# 7 - Caso di Studio Windows 8

**Sommario**





## Obbiettivi

- Storia dei S.O. da MS-DOS a Windows
- Architettura di Windows
- Sottosistemi di Windows
- Gestione dei processi, thread, memoria in Windows 8
- File system NT in Windows 8
- Il sottosistema di I/O
- Rete e multiprocessing
- Modello di sicurezza



## Introduzione

- Introduzione a Windows
  - S.O. proprietario Microsoft
  - S.O. per server, commerciale e per utenti privati
  - S.O. desktop, tablet, smartphone
  - S.O. per infrastruttura cloud computing Azure
- molte edizioni, alcuni sono diversi s.o.
  - *Windows XP Home Edition* desktop
  - *Windows XP Professional* sicurezza
  - *Windows XP Tablet PC Edition* supporto wireless
  - *Windows XP Media Center Edition* multimedia
  - *Windows XP 64-Bit Edition* applicazioni per molti dati
  - *Windows XP 64-Bit Edition for 64-Bit Extended Systems*
  - *Windows 6 Vista, Windows 7 (2009), 8 (2012-2013), 10 (2016)*
  - *Windows Server 2008, 2012, 2016*



## Storia

- Quattro fasi

① MS-DOS – anni '80
② Windows basato su MS-DOS – anni ' 90
③ Windows basato su NT – anni 2000
④ Modern Windows – anni 2010
- 1976 Bill Gates (studente di Harvard) e Paul Allen (programmatore a Honeywell) fondano Microsoft
- Linguaggio BASIC – accordo per PC IBM – dal s.o. CP/M (Digital) il s.o.:

(1) 1981 MS-DOS 1.0
- Indirizzamento a 16-bit
- Modalità reale, utente singolo
- Un solo processo per volta
- Accesso diretto alla memoria principale

(2) Windows basato su MS-DOS - 1985 Windows 1.0
- Primo S.O. Microsoft **GUI**
- GUI già sviluppata da Xerox negli anni '70, commercializzata dal 1981 e in Macintosh Apple nel 1983-84
- Introduce la modalità protetta per DOS, ma sempre accesso diretto alla memoria

## Storia

(2) 1990 Windows 3.1 e Windows per Workgroups 3.1
- Eliminata la modalità reale, introdotta la modalità 'potenziata' (*enhanced*)
- Aggiunto il supporto di rete (LAN)
- Tentativi di miglioramento della stabilità, limiti alla compatibilità a ritroso

(3) 1993 Windows NT 3.1
- S.O. con nuova tecnologia (*New Technology*)
- Il processore di riferimento era Intel 860 detto N10
- Analogie con sistema VMS di DEC (*Digital Equipment Corporation*)
- Creata una nuova linea **aziendale**
- Centrato su sicurezza e stabilità
- NTFS (*New Technology File System*)
- Eseguito in uno spazio protetto
- Eliminato l'accesso diretto alla memoria
- Alcune applicazioni multimediali meno efficienti
- Indirizzamento a 32 bit
- Interfacce: estensioni a 32 bit delle API di Windows, chiamate Win32



## Storia

(2) 1995 Windows 95
- Linea **consumer**
- Indirizzamento a 32 bit
- Accesso diretto alla memoria principale
- Introduzione di DirectX
  - Simula l'accesso diretto all'hardware tramite API
- Multithreading, gestione dei processi

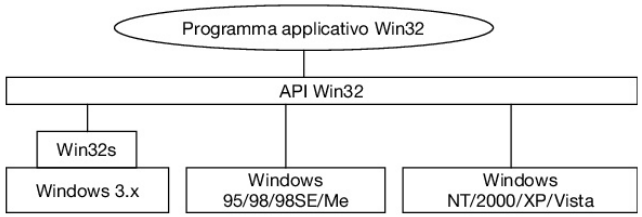
1998 Windows 98
- Fornisce Internet Explorer nel S.O.

2000 Windows ME (*Millenium Edition*)
- Ultima linea di S.O. desktop puramente consumer
- Miglior supporto multimediale e di driver
- Non si avvia in modalità DOS

(3) 1996 Windows NT 4.0
- Spostato il driver grafico nel nucleo
- Maggior sicurezza e supporto alla rete
- Interfaccia Windows 95



## Storia




Compatibilità: uso delle API Win32 che permette ai programmi di essere eseguiti su diverse versioni di Windows



## Storia

(3) 2000 Windows 2000
- L'ultima linea di S.O. desktop puramente aziendale
- Nucleo NT 5.0
- Plug and play
- Migliore GUI e gestione dell'energia
- Active Directory
  - Database di utenti, computer e servizi
- Sicurezza, autenticazione
- Kerberos
  - Consente firma singola sicura

