Genesi – 1

- **MS-DOS**
  - Mono-utente, in modalità *command line*, non multi-programmata, inizialmente basato sul modello 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 ma 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*)

Genesi – 2

- **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*), menu 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, 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 (con 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, ma di grande complessità (16 M linee di codice!)
      - Molto superiore a Windows 95, 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 incapsulata in processi di sistema eseguiti in modo utente ed offerti ai processi applicativi in modalità a scambio messaggi
  - Elevata portabilità (dipendenze localizzate nel nucleo) ma scarsa velocità (poca esecuzione in modo privilegiato)
  - **Architettura di NT 4.0** a **nucleo monolitico**: servizi di sistema riposizionati entro il nucleo

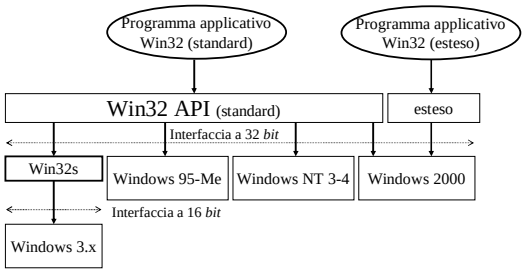
Genesi – 5

- **Windows** 3ª generazione (segue)
  - 1999 : Windows 2000 (alias di **NT 5.0**)
    - Il S/O esegue in modo nucleo, separato da quello dei processi utente, il cui spazio di indirizzamento è però interamente privato
    - Include supporto per periferiche rimuovibili (*plug-and-play*), per internazionalizzazione (unica versione configurabile per lingua nazionale) e alcune migliorie ad **`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
  - Vasta libreria pubblica di procedure detta **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 (**`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** che esegue 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
      - Recentemente 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
    - Oggetto ***Asynchronous Procedure Call***: come **DPC**, ma esegue in modo normale
  - **Gestione degli "oggetti di ordinamento"**, associati alle entità passive (semafori, eventi, orologi) usate dalle entità attive per sincronizzarsi

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, (1) allocando loro memoria virtuale
`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, cui 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

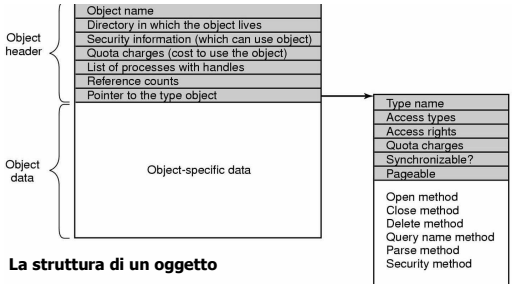
Architettura di sistema – 5

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

Architettura di sistema – 6

- Durante l'esecuzione il sistema crea, manipola e distrugge **oggetti interni**, nessuno dei quali permane tra due accensione successive
  - Un oggetto per ogni entità sia attiva che passiva
    - Tutti gli oggetti hanno alcuni metodi comuni
  - Gli oggetti sono **descrittori** (residenti in RAM) delle corrispondenti entità logiche e/o fisiche
    - Alcuni possono essere temporaneamente posti su disco
- 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

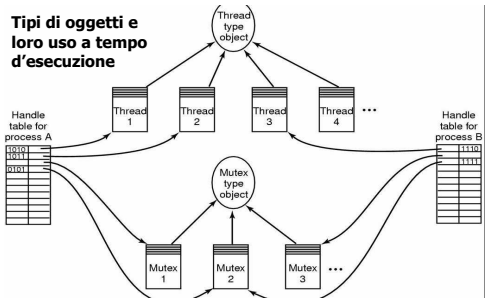
Architettura di sistema – 7



La struttura di un oggetto

Architettura di sistema – 8

Tipi di oggetti e loro uso a tempo d'esecuzione



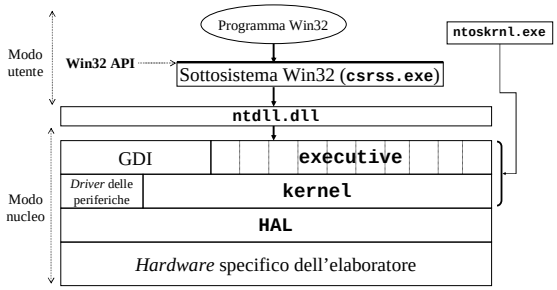
Architettura di sistema – 9

- 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 inutilizzati!
  - **Processi di servizio**

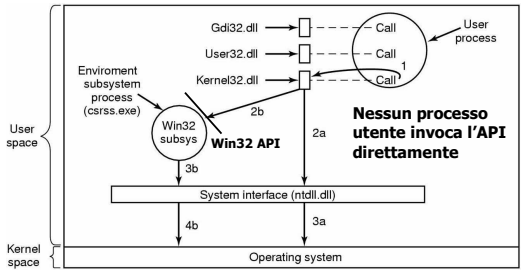
Architettura di sistema – 10

- In complesso, oltre 800 **`DLL`**, per più di 13.000 procedure invocabili dai processi utente, p.es.:
  - **`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 – 10



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 singola *thread*, simile al processo UNIX; non ha stato di avanzamento
- *Thread* = flusso di controllo ordinato 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 della *thread*, gestita interamente a livello di sottosistema Win32

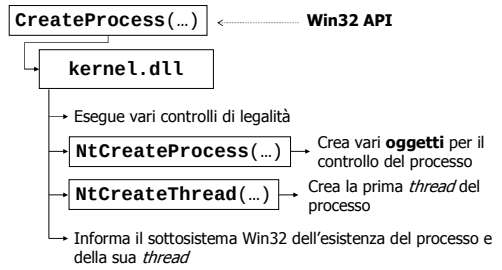
Gestione dei processi – 2

- Le *thread* hanno vari modi per comunicare
  - *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

Gestione dei processi – 3

- Le *thread* hanno vari modi per sincronizzarsi
  - **Semafori binari** (**`mutex`**) o contatori
  - **Sezioni critiche**, limitate allo spazio di indirizzamento della *thread* che la 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 una *thread* e poi cancella l'evento

Gestione dei processi – 4



Politica di ordinamento – 1

- **Ordinamento con prerilascio a priorità**
  - Effettuato da azioni esplicite della *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
  - Oppure causato da attività esterne eseguite nel contesto della *thread* corrente
    - Esecuzioni di *scheduler* programmate come DPC associate al trattamento di eventi asincroni (interruzione, allarme *time-out*) possono rilasciare *thread*

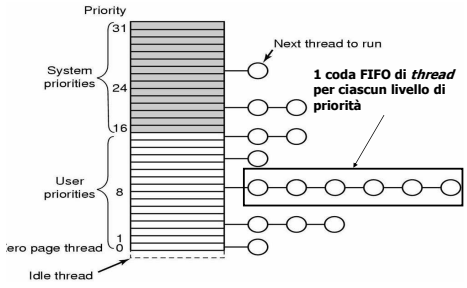
Politica di ordinamento – 2

- **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* pronte
  - *Thread* non distinte 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

- Ciascuna *thread* ha una priorità **base** (iniziale) e una **corrente** (che varia in esecuzione)
- La priorità corrente si eleva quando la *thread*
  - Completa un'operazione di I/O
    - Per maggior utilizzazione delle periferiche
  - Ottiene un semaforo o riceve un segnale d'evento
    - Per miglior risposta dei processi interattivi
- La priorità corrente decresce a ogni quanto consumato
- Usa una tecnica brutale per mitigare il problema di inversione di priorità
  - Una *thread* pronta non selezionata per una certa durata riceve 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`**)
    - 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
- 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`**