

# Il sistema operativo Windows XP

Microsoft Windows XP è un sistema operativo a 32/64-bit che funziona in multitasking ed è preemptive per processori AMD K6/K7, Intel IA32/IA64 e versioni successive. Windows XP, successore di Windows NT/2000, ha inoltre l'obiettivo di sostituire il sistema operativo Windows 95/98; i suoi obiettivi basilari sono: sicurezza, affidabilità, facilità d'uso, compatibilità delle applicazioni tra WINDOWS e POSIX, prestazione elevata, estensibilità, portabilità e supporto internazionale. In questo capitolo, esamineremo gli obiettivi basilari di Windows XP: l'architettura stratificata del sistema, che lo rende di facile uso, il file system, il supporto di rete e l'interfaccia di programmazione.

## 1 Storia

A metà degli anni 80, Microsoft e IBM hanno collaborato per sviluppare il sistema operativo OS/2, che è stato scritto in assembler per sistemi a singolo processore Intel 80286. Nel 1988, Microsoft ha deciso di iniziare da zero e sviluppare "una nuova tecnologia" o NT, un sistema operativo portabile che supportasse le interfacce di programmazione delle applicazioni (application programming interface, API) sia di OS/2 che di POSIX. Nell'ottobre del 1988, Dave Cutler, progettista del sistema operativo DEC VAX/VMS, è stato assunto da Microsoft e gli è stato affidato l'incarico di progettare questo nuovo sistema operativo.

Inizialmente, il gruppo aveva pianificato che NT usasse, come ambiente nativo, le API di OS/2, ma, durante lo sviluppo, NT è stato cambiato per poter usare le API a 32 bit di Windows (Win32 API) in conseguenza della popolarità di Windows 3.0. Le prime versioni di NT furono Windows NT 3.1 e Windows NT 3.1 Advanced Server (a quel tempo, Windows a 16 bit era giunto alla versione 3.1). La versione Windows NT 4.0 ha adottato l'interfaccia utente di Windows 95 e incorporato il software per Internet web-server e web-browser. Inoltre, le procedure dell'interfaccia utente e tutto il codice grafico, furono spostati nel kernel al fine di migliorare le prestazioni, causando l'effetto collaterale di diminuire l'affidabilità del sistema. Sebbene le versioni precedenti di NT fossero state portate su altre architetture di microprocessori, la versione Windows 2000, rilasciata nel febbraio del 2000, ha interrotto il supporto ad altri processori esclusi quelli Intel (e compatibili) a causa di fattori di mercato. Windows 2000 ha incorporato significativi cambiamenti rispetto a Windows NT ed ha aggiunto l'Active Directory (un servizio di direttorio basato su X.500), un migliore supporto per le reti e per computer portatili, il supporto per dispositivi plug-and-play, un file system distribuito e supporto per più processori e maggior quantità di memoria.

Nell'ottobre 2001 fu rilasciato Windows XP sia come aggiornamento di Windows 2000 sia come rimpiazzo di Windows 95/98 e nel 2002 furono disponibili le versioni server di Windows XP (chiamate Windows .Net Server). Windows XP aggiorna l'interfaccia grafica dell'utente con un aspetto visivo che trae vantaggio dai più recenti miglioramenti dell'hardware e da molte nuove caratteristiche **di facile uso**. Sono state aggiunte numerose caratteristiche per riparare automaticamente problemi nelle applicazioni e nel sistema operativo stesso. Windows XP fornisce miglior supporto per la rete e per i dispositivi (incluse le reti wireless autoconfiguranti - zero-configuration wireless networks -, la messaggistica istantanea, i flussi multimediali, la fotografia e i

video digitali), miglioramenti sostanziali nelle prestazioni sia per computer desktop che per grandi sistemi multiprocessore, migliore affidabilità e sicurezza rispetto a Windows 2000.

Windows XP usa un'architettura client-server (come quella di Mach) per implementare personalità molteplici del sistema operativo, quali Win32 e POSIX, con processi a livello utente chiamati sottosistemi; l'architettura del sottosistema permette di apportare miglioramenti ad un aspetto del sistema operativo senza intaccare la compatibilità delle applicazioni altrui.

Windows XP è un sistema operativo multiutente, che supporta accessi simultanei tramite servizi distribuiti, o tramite istanze multiple all'interfaccia grafica utente (GUI), mediante un terminal server di Windows. La versione server di windows XP supporta sessioni contemporanee di terminal server da sistemi desktop Windows. Per ogni utente collegato, le versioni desktop del terminal server duplicano la tastiera, il mouse ed il monitor tra sessioni virtuali. Questa caratteristica, chiamata commutazione veloce dell'utente, permette agli utenti di appropriarsi a vicenda del PC senza dover uscire e rientrare nel sistema.

Windows XP è la prima versione di Windows con supporto a 64 bit. Il file system nativo di NT (NTFS) e molte delle API Win32 hanno sempre usato numeri interi a 64 bit, quando necessario; pertanto, l'estensione a 64 bit di Windows XP serve come supporto per indirizzi estesi.

Ci sono due versioni desktop di Windows XP: la versione Windows XP Professional è il sistema desktop di maggiore importanza per gli utenti, sia che lavorino a casa che in ufficio; per gli utenti solo domestici che emigrano da Windows 95/98, Windows XP Personal fornisce la stessa affidabilità e facilità d'uso di Windows XP, ma senza le caratteristiche più avanzate necessarie per lavorare in modo continuo (seamlessly) con Active Directory o con applicazioni POSIX.

I membri della famiglia Windows.Net Server usano gli stessi componenti del nucleo centrale delle versioni desktop con l'aggiunta di alcune caratteristiche necessarie per l'uso di webserver, di server per i file e la stampa, di sistemi cluster e per grandi macchine dei centri di elaborazione dati che possono raggiungere i 64 GB di memoria e 32 processori nei sistemi con IA32; 128 GB e 64 processori nei sistemi con IA64.

## 2 Principi progettuali

Gli obiettivi progettuali di Windows XP includono la sicurezza, l'affidabilità, la compatibilità tra le applicazioni Windows e POSIX, l'elevata prestazione, l'estensibilità, la portabilità ed il supporto internazionale.

### 2.1 Sicurezza

Gli obiettivi di **sicurezza** (security) di Windows XP richiedevano aderenza agli standard progettuali che hanno permesso a Windows NT 4.0 di ricevere la classificazione di sicurezza C-2 dal governo degli Stati Uniti d'America (che vuole dire un moderato livello di protezione nei confronti di software difettoso e di attacchi malevoli). L'estesa revisione del codice e le prove sono state combinate con strumenti di analisi automatizzati per identificare e studiare potenziali difetti che potrebbero rappresentare delle falle di sicurezza.

## **2.2 Affidabilità**

Fino a quel momento, Windows 2000 era il sistema operativo più affidabile e più stabile sviluppato da Microsoft e molta della sua affidabilità era dovuta alla maturità del codice sorgente, a prove intensive del sistema ed alla rilevazione automatica di errori gravi nei driver. I requisiti di **affidabilità** (reliability) nei riguardi di Windows XP furono ancora più stringenti; Microsoft ha avviato una vasta revisione del codice sia in modo manuale che automatico per identificare oltre 63.000 linee nel codice sorgente che potrebbero contenere problemi non rilevati dalle prove ed ha cominciato la revisione di ogni parte per verificare che il codice fosse effettivamente corretto.

Windows XP amplia il controllo dei driver per individuare i banchi più sottili, migliora la possibilità di rilevare errori di programmazione nel codice utente e nelle applicazioni di terzi e sottopone i driver ed i dispositivi ad un rigoroso processo di certificazione. Inoltre, Windows XP aggiunge nuove funzionalità di controllo sullo stato di salute del PC, compreso il download di aggiornamenti relativi a problemi, prima che vengano riscontrati dagli utenti. La sensazione di affidabilità di Windows XP è stata pure migliorata rendendo l'interfaccia grafica utente più facile da usare con un miglior aspetto visivo, con menu più semplici e miglioramenti calibrati verso la scoperta di come svolgere task comuni.