2001 Windows XP
- Unisce codici per consumatore e aziende
- supporto a 64 bit
- GUI rinnovata

## Storia

- Da Windows XP in avanti architettura 32/64 bit
- Versioni per Smartphone e Tablet
- Windows Vista (versione 6.) 2006
- Windows Server 2008
- Windows 7 (versione 6.1) 2009
  - migliori prestazioni, affidabilità e sicurezza

(4) Windows 8 (versione 6.2) 2012
Windows 8.1 (versione 6.3) 2013
- migliore progettazione, codice meno ridondante
- installazione su diversi dispositivi

Windows 10 (versione 10.) 2016
- integrazione fra sistema desktop e sistema per smartphone e tablet



## Storia di Windows fino a Windows 8.1

| Anno | MS-DOS | Windows basato su MS-DOS | Windows basato su NT | Modern Windows | Note |
|---|---|---|---|---|---|
| 1981 | 1.0 | | | | Release iniziale per IBM PC |
| 1983 | 2.0 | | | | Supporto per PC/XT |
| 1984 | 3.0 | | | | Supporto per PC/AT |
| 1990 | | 3.0 | | | Dieci milioni di copie in due anni |
| 1991 | 5.0 | | | | Aggiunta gestione della memoria |
| 1992 | | 3.1 | | | Eseguito solo su 286 o superiori |
| 1993 | | | NT 3.1 | | |
| 1995 | 7.0 | 95 | | | MS-DOS integrato in Windows |
| 1996 | | | NT 4.0 | | |
| 1998 | | 98 | | | |
| 2000 | 8.0 | Me | 2000 | | Win Me era inferiore a Win 98 |
| 2001 | | | XP | | Sostituisce Win 98 |
| 2006 | | | Vista | | Vista non riesce a soppiantare XP |
| 2009 | | | 7 | | Miglioramenti sostanziali rispetto a Vista |
| 2012 | | | 8 | | Prima versione Modern |
| 2013 | | | 8.1 | | Microsoft passa a realease rapide |
| 2015 | | | 10 | | Ultima versione di Windows |

A. Tanenbaum – Modern Operating Systems

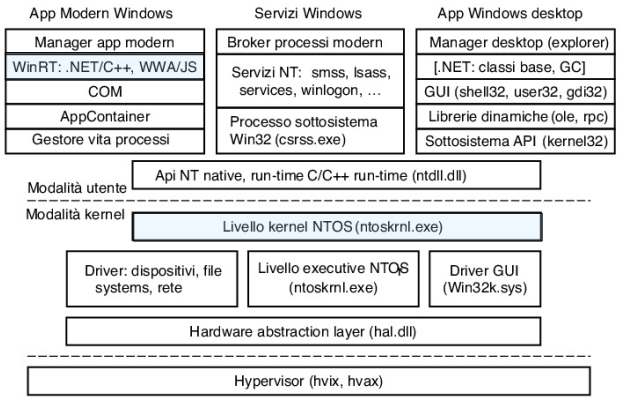


## Architettura di sistema − livelli di programmazione

- Sottosistema Win32 – API NT native per le chiamate di sistema
  - Interfacce di programmazione per creare applicazioni
  - Interfacce di programmazione implementate come servizi, chiamate da utente in modalità utente con RPC (*remote procedure call*)
- NTOS nucleo del S.O. NT che fornisce le tradizionali chiamate di sistema
- Le interfacce utente sono implementate dai sottosistemi che sono eseguiti sopra NTOS
- NT aveva tre personalizzazioni: OS/2 *(non in Windows XP*), POSIX (*non in Window 8.1*) e **Win32**
- Oggi tutte le applicazioni Windows sono scritte con API sopra Win32
- *Es. API WinFX nel modello di programmazione .NET*
- Modern Window, da Window 8 ha introdotto anche nuove API **WinRT**
  singola applicazione alla volta a tutto schermo, interfaccia a tocco
  Modern Software Development Kit (MSDK)
  e usa comunque alcune API Win 32 incluse in MSDK



## Architettura di sistema − livelli di programmazione




A. Tanenbaum – Modern Operating Systems

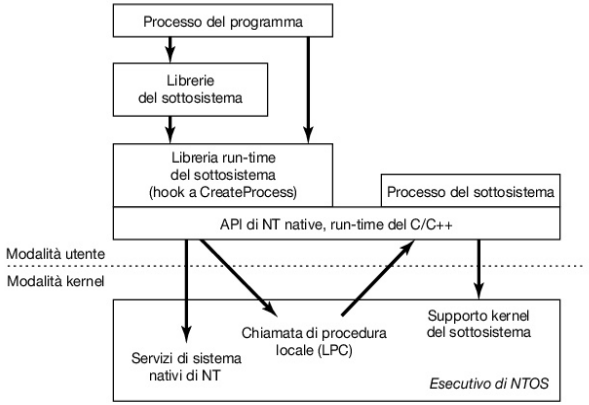
## Architettura di sistema – livelli di programmazione

