
`Process manager`: gestisce le entità concorrenti del sistema
`Memory manager`: gestisce la memoria virtuale con modalità "*paging-on-demand*"
`Cache manager`: gestisce in RAM una *cache* di blocchi di disco

Architettura di sistema – 4

- Solo Ⓐ e Ⓑ sono componenti attive
- Tutte però eseguono in modo nucleo

`Plug-and-play manager` (**A**): viene informato delle periferiche connesse al sistema e le associa il loro gestore (2)
`Power manager` (**B**): cerca di contenere il consumo energetico del sistema
`Configuration manager`: gestisce la `registry`
`Security manager`: si occupa dell'esecuzione delle politiche di sicurezza richiesti per applicazioni riservate
`Local procedure call manager`: fornisce meccanismi efficaci per la comunicazione tra le componenti attive del sistema

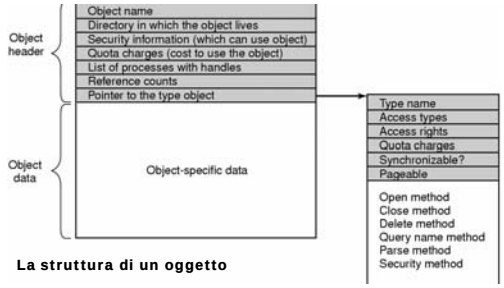
Architettura di sistema – 5

- Del livello **`executive`** fa parte anche il **GDI** che in in NT 3.x era posto in spazio di utente
  - *Graphics Design Interface*
  - Di gran lunga la componente più grande
  - Posto in modo nucleo da NT 4.0 per migliorare le prestazioni
    - La sua esecuzione può comportare frequente paginazione
- **`kernel`** ed **`executive`** sono raccolti in un unico **eseguibile** (`ntoskrnl.exe`)
- `HAL` viene invece fornito come **libreria condivisa** raccolta in un unico *file* (`hal.dll`)
- Gestori delle periferiche caricati **dinamicamente** e registrati in `registry` dal `Configuration manager`

Architettura di sistema – 6

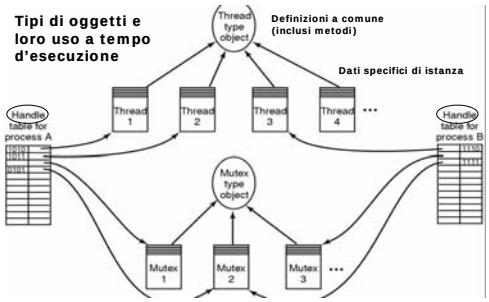
- Durante l'esecuzione il sistema crea, manipola e distrugge **oggetti interni** nessuno dei quali permane tra due accensioni successive
  - Un oggetto per ogni risorsa logica o fisica (entità attive o passive)
    - Tutti gli oggetti hanno alcuni metodi comuni
    - I loro tipi corrispondono alle **interfacce** dei linguaggi OO
  - Gli oggetti sono **descrittori** (residenti in RAM) delle entità logiche o fisiche corrispondenti
    - Alcuni oggetti possono essere temporaneamente posti su disco
    - I processi manipolano oggetti tramite riferimenti detti *handle*
- Il **`kernel`** mantiene una **tabella degli oggetti**
  - 29 *bit* per puntatore all'oggetto + 3 *bit* come *flag*
  - 32 *bit* per i diritti associati alle operazioni sull'oggetto
- L'`Object manager` suddivide gli oggetti in categorie (*directory*) specifiche