## **2.3 Compatibilità tra applicazioni Windows e Posix**

Windows XP non è solo un aggiornamento di Windows 2000: è un rimpiazzo di Windows 95/98. Windows 2000 si è focalizzato principalmente sulla compatibilità delle applicazioni commerciali, mentre Windows XP include una compatibilità molto più elevata con le applicazioni di largo consumo (consumer application) che funzionano in Windows 95/98. La **compatibilità delle applicazioni** (application compatibility) è difficile da ottenere perché ogni applicazione controlla la versione di Windows e può dipendere dalla capacità di realizzazione delle API e può avere dei banchi latenti nelle applicazioni mascherati nel sistema precedente o in altre dipendenze simili.

Windows XP introduce uno strato di compatibilità che si trova fra le applicazioni e le API di Win32 e tale strato rende la visione di Windows XP (all'incirca) compatibile baco per baco con le precedenti versioni di Windows. Windows XP, come le precedenti versioni di NT, mantiene il supporto per molte applicazioni a 16 bit usando uno strato di interfaccia che traduce le chiamate delle API a 16 bit in chiamate equivalenti a 32 bit. Similmente, la versione a 64 bit di Windows XP fornisce uno strato di interfaccia che traduce le chiamate delle API a 32 bit in chiamate native a 64 bit. Il supporto Posix in Windows XP è stato molto migliorato, ed è disponibile un nuovo sottosistema di POSIX chiamato Interix; la maggior parte del software compatibile con UNIX si compila e funziona sotto Interix senza alcuna modifica.

## **2.4 Prestazioni elevate**

Windows XP è progettato per fornire **prestazioni elevate** in sistemi desktop (che in gran parte sono vincolati dalle prestazioni I/O), in sistemi server (dove la CPU è spesso un collo di bottiglia) e in ambienti multithread e multiprocessore (dove la gestione dei blocchi e della cache sono punti chiave per la scalabilità). L'elevata prestazione è stata un obiettivo sempre più importante per Windows

XP. Windows 2000 con SQL 2000, montato su hardware Compaq, ha ottenuto valori elevati di TPC-C al momento della consegna.

Per soddisfare le richieste di prestazioni, NT usa parecchie tecniche quali I/O asincrono, protocolli ottimizzati per le reti (per esempio il blocco ottimista dei dati distribuiti, l'accodamento delle richieste), grafica nel kernel e tecniche sofisticate nell'uso della cache per il file system. Gli algoritmi di gestione della memoria e di sincronizzazione sono progettati con uno sguardo alle prestazioni legate alla linee di cache ed ai multiprocessori.

Windows XP ha ulteriormente migliorato le prestazioni riducendo la lunghezza del percorso del codice nelle funzioni critiche, usando algoritmi migliori e strutture dati dedicate al singolo processore, usando la coloratura della memoria per macchine NUMA (accesso non uniforme alla memoria) e implementando protocolli di blocco più scalabili, come le code di spinlock; i nuovi protocolli di lock contribuiscono a ridurre i cicli di bus del sistema, le liste senza lock e le code, l'uso di operazioni atomiche read-modify-write (come l'incremento interbloccato) ed altre tecniche avanzate di blocco.

I sottosistemi che costituiscono Windows XP comunicano tra loro in modo efficiente mediante una procedura locale di chiamata (LPC) che fornisce passaggio di messaggi ad alte prestazioni. Se non si esegue il dispatcher nel kernel, i thread nei sottosistemi di Windows XP possono acquisire diritto di precedenza tramite thread ad alta priorità, e pertanto il sistema risponde velocemente ad eventi esterni. Inoltre, Windows XP è progettato per multiprocessi simmetrici; in un computer multiprocessore, parecchi thread possono essere eseguiti contemporaneamente.

## **2.5 Estensibilità**

L'**estensibilità** (extensibility) si riferisce alla capacità di un sistema operativo di mantenersi aggiornato con lo sviluppo della tecnologia computazionale. In tal modo si facilitano i cambiamenti nel tempo; i progettisti hanno realizzato Windows XP usando un'architettura a strati. Il codice eseguibile di Windows XP funziona in modalità kernel o protetta e fornisce i servizi di base del sistema. In cima al codice eseguibile, parecchi sottosistemi server operano in modalità utente e fra questi vi sono i **sottosistemi di ambiente** (environmental subsystems) che emulano sistemi operativi differenti; pertanto programmi scritti per MSDOS, Microsoft Windows e POSIX funzionano tutti in Windows XP nell'ambiente appropriato. (consultare il Paragrafo 4 di questo capitolo per maggiori informazioni sui sottosistemi di ambiente). In virtù della struttura modulare, ulteriori sottosistemi di ambiente possono essere aggiunti senza alterare il codice eseguibile, inoltre, Windows XP usa driver caricabili nel sistema I/O e quindi, si possono aggiungere, durante il funzionamento del sistema, nuovi file system, nuovi tipi di dispositivi I/O ed i nuovi modelli di rete. Analogamente al sistema operativo Mach, Windows XP usa un modello client-server e supporta processi distribuiti mediante chiamate di procedura remota (RPC) come sono definite dall'Open Source Foundation.

## **2.6 Portabilità**

Un sistema operativo è **portabile** (portable) se può essere spostato da un'architettura hardware ad un'altra con relativamente pochi cambiamenti. Windows XP è progettato per essere portabile.

Come il sistema operativo di UNIX, la maggior parte del sistema è scritta in C ed in C++. La maggior parte del codice, dipendente dal processore, è isolata in una libreria di collegamento dinamico (DLL), chiamata **strato di astrazione hardware** (hardware-abstraction layer: HAL). Una DLL è una file che è mappato nello spazio degli indirizzi del processo in modo che qualsiasi funzione nella DLL sembri far parte del processo. Gli strati superiori del kernel di Windows XP dipendono dalle interfacce HAL piuttosto che dall'hardware sottostante, realizzando la portabilità di Windows XP. HAL manipola l'hardware direttamente, isolando il resto di Windows XP dalle differenze hardware fra le piattaforme in cui funziona.

Anche se per ragioni di mercato, Windows 2000 veniva consegnato solo per piattaforme compatibili con Intel IA32, è stato anche provato, prima del rilascio, su piattaforme IA32 e DEC Alpha per assicurare la portabilità. Windows XP funziona su processori IA32 compatibili e IA64. Microsoft riconosce l'importanza dello sviluppo e della prova su multipiattaforme, poiché, dal punto di vista pratico, mantenere la portabilità è una questione di *usarlo o perderlo*.

## 2.7 Supporto Internazionale

Windows XP è pure progettato per un uso **internazionale** e **multinazionale**, fornisce supporto per differenti lingue locali tramite **API di supporto al linguaggio nazionale** (national-language-support API: NLS). L'API NLS fornisce procedure specializzate al formato della data, dell'ora ed alla moneta corrente in conformità con gli usi nazionali. I confronti di stringhe sono particolarizzati per tener conto dei vari insiemi di caratteri alfabetici. UNICODE è il codice di carattere nativo di Windows XP che supporta anche i caratteri ANSI convertendoli in UNICODE prima di manipolarli (conversione da 8 bit a 16 bit). Le stringhe di testo del sistema sono mantenute nei file di risorsa che si possono sostituire per particolarizzare il sistema nei differenti linguaggi; più lingue locali si possono usare contemporaneamente, il che è importante per persone multilingui e nel campo degli affari.