- Nei sistemi desktop Win32 le applicazioni sono installate da un programma che fa parte della stessa applicazione
- Le applicazioni moderne devono essere installate dal programma AppStore di Windows dallo Store on line di Micorsoft
  - Quando in esecuzione l'applicazione moderna è eseguita in una cosiddetta *sandbox* (chiamata AppContainer) che isola il codice per motivi di sicurezza
  - Accesso alle risorse tramite api WinRT che comunicano con processi *broker* che accedono ad altre risorse, e.g. file utente

- Componenti dei sottosistemi NT
  - Processo del sottosistema (un servizio), avviato da *smss.exe* (manager di sessione a seguito della richiesta di `CreateProcess` in Win32
  - Librerie – funzioni di alto livello e funzioni di routine di comunicazione *stub*
  - *Hook* (agganci) a `CreateProcess` – quale sottosistema usare
  - Supporto nel kernel



## Architettura di sistema componenti per costruire sottosistemi NT




A. Tanenbaum – Modern Operating Systems



## Interfaccia di programmazione – applicazioni native NT

- Chiamate di sistema che Windows può eseguire, implementate nel livello Executive NTOS eseguito in modalità nucleo
- Le chiamate si riferiscono ad oggetti e restituiscono un *handle*, che possono essere usati e non possono di regola essere passati
- Descrittore di sicurezza degli oggetti per stabilire i diritti di accesso
- Gli oggetti in modalità nucleo hanno un nome, protezione, possono essere condivisi

- Alcuni tipi di oggetti utilizzati in modalità nucleo in Windows

| Categoria dell'oggetto | Esempi |
|---|---|
| Sincronizzazione | Semafori, mutex, eventi, porte IPC, code di completamento dell'I/O |
| I/O | File, dispositivi, driver, timer |
| Programma | Job, processi, thread, sezioni, token |
| Win32 GUI | Desktop, callback delle applicazioni |



## Interfaccia di programmazione - oggetti

Esempio di API native che usano *handle* espliciti per gestire oggetti in modalità nucleo (es. processi, thread, porte IPC, sezioni – per oggetti mappati in memoria)

| |
|---|
| NtCreateProcess(&ProcHandle, Access, SectionHandle, DebugPortHandle, ExceptPortHandle, ...) |
| NtCreateThread(&ThreadHandle, ProcHandle, Access, ThreadContext, CreateSuspended, ...) |
| NtAllocateVirtualMemory(ProcHandle, Addr, Size, Type, Protection, ...) |
| NtMapViewOfSection(SectHandle, ProcHandle, Addr, Size, Protection, ...) |
| NtReadVirtualMemory(ProcHandle, Addr, Size, ...) |
| NtWriteVirtualMemory(ProcHandle, Addr, Size, ...) |
| NtCreateFile(&FileHandle, FileNameDescriptor, Access, ...) |
| NtDuplicateObject(srcProcHandle, srcObjHandle, dstProcHandle, dstObjHandle, ...) |

Nota: in Unix si usano e accede agli oggetti con descrittori di file, PID, i-node
in Windows tramite handle
funzionalità uniforme, implementazione unificata con il gestore degli oggetti centralizzato – sincronizzazione, sicurezza
spazio dei nomi gerarchico ,

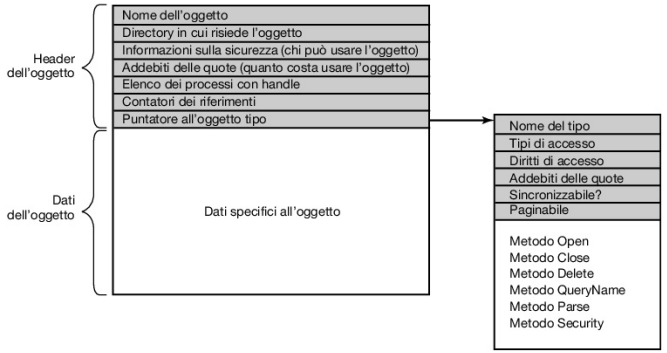
A. Tanenbaum – Modern Operating Systems

## Gestore degli Oggetti

- Oggetti in Windows (*Object*)
  - Rappresentano nomi di una
    - **risorsa fisica** (es.: periferica) o
    - **risorsa logica** (es.: processo)
  - ***N.B. NON*** *sono gli oggetti dei linguaggi di programmazione orientati ad oggetti*
  - Gestiti dal **Gestore di Oggetti** (*Object Manager*)
  - Rappresentato da una struttura dati in memoria
  - Associati ad un tipo
  - Un'istanza di tipo di oggetto
    - Definisce gli attributi dell'oggetto
    - Definisce le procedure standard dell'oggetto (es.: open, close, delete)
  - Esempi di tipi di oggetti: processi, thread, pipe, file, dispositivi, …