Architettura di sistema – 7



La struttura di un oggetto

Architettura di sistema – 8



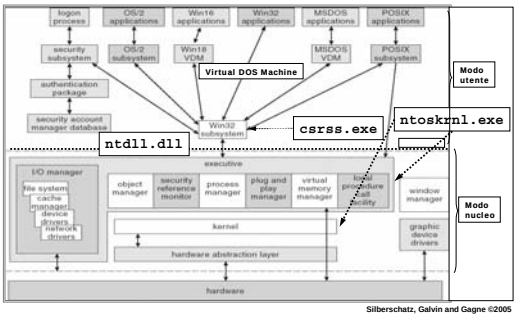
Architettura di sistema – 9

- In spazio di utente sono disponibili 3 ulteriori categorie di componenti di sistema
  - `DLL` (***Dynamic Link Libraries***) che raccolgono specifiche procedure di libreria in gruppi visibili ai e condivisi dai vari programmi
    - Ogni processo utente include **chiamate parametriche** a specifici `DLL` **al posto** del codice delle procedure richieste
  - **Sottosistemi d'ambiente** (`.exe`) che forniscono ciascuno uno specifico interfaccia di programmazione
    - Il principale è Win32 API, `csrss.exe`
      - *Client-server run-time subsystem*
    - Gli altri 2 (uno era per UNIX/POSIX) sono stati a lungo inutilizzati
    - Con Windows XP sono diventati elementi importanti della versione *server* del S/O
  - **Processi di servizio**

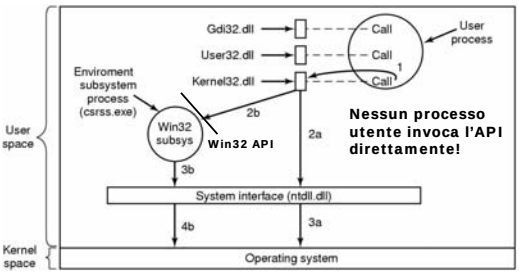
Architettura di sistema – 10

- Oltre 800 `DLL` complessivi per oltre 13.000 procedure invocabili dai processi utente
  - `user32.dll`
    - Invocate in **modo utente** per i servizi GUI
  - `gdi32.dll`
    - Invocate in **modo utente** per i servizi grafici di livello inferiore al GUI
  - `kernel32.dll`
    - Invocate in **modo utente** per tutti gli altri servizi
  - `ntdll.dll`
    - Il vero interfaccia di sistema tra modo utente e modo nucleo (`executive` e `kernel`)
  - `hal.dll`
    - Eseguite in modo nucleo per accedere all'*hardware* specifico dell'elaboratore

Architettura di sistema – 11



Architettura di sistema – 12



Gestione dei processi – 1

- *Job* = {processi gestiti come singola unità}
- Processo = possessore di risorse, con ≥ 1 *thread*

  ID unico, 4 GB di spazio di indirizzamento (2 in modo utente e 2 in modo nucleo), inizialmente con singolo *thread*, simile al processo UNIX; **non** ha stato di avanzamento
- *Thread* = flusso di controllo gestito dal nucleo

  Esegue per conto e nell'ambiente del processo (che **non** ha stato di avanzamento), con ID **localmente** unico, 2 *stack* (1 per modo)
- *Fiber* = suddivisione di *thread* ignota al nucleo

  Esegue nell'ambiente del *thread* e viene gestita interamente a livello di servizi offerti dal sottosistema **Win32**

Gestione dei processi – 2

- I *thread* hanno vari modi per sincronizzarsi tra loro tramite **oggetti di ordinamento**
  - **Semafori binari** (`mutex`) o contatori
  - **Sezioni critiche** limitate allo spazio di indirizzamento del *thread* che le crea
  - **Eventi** di 2 tipi
    - A *reset* manuale che rilascia più *thread* sino ad un esplicito *reset* che cancella l'evento
    - A *reset* automatico che rilascia solo un *thread* e poi cancella l'evento

Gestione dei processi – 3

- I *thread* hanno vari modi per comunicare senza bisogno di sincronizzarsi
  - *Pipe* : canali bidirezionali come in UNIX e GNU/Linux a sequenza di *byte* **senza** struttura oppure per messaggi (sequenze **con** struttura)
  - *Mailslot* : canali unidirezionali anche su rete
  - *Socket* : come *pipe* ma per comunicazioni remote
  - **RPC** (**chiamata di procedura remota**) : per invocare procedure nello spazio di altri processi e riceverne il risultato localmente
  - **Condivisione di memoria** : usando (porzioni di) *file* mappati in memoria
- Modi usati soprattutto per le interazioni tra le applicazioni e GDI

Gestione dei processi – 4

Politica di ordinamento – 1

- **Ordinamento con prerilascio a priorità**
  - Effettuato da azioni **esplicite** del *thread* eseguite in modo nucleo → non a carico di alcuna entità attiva dedicata di sistema
    - Nel sospendersi in attesa di una risorsa occupata o nell'inviare un segnale di sincronizzazione
      - L'esecuzione è già in modo nucleo
    - Al completamento del proprio quanto di tempo
      - L'esecuzione passa in modo nucleo tramite un **DPC**
  - Oppure causato da attività esterne eseguite nel contesto del *thread* corrente
    - Azioni di ordinamento programmate come **DPC** associate al trattamento di eventi asincroni (interruzione, allarme *time-out*) possono rilasciare *thread*