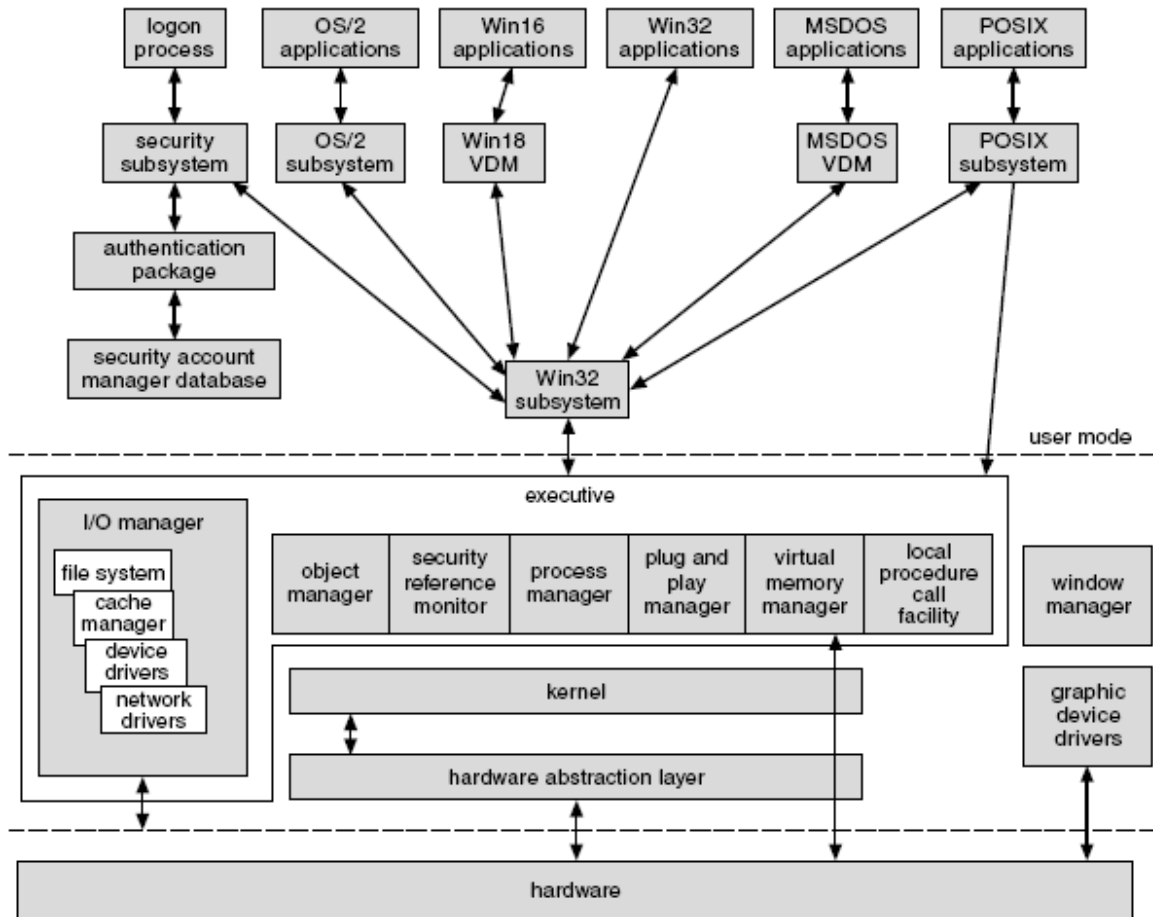
## 3 Componenti di Sistema

L'architettura di Windows XP è un sistema stratificato di moduli, come si vede in Figura 1. Gli strati principali sono HAL, il kernel e il codice eseguibile che funzionano in modo protetto e una collezione di sottosistemi e di servizi che funzionano in modalità utente. I sottosistemi in modalità utente si dividono in due categorie: sottosistemi di ambiente che emulano differenti sistemi operativi ed i **sottosistemi di protezione** (protection subsystems) che forniscono funzioni di sicurezza. Uno dei principali vantaggi di questo tipo di architettura è che le interazioni fra moduli sono mantenute semplici. Il resto di questo paragrafo descrive questi strati e sottosistemi.

### 3.1 Strato di astrazione dell'hardware

HAL è lo strato del software che nasconde le differenze hardware tra i livelli superiori del sistema operativo per aiutare a rendere portabile Windows XP. HAL esporta un'interfaccia di macchina virtuale che viene usata dal dispatcher del kernel, dal codice eseguibile e dai driver del dispositivo.

Un vantaggio di questo metodo è che viene richiesta solo una singola versione di ogni dispositivo driver: funziona su tutte le piattaforme hardware senza portare il codice del driver. HAL inoltre fornisce supporto per multiprocessi simmetrici; i driver delle periferiche mappano i dispositivi e vi accedono direttamente, ma i dettagli amministrativi della mappatura della memoria, dei bus di configurazione di I/O, della preparazione del DMA e la risposta a specifiche richieste della motherboard sono forniti dalle interfacce di HAL.



**Figura 1.** Diagramma a blocchi di Windows XP.

- logon process = processo di logon
- OS/2 applications = applicazioni OS/2
- Win16 applications = applicazioni Win16
- Win32 applications = applicazioni Win32
- MSDOS applications = applicazioni MSDOS
- POSIX applications = applicazioni POSIX
- security subsystem = sottosistema di sicurezza

OS/2 subsystem =sottosistema OS/2  
Win18 VDM  
MSDOS VDM  
POSIX subsystem =sottosistema POSIX  
authentication package = pacchetto di autenticazione  
security account manager database = gestore del database dell'account di sicurezza  
Win32 subsystem = sottosistema Win32  
user mode = modo utente  
executive = esecutivo  
I/O manager = gestore I/O  
file system  
cache manager = gestore della cache  
device drivers = dispositivi driver  
network drivers =driver di rete  
object manager = gestore degli oggetti  
security reference monitor = monitor di riferimento della sicurezza  
process manager = gestore di processi  
plug and play manager = gestore di plug and play  
virtual memory manager = gestore della memoria virtuale  
local procedure call facility = funzionalità di chiamata della procedura locale  
window manager = gestore delle finestre  
kernel  
graphic device manager = gestore dei dispositivi grafici  
hardware abstraction layer = strato di astrazione dell'hardware

## **3.2 Kernel**

Il kernel di Windows XP fornisce le fondamenta per il codice eseguibile e per i sottosistemi, esso rimane in memoria e l'esecuzione non viene mai interrotta; il kernel è responsabile di quattro attività principali: schedulazione dei thread, gestione degli interrupt e delle eccezioni, sincronizzazione a basso livello del processore e recupero dopo un'interruzione dell'alimentazione.

Il kernel è orientato agli oggetti. Un *tipo di oggetto* in Windows 2000 è un tipo di dati definito dal sistema che possiede un insieme di attributi (valori dei dati) e un insieme di metodi (per esempio, funzioni od operazioni). Un *oggetto* è un'istanza di un tipo di oggetto. Il kernel esegue il proprio job usando un insieme di oggetti del kernel, i cui attributi memorizzano i dati del kernel ed i cui metodi eseguono le attività del kernel.

### **3.2.1 Kernel dispatcher**

Il kernel dispatcher fornisce le fondamenta per il codice eseguibile e per i sottosistemi; la maggior parte del dispatcher non è mai paginato fuori dalla memoria e la sua esecuzione non viene mai interrotta. Le responsabilità principali sono: schedulazione dei thread, realizzazione delle primitive di

sincronizzazione, gestione del timer, interrupt software (chiamate asincrone e chiamate di procedura rinviate) e la spedizione di eccezioni.

### 3.2.2 Thread e schedulazione

Come in molti sistemi operativi moderni, Windows XP usa i concetti di processi e thread per il codice eseguibile. Il processo ha uno spazio di indirizzamento di memoria virtuale e informazioni usate per inizializzare ogni thread, quali una priorità di base ed un'affinità per uno o più processori. Ogni processo ha uno o più thread, ognuno dei quali è un'unità eseguibile gestita dal kernel. Ogni thread ha un proprio stato di schedulazione, compresa la priorità attuale, l'affinità del processore e le informazioni sull'uso della CPU.

I sei possibili stati del thread sono: pronto, standby, funzionante, in attesa, in transizione e terminato. **Pronto** (ready) indica che è in attesa di funzionare. Il thread pronto con priorità più alta viene spostato verso la condizione **standby**, che significa che sarà il thread successivo ad entrare in funzione. In un sistema multiprocessore, ogni processo mantiene un thread in condizione di standby. Un thread è **funzionante** (running) quando opera su di un processore e funziona fino a che non è interrotto da un thread a priorità più alta che lo fa terminare: ossia, finisce il proprio tempo di esecuzione (**quantum**), o si blocca su di un oggetto del dispatcher, quale un evento che segnala il completamento di un'operazione di I/O. Un thread è in condizione di **attesa** (waiting) quando sta aspettando un segnale di un oggetto del dispatcher. Un nuovo thread è in condizione di **transizione** (transition) quando aspetta risorse necessarie per l'esecuzione. Un thread è nella condizione **terminato** (terminated) quando finisce l'esecuzione.