## Gestore degli Oggetti




Struttura di un oggetto – gestito dal gestore di oggetti

A. Tanenbaum – Modern Operating Systems



## Gestore degli Oggetti

- Handles e Pointers
  - Pointer
    - Puntatore referenziato per contare i riferimenti
  - Handle
    - Usati da processi in modalità utente e da componenti del nucleo
    - Permette il controllo del sistema su ciò che un thread può fare con l'oggetto (permessi)
    - Possono essere duplicati e passati ad altri processi
    - Handle di kernel: vi si può accedere dallo spazio di un processo, ma solo in modalità nucleo
    - Contenuti nella tabella di handle dei processi di sistema



## Gestore degli Oggetti

- Denominazione degli oggetti (*Object naming*)
  - Un oggetto può essere con o senza nome
  - Oggetti con nome classificati nello spazio dei nomi dell'Object Manager
  - solo i threads del nucleo possono aprire un *handle* per un oggetto senza nome
  - Spazio dei nomi gerarchico (NT namespace) ed estensibile
  - Il gestore degli oggetti implementa directory e link simbolici
  - Routine di gestione `Parse`
  - Nomi permanenti, l'oggetto rimane finché non è cancellato, anche se non collegato
- Eliminazione di oggetti
  - Niente più *handle*: oggetti cancellati dallo spazio dei nomi
  - Non più *handle* e puntatori: oggetti cancellati dalla memoria

## Interfaccia di programmazione – applicazioni native Win32

Chiamate di funzione Win32, sono chiamate API Win32

Esempi di chiamate API Win32 e relative chiamate API NT native

| Chiamata Win32 | Chiamata API NT nativa |
|---|---|
| CreateProcess | NtCreateProcess |
| CreateThread | NtCreateThread |
| SuspendThread | NtSuspendThread |
| CreateSemaphore | NtCreateSemaphore |
| ReadFile | NtReadFile |
| DeleteFile | NtSetInformationFile |
| CreateFileMapping | NtCreateSection |
| VirtualAlloc | NtAllocateVirtualMemory |
| MapViewOfFile | NtMapViewOfSection |
| DuplicateHandle | NtDuplicateObject |
| CloseHandle | NtClose |

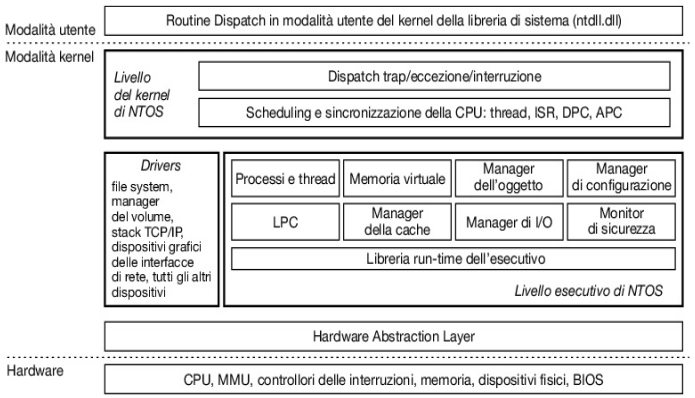


## Architettura di sistema

- Progettazione del S.O.
  - **Kernel NTOS** con due livelli
    - Esecutivo – maggior parte dei servizi
    - Kernel – Scheduling dei thread, interrupt dispatching
      - Meccanismi e servizi di base del sistema
      - Sincronizzazione dei thread,
      - Gestore delle trap, delle interruzioni
- Componenti
  - **HAL** (**Hardware Abstraction Layer**)
    - Interagisce con l'hardware, gestisce i dispositivi sulla scheda madre
    - Astrae dalle specifiche hardware che differiscono tra sistemi con la stessa architettura
  - Driver dei dispositivi
    - Controllo dei dispositivi periferici
  - Hypervisor
    - Livello più basso – schedula processori virtuali su processori fisici



## Architettura di sistema






## Architettura di sistema

- **DLL** (*Dynamic Link Libraries*)
  - Librerie di collegamento dinamico
  - Moduli in modalità utente che i processi possono collegare in modo dinamico
  - Forniscono dati e funzioni ai processi utente
  - Punti di ingresso per il nucleo per gestire eccezioni e APC: chiamate di procedura asincrona
- **Servizi** di sistema
  - Processi speciali eseguiti in modalità utente
  - Come i demoni in Linux: processi che vengono eseguiti in background costantemente
  - Es.: Task Scheduler, IPSec, Computer Browser (elenco di sistemi connessi alla rete locale), ecc

## Architettura di sistema

- **Livello kernel** (nel NTOS)
  - Sopra al livello executive
  - Astrazioni per la gestione della CPU: astrazione del thread
  - Scheduling e sincronizzazione di thread
  - Gestione di trap, interruzioni
  - Fornisce al livello più basso due meccanismi di sincronizzazione:
    - Oggetti di controllo – strutture dati per gestire la CPU
    - Oggetti dispatcher – oggetti esecutivi ordinari che usano una comune struttura dati per la sincronizzazione