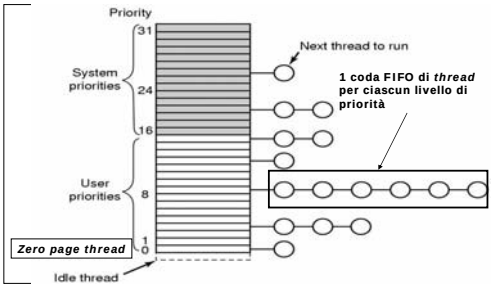
Politica di ordinamento – 2

- **6 classi di priorità per processo**
  - Realtime, high, above-normal, normal, below-normal, idle
- **7 classi di priorità per *thread***
  - Time-critical, highest, above-normal, normal, below-normal, lowest, idle
- **32 livelli di priorità (31 .. 0)**
  - Ciascuno associato a una coda di *thread* pronti
  - *Thread* **non** distinti per processo di appartenenza
    - 31 .. 16 priorità di sistema
    - 15 .. 0 priorità ordinarie
- Ricerca per priorità decrescente
- Selezione dalla testa della coda

Politica di ordinamento – 3

		Win32 process class priorities					
		Realtime	High	Above Normal	Normal	Below Normal	Idle
Win32 thread priorities	Time critical	31	15	15	15	15	15
	Highest	26	15	12	10	8	6
	Above normal	25	14	11	9	7	5
	Normal	24	13	10	8	6	4
	Below normal	23	12	9	7	5	3
	Lowest	22	11	8	6	4	2
	Idle	16	1	1	1	1	1

Politica di ordinamento – 4



Politica di ordinamento – 5

- Ciascun *thread* ha una priorità **base** iniziale e una priorità **corrente** che **varia** nel corso dell'esecuzione
  - Entro la fascia della classe di priorità del processo di appartenenza
- La priorità corrente si eleva quando il *thread*
  - Completa un'operazione di I/O
    - Per favorire maggior utilizzazione delle periferiche
    - Insieme a un ampliamento temporaneo della durata del quanto
  - Ottiene un semaforo o riceve un segnale d'evento
    - Per ridurre il tempo di attesa dei processi interattivi
- La priorità corrente decresce a ogni quanto consumato
- Usa una tecnica brutale per mitigare il problema di inversione di priorità
  - Un *thread* pronto non selezionato per un certo tempo riceve un incremento di priorità per 2 quanti

Inizializzazione – 1

- Sequenza di *boot* come in GNU/Linux
  - Lettura della struttura di FS
  - Localizzazione ed esecuzione del *file* **`ntldr`** che carica **Win NT**
    - Il FS può avere struttura FAT-16, FAT-32, **`NTFS`**
  - Lettura del *file* di configurazione **`Boot.ini`**
  - Caricamento di **`hal.dll`**, **`ntoskrnl.exe`** e **`bootvid.dll`**
  - Lettura di **`registry`** e configurazione delle periferiche
  - Attivazione di **`ntoskrnl.exe`** e creazione del gestore di sessione (processo utente **nativo** **`smss.exe`**, *session manager subsystem*) che si occupa di
    - Creazione del *daemon* di *login* (**`winlogon.exe`**)
    - Attivazione del gestore di autenticazione (**`lsass.exe`**)
    - Attivazione del capostipite di tutti i servizi (**`services.exe`**)

Inizializzazione – 2

- **`winlogon.exe`** usa un programma della libreria **`msgina.dll`** per eseguire la sequenza di *login* desiderata
  - L'uso di un programma di libreria rende la sequenza più facilmente configurabile dagli amministratori di sistema ma anche più misterioso
  - ***M**icros**o**ft **G**raphical **I**dentification a**n**d **a**uthentication*
- Poi preleva da **`registry`** il profilo d'utente da cui determina il programma di *shell* da eseguire
  - Generalmente si tratta di **`explorer.exe`** ma la scelta è configurabile tramite **`registry`**