Il dispatcher usa uno schema di priorità a 32 livelli per stabilire l'ordine di esecuzione dei thread. Le priorità sono divise in due classi: classe variabile e classe in tempo reale. La classe variabile contiene i thread con priorità da 0 a 15, la classe in tempo reale contiene i thread con priorità da 16 a 31. Il dispatcher usa una coda per ogni priorità di schedulazione ed attraversa l'insieme delle code dalla più alta alla più bassa finché non trova un thread pronto a funzionare. Se un thread ha una particolare affinità di processore, ma quel processore non è disponibile, il dispatcher passa oltre e continua a cercare un thread pronto che può funzionare su di un processore disponibile. Se non trova alcun thread pronto, il dispatcher esegue un thread speciale chiamato thread inattivo (idle thread).

Quando il quantum di tempo del thread si esaurisce, l'interrupt dell'orologio mette in coda un quantum di fine DPC al processore per rischedulare il processore stesso. Se il thread interrotto è nella classe a priorità variabile, la sua priorità viene abbassata, mai sotto quella base. L'abbassamento della priorità del thread serve a limitare il consumo della CPU dei thread con prevalente attività computazionale (compute bound). Quando un thread a priorità variabile viene rilasciato da un'operazione di wait, il dispatcher amplifica la priorità. La quantità di amplificazione dipende dal dispositivo che il thread stava attendendo; per esempio, un thread in attesa di un'operazione di I/O da tastiera avrebbe un grande incremento di priorità, mentre un thread in attesa di un'operazione su disco avrebbe un incremento modesto. Questa strategia tende a dare buoni tempi di risposta ai thread interattivi che usano il mouse e le finestre, e permette a thread collegati ad attività di I/O di tenere i dispositivi I/O occupati, mentre consente ai thread collegati ad attività di elaborazione di utilizzare cicli sparsi di CPU in background. Questa strategia è usata da parecchi sistemi operativi in time-sharing, compreso UNIX; inoltre il thread associato con la finestra attiva GUI dell'utente riceve un'amplificazione della priorità per migliorare il proprio tempo di risposta.

La schedulazione si presenta quando un thread entra nello stato pronto o di attesa oppure quando un thread termina, o se un'applicazione cambia la priorità del thread o l'affinità del processore. Se un thread, in tempo reale con priorità più alta, diviene pronto, mentre è in funzione un thread a priorità più bassa, quello a priorità più bassa viene interrotto e tale interruzione dà al thread un accesso preferenziale, in tempo reale, alla CPU, quando il thread ha bisogno di eseguire un tale accesso. Windows XP non è un sistema operativo hard real time, poiché non garantisce che un thread in tempo reale inizi l'esecuzione entro un particolare limite di tempo.

### **3.2.3 Realizzazione delle primitive di sincronizzazione**

Le principali strutture dati del sistema operativo sono gestite come oggetti che usano funzionalità comuni per l'allocazione, il conteggio dei riferimenti e la sicurezza. Gli **oggetti del dispatcher** (dispatcher object) controllano l'invio e la sincronizzazione nel sistema; esempi di questi oggetti sono gli eventi, i mutanti, i mutex, i semafori, i processi, i thread ed i temporizzatori. L'**evento oggetto** (event object) viene usato per registrare il verificarsi di un evento e per sincronizzare questo ultimo con una qualche azione. Gli eventi di notifica segnalano tutti i thread in attesa, mentre gli eventi di sincronizzazione segnalano un singolo thread in attesa. Il **mutante** (mutant) fornisce la mutua esclusione in modalità kernel o utente con la nozione di proprietà. Il **mutex**, disponibile soltanto in modalità kernel, fornisce mutua esclusione senza stalli. Un **oggetto semaforo** (semaphore object) opera come un contatore o porta (gate) per controllare il numero di thread che accedono alla risorsa. L'**oggetto thread** (thread object) è l'entità che è schedulata dal dispatcher del kernel ed è associata con un **oggetto processo** (process object) che incapsula uno spazio di indirizzamento virtuale. Gli **oggetti timer** (timer objects) sono usati per tenere traccia del tempo e per segnalare la mancata sincronizzazione quando le operazioni impiegano troppo tempo e devono essere interrotte, oppure deve essere schedulata un'attività ripetitiva.

A molti degli oggetti del dispatcher si può accedere in modalità utente tramite un'operazione `open` che ritorna un handle; il codice in modalità utente controlla/ o aspetta gli handle per sincronizzarsi con gli altri thread come pure con il sistema operativo (consultare il Paragrafo 7.1 di questo capitolo).

### **3.2.4 Software interrupt: chiamate asincrone di procedure**

Il dispatcher implementa due tipi di interrupt software: chiamate di procedure asincrone e chiamate di procedure rinviate (deferred). La chiamata di procedura asincrona (APC) interrompe un thread in esecuzione e chiama una procedura. Le APC sono usate per cominciare l'esecuzione di un nuovo thread, per terminare i processi e per consegnare la notifica che un'operazione di I/O asincrona è stata completata. Gli APC sono accodati a specifici thread e permettono al sistema di eseguire sia il codice di sistema che quello utente entro il contesto del processo.

### **3.2.5 Interrupt software: chiamate di procedura rinviate**

Le chiamate di procedura rinviate (DPC) si usano per posporre il processo di interrupt. In seguito alla gestione di tutti i processi bloccati da interrupt dei dispositivi, la procedura di servizio degli

interrupt (ISR) schedula il processo rimanente accodando un DPC. Il dispatcher schedula gli interrupt software ad una priorità più bassa di quella degli interrupt del dispositivo, in modo che la DPC non blocchi altri ISR.

Inoltre per rinviare l'elaborazione degli interrupt dei dispositivi, il dispatcher usa le DPC per elaborare l'esaurimento del tempo del timer e per impedire l'esecuzione del thread alla fine del quantum di schedulazione.

L'esecuzione di DPC impedisce ai thread di venire schedulati nel processore corrente e anche all'APC di segnalare il completamento di un'operazione di I/O; ciò affinché le procedure DPC non impieghino una quantità di tempo notevole per essere completate. In alternativa, il dispatcher mantiene un gruppo di thread al lavoro; ISR e DPC accodano il lavoro ai thread lavoratori. Le procedure DPC sono limitate in modo che non possano compiere dei page fault, dei servizi di chiamata del sistema o qualsiasi altra azione che potrebbe dar luogo ad un tentativo di bloccare l'esecuzione di un oggetto del dispatcher. A differenza delle APC, le procedure DPC non fanno presunzioni su quale sia il contesto di processo in cui il processore sta operando.

### **3.2.6 Eccezioni ed interruzioni**

Il dispatcher del kernel fornisce la gestione di trap per eccezioni ed interrupt generati dall'hardware o dal software. Windows XP definisce parecchie eccezioni indipendenti dall'architettura, tra cui: la violazione di accesso alla memoria, il superamento della capacità per numeri in virgola mobile o sotto la capacità minima, divisione di un numero intero per zero, divisione in virgola mobile per zero, istruzioni illegali, disallineamento dei dati, istruzioni privilegiate, errore di lettura di pagina, violazione di accesso, superamento della quota di impaginazione, punti di interruzione del debugger, e debugger a passo singolo.

I gestori di trap hanno a che fare con semplici eccezioni. La gestione di eccezioni elaborate è affidata al dispatcher delle eccezioni del kernel. L'**exception dispatcher** (dispatcher di eccezioni) crea un record dell'eccezione che contiene il motivo dell'eccezione stessa e trova un gestore di eccezione per gestirla.

Quando si presenta un'eccezione in modalità kernel, l'exception dispatcher chiama semplicemente una procedura per localizzare il gestore dell'eccezione; se non lo trova, segnala un errore fatale di sistema e l'utente è lasciato con il malfamato "schermo blu di morte" (blue screen of death) che indica un fallimento del sistema.