## Architettura di sistema

- **Livello Executive**
  - Sotto al livello kernel di NTOS
  - Scritto in C
  - Eseguito in modalità kernel
  - Indipendente dall'architettura (eccetto gestore della memoria)
  - Componenti (chiamati gestori) con strutture dati interne ed esterne
    - Gestore di oggetti
    - Gestore di I/O
    - Gestore dei processi
    - Gestore della memoria
      - Memoria virtuale e paginazione su richiesta
    - Gestore della cache
    - Security reference monitor
    - Comunicazione: LPC, notifica WNF (Window Notification Facility)
    - Gestore della configurazione: implementa il registro di sistema



## Meccanismi di gestione del sistema

- Ambiente in cui vengono eseguite le componenti di *Windows*
  - Come sono conservati e recuperati i dati (registry)
  - Oggetti
  - Priorità di interrupt
  - Interrupt software e gestione delle priorità
  - Thread di sistema



## Oggetti gestiti nell'Executive − gestore di oggetti

| Tipo | Descrizione |
|---|---|
| Processo | Processo utente |
| Thread | Thread all'interno di un processo |
| Semaforo | Semaforo contatore usato per la sincronizzazione tra processi |
| Mutex | Semaforo binario usato per accedere a una regione critica |
| Evento | Oggetto di sincronizzazione con stato persistente (segnalato/non segnalato) |
| Porta ALPC | Meccanismo per il passaggio di messaggi tra processi |
| Timer | Oggetto che consente a un thread di dormire per un intervallo di tempo fissato |
| Coda | Oggetto usato per la notifica di completamento di I/O asincrono |
| File aperto | Oggetto associato a un file aperto |
| Token di accesso | Descrittore di sicurezza per un oggetto |
| Profilo | Struttura dati usata per profilare l'uso della CPU |
| Sezione | Oggetto usato per rappresentare file mappabili |
| Chiave | Chiave del registro, usata per attaccare il registro allo spazio dei nomi del gestore degli oggetti |
| Oggetto directory | Directory per raggruppare gli oggetti all'interno del gestore degli oggetti |
| Link simbolico | Si riferisce a un altro oggetto del gestore degli oggetti attraverso il path name |
| Dispositivo | Device object di I/O per un dispositivo fisico, bus, driver, o istanza del volume |
| Driver del dispositivo | Ciascun driver del dispositivo caricato ha il proprio oggetto |

A. Tanenbaum – Modern Operating Systems

## Registro di sistema - Registry

- **File** speciale - **database** centrale che memorizza i dati di configurazione
- accessibili a tutti i processi in modalità nucleo
  - dati di utente (es. impostazioni desktop)
  - dati di sistema (es. associazioni dei file)
  - dati hardware (es. driver dei dispositivi collegati)
  - dati per l'applicazione (es. voci di menù usate, preferenze)
- Struttura logica memorizzata come
  - Albero i cui nodi sono le chiavi
    - Sottochiavi e valori, nome e dato
    - Chiavi predefinite (es.: HKEY_LOCAL_MACHINE radice dei dati di configurazione dell'elaboratore locale)
  - Utilizzando il Registro di sistema
    - I thread navigano la struttura dell'albero passando dalle chiavi alle sottochiavi
    - GUI per utenti: `regedit` per esplorare le directory (chiavi) e dati (valori)



## Registro di sistema

- Windows amministratore del registro
  - Gestore delle configurazioni (*Configuration Manager)* componente executive
  - I dati memorizzati in insiemi di *hives* (alveari) – porzioni dell'albero del registro, volumi - , ciascun hive memorizzato in un file
    *nella directory C:\Windows\system32\config\ del volume di boot*
  - I programmatori di Win32 possono accedere al registro, con chiamate, esempi

| Funzione delle API Win32 | Descrizione |
|---|---|
| RegCreateKeyEx | Crea una nuova chiave di registro |
| RegDeleteKey | Cancella una chiave di registro |
| RegOpenKeyEx | Apre una chiave per ottenere un handle a essa |
| RegEnumKeyEx | Calcola le sottochiavi subordinate alla chiave dell'handle |
| RegQueryValueEx | Cerca tra i dati per trovare un valore all'interno di una chiave |



## Registro di sistema

Hive del registro di sistema di Windows

| File hive | Nome montato | Uso |
|---|---|---|
| SYSTEM | HKLM TEM | Informazioni di configurazione del SO, usate dal kernel |
| HARDWARE | HKLM DWARE | Hive nella memoria che registra l'hardware trovato |
| BCD | HKLM BCD* | Base di dati della configurazione di boot |
| SAM | HKLM | Informazioni dell'account dell'utente locale |
| SECURITY | HKLM URITY | Account di lsass e altre informazioni sulla sicurezza |
| DEFAULT | HKEY_USERS.DEFAULT | Hive di default per nuovi utenti |
| NTUSER.DAT | HKEY_USERS<user id> | Hive specifico dell'utente, mantenuto nella directory home |
| SOFTWARE | HKLM TWARE | Classi applicative registrate da COM |
| COMPONENTS | HKLM NENTS | Manifesti e dipendenze per i componenti del sistema |

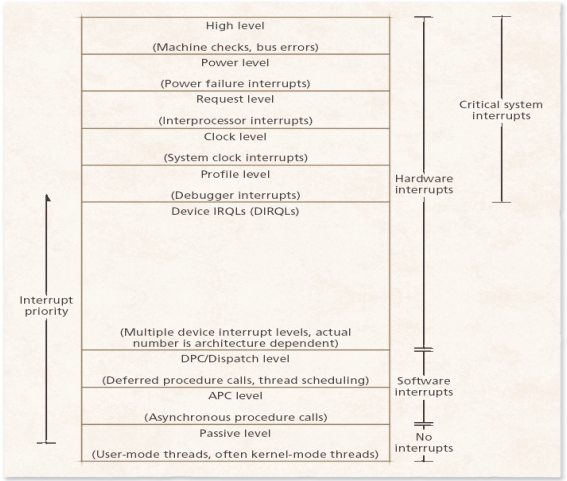
A. Tanenbaum – Modern Operating Systems



## Livelli di richiesta di Interrupt

- Interrupt Service Routine (ISR)
  - Per la gestione delle interruzioni
  - Parametro di priorità di interruzione
  - Il processore viene sempre in esecuzione con riferimento ad un IRQL (Interrupt Request Level)
  - Il processore maschera l'interrupt con IRQL ≤ IRQL corrente
- Livelli di priorità
  - Passivo: nessuna interrupt in fase di elaborazione — No Interrupt
  - APC: chiamate di procedura asincrona nel contesto di un thread — Interrupt software
  - DPC/dispatch: chiamate di procedura differite, scheduling dei thread – nel contesto di una CPU — Interrupt software
  - ISR di dispositivo: interrupt di periferica (dispositivo) — Interrupt Hardware
  - Interrupts di sistema critici: Profilo (debugger), Clock, Request (interprocessor), Power, High — Interrupt Hardware

## ISR - Interrupt Request Levels (IRQLs)






## Thread di Sistema

- Utilizzato da componenti in modalità nucleo che devono eseguire azioni non in risposta a una richiesta dell'utente
  - gestore della cache per lo scaricamento delle pagine di cache modificate (*dirty*)
  - driver di periferica che non si può soddisfare tutti gli interrupt a un IRQL elevato (es.: perché deve accedere ai dati paginabili)
- Due possibili approcci

1 - Thread del nucleo
- Creato da componenti in modalità nucleo
- In genere appartengono al processo di sistema
- In generale, si comportano e sono schedulati come thread utente
- Solitamente eseguiti a livello IRQL passivo



## Thread di Sistema

2 - System worker thread
- Forniti dal microkernel all'inizio o creati in base alla quantità di richieste
- Dormienti finché un componente in modalità nucleo accoda una richiesta di lavoro
- 3 tipi (diverse priorità): delayed, critical, hypercritical



## Gestione di Processi e Thread

- **Processi**
  - Contesto di esecuzione (spazio di indirizzamento virtuale, attributi,…)
  - Codice del programma
  - Risorse
  - Threads associati
- **Threads**: unità di esecuzione
- Processi e threads sono oggetti

## Gestione di Processi e Thread

| Nome | Descrizione | Note |
|---|---|---|
| Job | Raccolta di processi che condividono quote e limiti | Usato in AppContainers |
| Processo | Contenitore per le risorse | |
| Thread | Entità schedulata dal kernel | |
| Fibra | Thread leggero gestito interamente nello spazio utente | Usato raramente |
| Thread pool | Modello di programmazione orientato ai task | Costruito sui thread |
| Thread in modalità utente | Astrazione che consente lo scambio di thread in modalità utente | Estensione dei thread |

A. Tanenbaum – Modern Operating Systems



## Organizzazione di Processi e Thread

- Strutture dati che descrivono un processo
  - **Blocco di ambiente del Processo** (PEB):
    - informazioni del processo disponibili ai thread utente
    - DLL (Librerie) collegati al processo, informazioni heap, …
    - Memorizzato nello spazio del processo
  - Blocco EPROCESS (usato principalmente da executive):
    - Descrive un processo
    - ID del processo, token di accesso, tabella handle, etc.
    - Contiene un blocco KPROCESS
    - Punta a PEB
    - Memorizzato in una linked list
  - Blocco KPROCESS (info usate principalmente dal kernel):
    - informazioni di scheduling
    - informazioni di sincronizzazione