Il trattamento delle eccezioni è più complesso per i processi in modalità utente, perché un sottosistema ambiente (quale il sistema POSIX) installa una porta per il debugger e una porta di eccezione per ogni processo creato. Se una porta per il debugger viene registrata, il gestore di eccezione invia l'eccezione alla porta. Se non trova la porta del debugger o non è in grado di gestire quella eccezione, il dispatcher cerca di trovare un gestore adatto all'eccezione; se non trova un gestore, chiama di nuovo il debugger per trattare l'errore tramite debugger. Se un debugger non è in funzione, un messaggio viene trasmesso alla porta di eccezione del processo per fornire al sottosistema di ambiente una possibilità di tradurre l'eccezione. Per esempio, l'ambiente POSIX traduce i messaggi di eccezione di Windows XP in segnali POSIX prima di inviarli al thread che ha causato l'eccezione. Alla fine, se non funziona alcunché, il kernel termina semplicemente il processo che contiene il thread che ha causato l'eccezione.

Il dispatcher di interrupt nel kernel gestisce gli interrupt chiamando o una procedura di servizio dell'interrupt (ISR), fornita da un driver del dispositivo, o una procedura di gestione della trap del kernel.

L'interrupt è rappresentato da un oggetto interrupt che contiene tutte le informazioni necessarie per la gestione. L'uso di un oggetto interrupt rende facile associare procedura di servizio dell'interrupt con un interrupt senza dovere accedere all'hardware dell'interrupt direttamente.

Le varie architetture di processore, quali Intel o DEC Alpha, hanno differenti tipi e numeri di interrupt. Ai fini della portabilità, il dispatcher dell'interrupt mappa gli interrupt hardware in un insieme standard. Gli interrupt sono dotati di priorità e sono serviti in ordine di priorità. In Windows XP, ci sono 32 livelli di richiesta di interrupt (**IRQL**): otto sono riservati ad uso del kernel mentre i rimanenti 24 rappresentano interrupt hardware gestiti tramite HAL, sebbene la maggior parte dei sistemi IA32 ne usi solo 16. Gli interrupt di Windows XP sono definiti in Figura 2.

Il kernel usa una **tabella di dispatch degli interrupt** (interrupt-dispatch table) per collegare ogni livello di interrupt ad una procedura di servizio. In un sistema multiprocessore, Windows XP mantiene una tabella separata per ogni processore, e l'IRQL di ogni processore può essere regolato indipendentemente per mascherare gli interrupt. Tutti gli interrupt che si presentano ad un livello uguale o inferiore a quello degli IRQL di un processore vengono bloccati finché l'IRQL non viene abbassato da un thread a livello kernel o da un ISR di ritorno da un processo di interrupt. Windows XP trae vantaggio da questa proprietà e usa gli interrupt software per inviare l'APC e il DPC per eseguire funzioni di sistema quali la sincronizzazione di thread con completamento di I/O, per iniziare spedizioni di thread ed per gestire i timer.

livelli di interrupt	tipi di interrupt
31	controllo della macchina o errore del bus
30	manca di alimentazione
29	notifica interprocessore (richiede un altro processore per operare; per esempio, spedire un processo o aggiornare la TLB)
28	orologio (utilizzato per tenere traccia del tempo)
27	profilo
3-26	interrupt hardware degli IRQ del PC tradizionale
2	chiamata di procedura del dispatch e rinviata (DPC) (kernel)
1	chiamata asincrona di procedura (APC)
0	passivo

**Figura 2.** Livelli di richiesta di interrupt di Windows XP.

### 3.3 Codice eseguibile

Il codice eseguibile di Windows XP fornisce un insieme dei servizi usati da tutti i sottosistemi di ambiente; tali servizi sono raggruppati nel modo seguente: gestore dell'oggetto, gestore di memoria

virtuale, gestore di processo, chiamata di procedura locale, gestore di I/O, controllo di riferimento di sicurezza, gestori plug-and-play e di sicurezza, registro ed avvio.

### 3.3.1 Gestore dell'oggetto

Windows XP usa un insieme generico di interfacce per la gestione di entità in modalità kernel che vengono manipolate dai programmi in modalità utente e le chiama *oggetti* (objects) ed il componente di codice eseguibile che li gestisce è il **gestore degli oggetti** (object manager). Ogni processo ha una tabella degli oggetti contenente gli elementi che controllano gli oggetti usati dal processo. Il codice in modalità utente accede a questi oggetti tramite un valore detto *handle* (maniglia) che è restituito da molte API. Gli handle dell'oggetto possono anche essere creati duplicando un handle esistente sia attraverso lo stesso processo o tramite un differente processo.

Esempi di oggetti sono: i semafori, i mutex, gli eventi, i processi ed i thread, che sono tutti *oggetti del dispatcher*. I thread possono bloccarsi nel dispatcher del kernel in attesa che uno di questi oggetti sia segnalato. Il processo, il thread e le API della memoria virtuale usano gli handle del processo e del thread per identificare quello su cui operare. Altri esempi di oggetti includono i file, le sezioni, le porte e vari oggetti interni di I/O. Gli oggetti file sono usati per mantenere lo stato aperto dei file e dei dispositivi; le sezioni sono usate per mappare i file. I file aperti sono descritti in termini di oggetti file (file objects). I punti finali di comunicazione locale sono realizzati come oggetti porta.

Il gestore degli oggetti mantiene lo spazio dei nomi (namespace) interno di Windows XP. Contrariamente a UNIX, che radica il namespace del sistema nel file system, Windows XP usa un namespace astratto e collega i file system come dispositivi.

Il gestore dell'oggetto fornisce le interfacce per definire sia i tipi dell'oggetto che le istanze, la traduzione dei nomi in oggetti, mantenendo il namespace astratto, tramite direttori interni e collegamenti simbolici, la gestione e la cancellazione dell'oggetto. Gli oggetti sono tipicamente gestiti mediante conteggi di riferimento in modalità protetta e handle in modalità utente. Tuttavia, alcuni componenti in modalità kernel usano le stesse API del codice in modalità utente e di conseguenza utilizzano gli handle per manipolare gli oggetti. Se un handle deve vivere oltre il periodo di vita del processo corrente, esso viene marcato come handle del kernel e memorizzato nella tabella dell'oggetto relativa al processo di sistema. Il namespace astratto non sopravvive ad un riavvio, ma viene ricostruito mediante le informazioni di configurazione memorizzate nel registro di sistema, dalla scoperta di dispositivi plug-and-play e dalla creazione di oggetti tramite i componenti di sistema.

Il codice eseguibile di Windows XP permette che ad ogni oggetto sia assegnato un **nome**. Un processo può creare un oggetto con nome, mentre un secondo processo apre un handle per l'oggetto che condivide con il primo processo; i processi possono anche condividere gli oggetti duplicando gli handle fra processi e, in tal caso, gli oggetti non devono avere un nome.

Un nome può essere permanente o temporaneo; un nome permanente rappresenta un'entità, quale un disk drive, che permane anche se nessun processo vi accede, mentre un nome provvisorio esiste solo finché un processo mantiene un handle dell'oggetto.

I nomi dell'oggetto sono strutturati come nomi di percorso del file in MSDOS ed in UNIX. I direttori del namespace sono rappresentati da un **direttorio oggetto** (directory object) che contiene i nomi degli oggetti nel direttorio. Il namespace oggetto viene ampliato tramite l'aggiunta di oggetti del device che rappresentano i volumi che contengono il file system.

Gli oggetti sono manipolati da funzioni virtuali con implementazioni associate ad ogni tipo oggetto: `create`, `open`, `close`, `delete`, `query name`, `parse` e `security`. Gli ultimi tre oggetti necessitano di spiegazioni:

- `query name` viene chiamato quando un thread ha un riferimento ad un oggetto e vuole conoscerne il nome.
- `parse` è usato dal gestore durante la ricerca di un oggetto, una volta noto il nome.
- `security` viene chiamato per controlli di sicurezza su tutte le operazioni relative all'oggetto, quando un processo apre o chiude un oggetto, esegue cambiamenti al descrittore di sicurezza, o duplica un handle di un oggetto.

La procedura `parse` è usata per estendere il namespace astratto in modo da includere i file. La traduzione del percorso in un file oggetto inizia dalla radice del namespace astratto ed i componenti del percorso sono separati da caratteri `\` piuttosto che da `/` usati da UNIX. Ogni componente viene cercato nel direttorio corrente del namespace. I nodi interni nel namespace sono o direttori o collegamenti simbolici. Se viene trovato un oggetto foglia e non vi sono componenti del nome del percorso rimanenti, viene restituito l'oggetto foglia; in caso contrario la procedura dell'oggetto foglia è invocata con il nome del percorso restante.