## Organizzazione di Processi e Thread

- Strutture dati che descrivono un thread
  - **Thread environment block** (TEB)
    - informazioni del thread disponibili ai thread utente
    - Sezioni critiche che appartengono al thread, stack information, …
    - Memorizzato nello spazio del processo al quale il thread appartiene
    - Memoria per il thread (Thread local storage)
    - Campi per Win32
  - Blocco ETHREAD (usato principalmente da executive)
    - Descrive un thread
    - ID del processo, richieste I/O sospese, indirizzo di inizio del thread
    - Contiene un blocco KTHREAD
    - Punta al blocco EPROCESS del processo del thread
  - Blocco KTHREAD (usato principalmente dal kernel)
    - Priorità di scheduling, stato del thread, …
    - Punta al blocco TEB



## Organizzazione di Processi e Thread

- I thread che appartengono ad un processo condividono lo spazio di indirizzamento virtuale
- Memoria locale del Thread (TLS)
  - Area dove i threads possono memorizzare dati locali (es.: per una DLL)
  - Un thread può allocare un indice TLS per il processo
  - L'indice TLS contiene uno *slot TLS* per ogni thread di un processo, puntatore alla locazione dei dati locali
  - Il processo può avere molti indici TLS per vari scopi (es. varie DDL e sottosistema ambiente)

## Organizzazione di Processi e Thread

- Creazione e terminazione di processi
  - Creazione di processi
    - chiamata di funzione API
    - Processo genitore e figlio sono indipendenti, eccetto che
    - Possibile ereditarietà per attributi del processo figlio dal processo padre (es. *handle*, variabili di ambiente)
    - Un processo ha un thread principale che può crearne altri interni al processo
  - Terminazione di processi
    - Se tutti i thread del processo terminano
    - Un thread di un processo dirige la terminazione
    - L'utente che possiede il processo si disconnette
    - Terminazione di processi genitore e figlio sono indipendenti



## Organizzazione di Processi e Thread

- Jobs
  - Gruppo di processi trattati insieme come una unità
  - Gestisce le risorse consumate da questi processi (es.: tempo di CPU, consumo di memoria, …) può vincolare con token limitati
  - Termina tutti i processi in una sola volta
  - Ogni processo può far parte di un solo job
  - Attributi (priorità base, sicurezza, dimensione del *working set* e memoria virtuale,..) del job ereditati dal processo
- Fibers
  - Unità di esecuzione (come un thread), possono essere schedulati dal thread che li crea, non dal kernel
  - Creata allocando uno stack e struttura dati in modalità utente
  - Eseguiti nel contesto del thread
  - Il thread si deve convertire in una fibra per creare fibre
  - Minor overhad di cambio contesto fra fibre
  - Usato per il *porting*
  - memorizzazione locale di fibra (*Fiber local storage,* FLS), come TLS



## Organizzazione di Processi e Thread

- Thread pooling
  - Thread di lavoro che dormono in attesa di lavoro
  - Ogni processo riceve un pool di thread
  - Utile in determinate situazioni
    - Tipicamente per processi serventi che devono soddisfare le richieste dei clienti
    - I/O asincrono
    - Combinazione di molti thread che dormono la maggior parte del tempo
  - Overhead di memoria e meno controllo per il programmatore
  - Funzionalità thread pool di Win32
  - Se un thread si blocca su una risorsa, il thread non può essere riassegnato ad un altro



## Scheduling di Thread

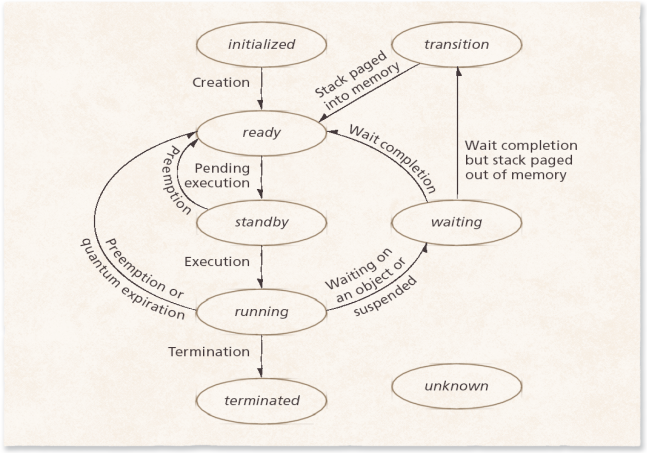
- Per ogni thread visto a livello utente
  - un thread eseguito in modalità kernel
  - un thread eseguito in modalità utente
- User-mode scheduling funzione di attivazione dello scheduler
  - Per passare tra thread utente senza attivare il kernel
  - Funzionalità di basso livello da usare nelle librerie run-time e applicazioni
- API Win32 per la gestione dei thread, fra cui
  - `SetPriorityClass` - imposta la classe di priorità dei thread del processo
  - `SetThreadPrioritys` - imposta la classe di priorità relativa di un thread