Le procedure `parse` sono usate con un piccolo numero di oggetti che appartengono alla GUI di Windows, al gestore di configurazione (registro), e, più importante, agli oggetti del dispositivo che rappresentano i file system.

La procedura `parse` per il tipo di oggetto del dispositivo alloca un file oggetto ed inizia operazioni di I/O, `open` o `create` sul file system, e se hanno successo, i campi del file oggetto sono riempiti per descrivere il file.

In conclusione, il pathname di un file viene usato per attraversare il namespace del gestore dell'oggetto, traducendo il pathname assoluto originale nella coppia (oggetto del dispositivo, pathname relativo). Questa coppia viene poi passata al file system tramite il gestore di I/O, che riempie il file oggetto; il file oggetto stesso non ha nome, ma ci si riferisce ad esso tramite un handle.

I file system di UNIX hanno **collegamenti simbolici** (symbolic links) che permettono soprannomi multipli o pseudonimi per lo stesso file. L'**oggetto di collegamento simbolico** (symbolic-link object), implementato dal gestore di oggetto di Windows XP, è usato all'interno del namespace astratto, ma non per assegnare uno pseudonimo ai file in un file system; anche in questo modo, i collegamenti simbolici sono ancora molto utili e sono usati per organizzare il namespace, con un'organizzazione di direttorio dei / dispositivi come in UNIX. Inoltre sono usati per mappare lettere di drive standard MSDOS in nomi di drive. Le lettere di drive sono collegamenti simbolici che possono essere rimappate in base alla convenienza dell'utente o dell'amministratore.

Le lettere di drive sono un posto in cui il namespace astratto in Windows XP non è globale. Ciascuno utente connesso ha un proprio insieme di lettere di drive in modo da evitare interferenze reciproche. D'altra parte, le sessioni del terminal server condividono tutti i processi all'interno di una sessione. `BaseNamedObjects` contiene gli oggetti con nome generati dalla maggior parte delle applicazioni.

Sebbene il namespace non sia direttamente visibile in rete, il metodo del gestore dell'oggetto `parse` viene usato per aiutare ad accedere ad un oggetto con nome in un altro sistema. Quando un

processo tenta di aprire un oggetto che risiede in un computer remoto, il gestore dell'oggetto chiama il metodo `parse` per l'oggetto del dispositivo che corrisponde ad un reindirizzamento della rete avendo come risultato un'operazione di I/O che accede ai file in rete.

Gli oggetti sono istanze di un **tipo oggetto** (object type) che specificano come debbano essere allocate le istanze, le definizioni dei campi dati e la realizzazione dell'insieme standard delle funzioni virtuali usate per tutti gli oggetti. Queste funzioni implementano operazioni quali la mappatura dei nomi in oggetti, la chiusura e la cancellazione e applicazioni di sicurezza.

Il gestore dell'oggetto tiene traccia di due conteggi per ogni oggetto; il puntatore conteggio è il numero di riferimenti ad un oggetto. Il codice della modalità di protezione che si riferisce agli oggetti deve mantenere un riferimento all'oggetto per assicurarsi che l'oggetto non venga cancellato mentre è in uso. Il conteggio degli handle è il numero di elementi nella tabella degli handle che fanno riferimento ad un oggetto e ogni handle è riportato nel conteggio di riferimento.

Quando un handle di un oggetto viene chiuso, si chiama la procedura di chiusura dell'oggetto. Nel caso di oggetti file, ciò provoca che il gestore di I/O esegua un'operazione di pulizia completa alla chiusura dell'ultimo handle. L'operazione di pulizia comunica al file system che al file non si può più accedere in modalità utente in modo da rimuovere la condivisione di restrizioni, i blocchi, ed altri stati specifici della procedura `open` corrispondente.

La chiusura di ogni handle rimuove un riferimento dal conteggio del puntatore, ma le componenti interne del sistema possono mantenere riferimenti addizionali; quando si rimuove il riferimento finale, si invoca la procedura di cancellazione dell'oggetto. Inoltre, usando come esempio gli oggetti file, la procedura di cancellazione provoca che il gestore di I/O richieda al file system un'operazione `close` sul file oggetto che provoca che il file system deallochi tutte le strutture dati interne e allocate in precedenza per il file oggetto.

Dopo il completamento della procedura di cancellazione per un oggetto temporaneo, esso viene cancellato dalla memoria. Gli oggetti possono essere resi permanenti (almeno riguardo all'avvio corrente del sistema) chiedendo al gestore dell'oggetto di prendere un riferimento supplementare per l'oggetto; quindi gli oggetti permanenti non sono cancellati anche quando viene rimosso l'ultimo riferimento esterno al gestore dell'oggetto. Quando un oggetto permanente è reso di nuovo temporaneo, il gestore dell'oggetto rimuove il riferimento supplementare e se questo era l'ultimo riferimento, l'oggetto viene cancellato. Gli oggetti permanenti sono rari, e sono usati principalmente per i dispositivi, la mappatura delle lettere dei drive e dei direttori e gli oggetti di collegamento simbolici.

Il compito del **gestore degli oggetti** (object manager) è di sorvegliare l'uso di tutti gli oggetti gestiti. Quando un thread vuole usare un oggetto, chiama il metodo `open` del gestore dell'oggetto per ottenerne un riferimento. Se l'oggetto viene aperto da una API in modalità utente, il riferimento viene inserito nella tabella degli oggetti del processo e viene restituito un handle.

Un processo ottiene un handle creando un oggetto, aprendo un oggetto esistente, ricevendo un handle duplicato da un altro processo, o ereditando un handle da un **processo genitore** simile a come UNIX ottiene un descrittore di file. Questi handle sono tutti memorizzati nella **tabella degli oggetti** (object table) come elementi nella tabella dell'oggetto che contiene i diritti di accesso e le dichiarazioni se l'handle può essere ereditato dal **processo figlio**. Al termine del processo, Windows XP chiude automaticamente gli handle aperti dei processi.

Gli **handle** sono interfacce standardizzate per tutti i generi di oggetti. Come un descrittore di file in UNIX, un handle dell'oggetto è un identificatore unico per un processo che conferisce la capacità di accedere e manipolare risorse di sistema. Gli handle possono essere duplicati entro un processo, o fra processi; si usa questo ultimo caso quando si creano processi figlio o si implementano contesti di esecuzione di fuori dal processo.

Poiché il gestore dell'oggetto è l'unica entità che genera handle dell'oggetto, è il posto naturale per controllare la sicurezza; il gestore dell'oggetto controlla se un processo ha i diritti di accedere ad un oggetto quando il processo cerca di aprirlo. Inoltre, il gestore fa rispettare le quote, come la quantità massima di memoria che un processo può usare, caricando un processo per la memoria occupata da tutti gli oggetti referenziati e rifiutando di allocare più memoria quando i carichi accumulati superano la quota del processo.

Quando il processo di login autentica un utente, viene associato un token al processo utente. Il token di accesso contiene informazioni quali l'identificatore di sicurezza, l'identificatore di gruppo, i privilegi, il gruppo primario e la lista di default di controllo dell'accesso. I servizi e gli oggetti, a cui un utente può accedere, sono determinati da questi attributi.

Il token che controlla l'accesso è associato con il thread che esegue l'accesso. Normalmente il token del thread è mancante e commuta sul token di processo, ma i servizi devono spesso eseguire codice a nome del loro client. Windows XP permette ai thread di usare una personalità temporanea mediante il token del client e, di conseguenza, il token del thread non è necessariamente lo stesso del processo.

In Windows XP, ogni oggetto è protetto da una lista di controllo degli accessi contenente gli identificatori di sicurezza ed i diritti di accesso rilasciati. Quando un thread tenta di accedere ad un oggetto, il sistema paragona l'identificatore con il token di accesso del thread per stabilire se è consentito l'accesso; il controllo avviene solo quando si apre un oggetto, in modo da non sia possibile negare l'accesso dopo l'apertura. I componenti del sistema operativo che operano in modalità kernel evitano il controllo di accesso, poiché il codice in modalità kernel si presume che sia affidabile e, di conseguenza, deve evitare le vulnerabilità di sicurezza, come il lasciare i controlli disabilitati mentre si crea un handle accessibile in modalità utente in un processo non affidabile.

Generalmente, il creatore dell'oggetto stabilisce la lista di controllo di accesso per l'oggetto e se non ne viene data una esplicitamente, si può porre al valore di default mediante la procedura `open` del tipo oggetto, oppure si può ottenere una lista di default dall'oggetto dal token di accesso dell'utente.

Il token di accesso ha un campo che controlla la verifica degli accessi all'oggetto; le operazioni verificate sono annotate nel log di sicurezza del sistema con un identificativo dell'utente. Un amministratore controlla il log per scoprire i tentativi di violazione del sistema o di accesso ad oggetti protetti.

### **3.3.2 Gestore di memoria virtuale**

Il componente di codice eseguibile che controlla lo spazio d'indirizzo virtuale, l'allocazione della memoria fisica e la paginazione è **il gestore della memoria virtuale** (virtual-memory manager: VM). Il progetto di gestore di VM presuppone che l'hardware sottostante supporti la mappatura da memoria virtuale a quella fisica, un meccanismo di paginazione e la coerenza trasparente della cache nei sistemi multiprocessore e che permetta elementi multipli nella tabella di pagina da mappare nello stesso frame di pagina fisica. Il gestore di VM in Windows XP usa uno schema di gestione basato sulla pagina di dimensione di 4 Kb nei processori IA32 compatibili, e di 8 Kb in quelli IA64. Le pagine dei dati allocate ad un processo che non sono in memoria fisica sono memorizzate o nei **file di paginazione** (paging files) su disco, o mappate direttamente in un file normale nel file system locale o remoto. Le pagine possono anche essere marcate come zero-on-demand (zero a richiesta).

Nei processori IA32, ogni processo ha uno spazio di indirizzamento virtuale di 4 GB, in cui i 2 GB superiori sono quasi identici in tutti i processi e sono usati da Windows XP in modalità kernel per accedere al codice e alle strutture dati del sistema operativo. Le zone chiave della regione in modalità kernel, che non sono identiche in ogni processo, sono **l'automappatura della tabella di pagina** (page-table self-map), **l'iperspazio** (hyperspace) e lo **spazio di sessione** (session space). L'hardware fa riferimento alle tabelle di pagina di un processo usando la numerazione di pagina del frame fisico. Il gestore di VM mappa le tabelle di pagina in una singola regione di 4 MB nello spazio di indirizzamento del processo in modo da potervi accedere tramite indirizzi virtuali. L'iperspazio mappa le informazioni del working-set del processo corrente nello spazio di indirizzamento in modalità kernel.

Lo spazio di sessione è usato per condividere Win32, ed altri driver specifici della sessione, tra tutti i processi nella stessa sessione del server di terminali (terminal server), invece che con tutti i processi del sistema. I 2 GB inferiori sono specifici di ogni processo, e sono accessibili sia dai thread utente sia da quelli in modalità kernel. Certe configurazioni di Windows XP riservano solo 1 GB ad uso del sistema operativo, permettendo ad un processo di usare 3 GB di spazio di indirizzamento; l'uso di 3GB riduce drasticamente la quantità di cache dei dati del kernel, tuttavia, per grandi applicazioni che gestiscono un proprio I/O quali i database SQL, il vantaggio di un maggior spazio di indirizzamento in modalità utente può valere la perdita della cache.

Il gestore di VM di Windows XP usa un processo a due passi per allocare la memoria utente: nel primo passo, *riserva* una porzione dello spazio di indirizzamento virtuale del processo; nel secondo passo, *coinvolge* l'allocazione assegnando lo spazio di memoria virtuale (memoria fisica o spazio nei file di paginazione). Windows XP limita la quantità di spazio di memoria virtuale che un processo utilizza facendo rispettare una quota di memoria coinvolta: un processo rilascia la memoria, non più in uso, liberando completamente la memoria virtuale per altri processi. Le API usate per riservare indirizzi virtuali e per coinvolgere la memoria virtuale, prendono, come parametro, un handle di un processo oggetto; ciò consente ad un processo di controllare la memoria virtuale di un altro processo: questa è la modalità mediante la quale i sottosistemi di ambiente gestiscono la memoria dei loro processi client.

Per favorire le prestazioni, il gestore di VM permette ad un processo privilegiato di bloccare le pagine selezionate in memoria fisica, assicurando al file paginante, in tal modo, che le pagine non vengano paginate fuori. I processi allocano pure memoria fisica grezza e poi mappano regioni nel proprio spazio di indirizzamento virtuale. I processori IA32, dotati della modalità Estensione di Indirizzo Fisico (Physical Address Extension: PAE) possono avere, in un sistema, fino a 64 GB di memoria fisica e tale memoria non può essere tutta mappata immediatamente nello spazio di indirizzamento del processo, ma Windows XP rende disponibile la memoria usando le API: Estensione dell'Indirizzo di Windows (address windowing extension: AWE) che allocano memoria fisica e poi mappano regioni di indirizzi virtuali nello spazio di indirizzamento del processo come parte della memoria fisica. La funzione AWE è, in primo luogo, usata da applicazioni molto grandi quali i database SQL.

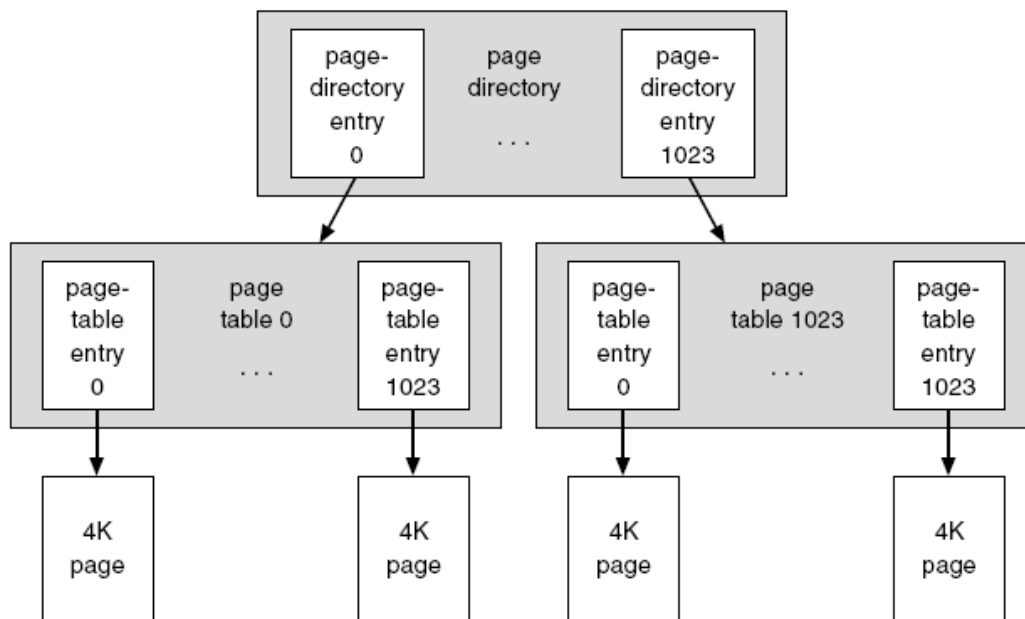
Windows XP implementa la memoria condivisa, definendo una **sezione oggetto** (section object); dopo avere ottenuto un handle per una sezione oggetto, un processo mappa la porzione di memoria necessaria nel proprio spazio di indirizzamento; questa porzione è chiamata **vista** (view). Un processo ridefinisce la propria vista di un oggetto per avere accesso all'intero oggetto: una regione alla volta.