## Scheduling di Thread

- Stati del Thread
  - *Initialized*
  - *Ready* (eseguibile)
  - *Standby* (eseguibile e schedulato)
  - *Running* (in esecuzione)
  - *Waiting*
  - *Transition* (lo stack è stato spostato dalla memoria )
  - *Terminated* (terminato ma non cancellato)
  - *Unknown*
- Transizioni



## Scheduling di Thread






## Scheduling di Thread

- Thread sono schedulati senza considerare i processi
- 32 livelli di priorità
  - 0 priorità minima
  - 31 priorità massima
- 32 code *ready*
- *Round-robin* disciplina di coda per code ready non vuote a partire dalla massima priorità



## Scheduling di Thread

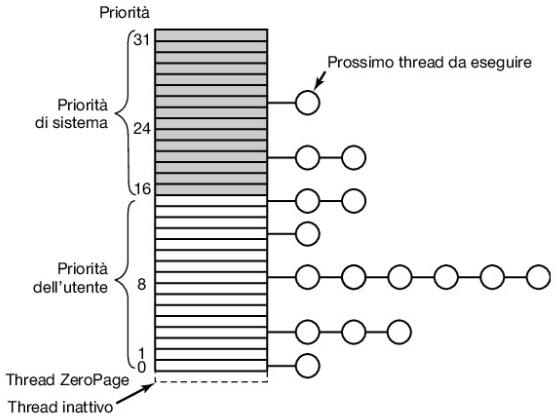
Mappa delle priorità dei thread Win32 nelle priorità di Windows

| | Classi di priorità dei processi Win32 | | | | | |
|---|---|---|---|---|---|---|
| | Real-time | Alta | Sopra il normale | Normale | Sotto il normale | Inattivo |
| Time critical | 31 | 15 | 15 | 15 | 15 | 15 |
| La più alta | 26 | 15 | 12 | 10 | 8 | 5 |
| Sopra il normale | 25 | 14 | 11 | 9 | 7 | 5 |
| Normale | 24 | 13 | 10 | 8 | 6 | 4 |
| Sotto il normale | 23 | 12 | 9 | 7 | 5 | 3 |
| La più bassa | 22 | 11 | 8 | 6 | 4 | 2 |
| Inattivo | 16 | 1 | 1 | 1 | 1 | 1 |

**Priorità dei thread Win32**

A. Tanenbaum – Modern Operating Systems

## Scheduling di Thread




Priorità per i thread – code run



## Thread Scheduling

- Prelazione
  - Può accadere quando:
    - Cambia la priorità del thread
    - Il thread entra nello stato *ready*
    - Il thread esce nello stato *running*
  - I thread che subiscono il prerilascio tornano in coda ready
  - Thread *real-time*: ottengono un nuovo quanto
  - Altri thread: non si riassegna il quanto



## Thread Scheduling

- Determinazione della priorità
  - Thread dinamici: livelli di priorità 0-15
  - Thread in tempo reale: livelli di priorità 16-31 - statica
  - Processo
    - classe base di priorità
    - Determina un intervallo limitato per le priorità di base dei suoi thread
    - Sei classi base
      - idle, below normal, normal, above normal, high, real-time
  - Thread
    - Livello base di priorità
      - idle, lowest, below normal, normal, above normal, highes, real-time
    - Determina l'esatta priorità di base all'interno di una classe di priorità
    - Priorità di base = Classe base di priorità + livello base di priorità



## Thread Scheduling

- Determinazione della priorità
  - Cambiamenti di priorità
    - Solo per thread dinamici
    - Aumento delle priorità
      - All'uscita dallo stato *wait*
      - Window riceve un input da utente
      - Non è stato eseguito da molto tempo
    - Riduzione priorità
      - Non può diminuire oltre la priorità di base
      - Decremento unitario se il thread viene eseguito per l'intero quanto
      - La priorità torna al livello precedente dopo un quanto nel caso in cui la priorità del thread è aumentata perché non era stato eseguito da molto tempo

## Thread Scheduling

- Supporta anche una modalità di Scheduling interattiva per Terminal server
  - protocollo Remote Desktop Protocol
- Implementa anche un algoritmo di Scheduling Fair-share
  - Dynamic Fair Share Scheduling
  - gruppi con diversi vincoli di risorse assegnate, priorità del gruppo



## Thread Scheduling

- Scheduling del Multiprocessor
  - Maschera di affinità insieme di processori dove poter eseguire il thread
  - Ultimo processor = processore sul quale il thread è stato eseguito l'ultima volta
  - Processore ideale
    - Massimizzare il parallelismo – threads in relazione, diversi processori ideali
    - Massimizzare cache hits - threads in relazione, stesso processore ideale
  - Dispatcher sceglie nella coda run non vuota con massima priorità basandosi sul tempo attesa, ultimo processore e processore ideale