Un processo può controllare, in parecchi modi, l'uso della sezione della memoria condivisa, limitando la dimensione massima di una sezione che può essere spalleggiata dallo spazio su disco sia in un file del sistema di paginazione sia da un normale file: un **file a memoria mappata**

(memory-mapped file). Una sezione può essere *basata*, intendendo con ciò che la sezione compare allo stesso indirizzo virtuale per tutti i processi che tentano di accedervi. Infine, la protezione di memoria delle pagine nella sezione, può essere posta a sola lettura, a lettura e scrittura, a lettura-scrittura–esecuzione, a sola esecuzione, ad accesso vietato, o copy-on-write. Gli ultimi due tipi di protezione hanno bisogno di una qualche spiegazione:

- Una pagina in cui sia vietato l'accesso, segnala un'eccezione se si tenta di accedervi; tale eccezione è usata, per esempio, per controllare se un programma difettoso itera oltre la fine di un array. Sia l'allocatore della memoria in modalità utente, che quello speciale del kernel, usato dal dispositivo di verifica, possono essere configurati per mappare ogni allocazione alla fine di una pagina seguita da una pagina ad accesso vietato per rilevare sovraccarichi nel buffer.
- Il meccanismo di copy-on-write incrementa l'uso efficiente di memoria fisica da parte del gestore di VM: quando due processi desiderano copie indipendenti di un oggetto, il gestore di VM mette una singola copia condivisa in memoria virtuale ed attiva la proprietà di copy-on-write per quella regione di memoria. Se uno dei processi cerca di modificare i dati in una pagina copy-on-write, il gestore della VM esegue una copia privata della pagina per il processo.

In Windows XP, la traduzione dell'indirizzo virtuale usa una tabella di pagine a più livelli; nei processori IA32, senza PAE abilitato, ogni processo ha un **direttorio di pagine** (page directory) che contiene 1024 elementi nel **direttorio delle pagine** (page-directory entries: PDE) da 4 byte. Ogni PDE punta ad **una tabella di pagine** che contiene 1024 **elementi della tabella delle pagine** (page-table entries: PTE) di dimensione di 4 byte, e ogni PTE punta ad una **struttura di pagina** (page frame) nella memoria fisica da 4 Kb. In un processo, la dimensione totale di tutte le tabelle di pagina è di 4 MB, pertanto il gestore di VM pagina esternamente, quando è necessario, le tabelle su disco. Consultare la Figura 3 per vedere uno schema di questa struttura.



**Figura 3.** La tabella di pagina.

Page directory = direttorio di pagina

Page-directory entry 0 = ingresso 0 al direttorio di pagina

Page-directory entry 1023 = ingresso 1023 al direttorio di pagina

page table 0 = tabella di pagina 0

page table 1023 = tabella di pagina 1023

Page-table entry 0 = ingresso 0 alla tabella di pagina

Page-table entry 1023 = ingresso 1023 alla tabella di pagina

4K page = pagina di 4K

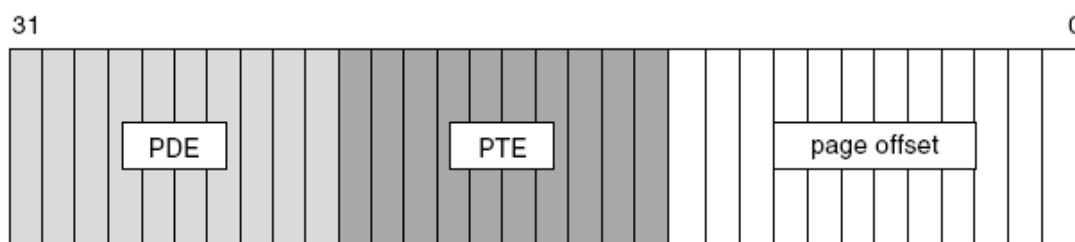
Il direttorio di pagine e le tabelle delle pagine, sono referenziati dall'hardware tramite gli indirizzi fisici. Per migliorare le prestazioni, il gestore di VM automappa il direttorio delle pagine e le tabelle delle pagine in una regione di indirizzi virtuali da 4 MB; l'automappatura permette al gestore di VM di tradurre un indirizzo virtuale nel corrispondente PDE o PTE, senza ulteriori accessi alla memoria. Quando il contesto di un processo cambia, deve essere cambiato un singolo elemento nel direttorio delle pagine al fine di mappare le nuove tabelle di pagina del processo. Per varie ragioni, l'hardware richiede che il direttorio di ogni pagina o la tabella di pagina occupino una singola pagina, e pertanto il numero di PDE o PTE che stanno in una pagina determina come sono tradotti gli indirizzi virtuali.

Nel seguito, descriveremo come gli indirizzi virtuali sono tradotti in indirizzi fisici nei processori IA32 compatibili (senza PAE abilitato). Un valore di 10-bit può rappresentare qualsiasi valore tra

0 e 1023, pertanto tale valore può selezionare un qualsiasi elemento o nel direttorio di pagina o nella tabella della pagina; questa proprietà viene usata quando un puntatore all'indirizzo virtuale è tradotto in un indirizzo in byte nella memoria fisica. Un indirizzo di memoria virtuale a 32-bit è suddiviso in tre valori, come mostra la Figura 4. I primi 10 bit dell'indirizzo virtuale sono usati come indice nel direttorio delle pagine e questo indirizzo seleziona un elemento nel direttorio di pagina (PDE) che contiene l'effettiva struttura della pagina per una tabella di pagina. L'unità di gestione della memoria (MMU) usa i 10 bit successivi dell'indirizzo virtuale per selezionare un PTE dalla tabella di pagina che specifica una struttura di pagina nella memoria fisica; i restanti 12 bit dell'indirizzo virtuale rappresentano lo spiazzamento di uno specifico byte nella struttura di pagina. MMU crea un puntatore al byte specifico in memoria fisica concatenando i 20 bit del PTE con i 12 bit inferiori dell'indirizzo virtuale. Pertanto, il PTE a 32-bit ha 12 bit per descrivere lo stato della pagina fisica. L'hardware IA32 riserva 3 bit ad uso del sistema operativo. I rimanenti bit specificano se si è avuto accesso o si è scritto nella pagina, gli attributi nascosti, la modalità di accesso: se la pagina è globale, e se PTE è valido.

I processori IA32, dotati di PAE, usano PDE e PTE a 64 bit per poter rappresentare il valore più grande a 24-bit del campo del frame di pagina. Pertanto i direttori di pagina di secondo livello e le tabelle di pagina contengono, rispettivamente, solo 512 PDE e PTE. Per fornire 4 GB di spazio di indirizzamento virtuale si richiede un livello supplementare del direttorio di pagina con quattro PDE. La traduzione di un indirizzo virtuale a 32 bit usa 2 bit per l'indice del direttorio ad alto livello e 9 bit per ognuno dei direttori di pagina di secondo livello e per le tabelle di pagina.

Per evitare un overhead nella traduzione di ogni indirizzo virtuale mediante l'esame di PDE e PTE, i processori usano **una memoria associativa** (translation-lookaside buffer: TLB) che contiene una cache di memoria associativa per mappare le pagine virtuali in PTE. A differenza dell'architettura IA32, dove la TLB è mantenuta dalla MMU hardware, IA64 utilizza una procedura trap software per fornire le traduzioni mancanti alla TLB e ciò fornisce al gestore di VM flessibilità nello scegliere le strutture dati da usare. In Windows XP si è scelta, per i processori IA64, una struttura ad albero a tre livelli per mappare gli indirizzi virtuali in modalità utente.



**Figura 4.** Traduzione di un indirizzo da virtuale a fisico in un processore IA32.

page offset= spiazzamento della pagina

Nei processori IA64, il formato di pagina è di 8 Kb, ma i PTE impiegano 64 bit, in tal modo una pagina contiene ancora solo 1024 (il valore di 10 bit) PDE o PTE; di conseguenza, con un valore di 10 bit a livello superiore di PDE, di 10 bit al secondo livello, di 10 bit della tabella di pagina e 13 bit di spiazamento della pagina, la porzione utente dello spazio di indirizzamento virtuale del processo in Windows XP, per un processore IA64, è 8 di TB (corrispondente a 43 bit). La limitazione a 8 TB della versione corrente di Windows XP, è inferiore alle capacità del processore IA64, ma rappresenta un'alternanza fra il numero di riferimenti di memoria richiesti per gestire le perdite di TLB e la dimensione dello spazio di indirizzamento supportato in modalità utente.

Una pagina fisica può essere in una delle sei situazioni: valida, libera, azzerata, standby, modificata, difettosa, o in transizione. Una pagina *valida* è in uso in un processo attivo. Una pagina *libera* è quella che non è referenziata in un PTE. Una pagina *azzerata* è una pagina libera che è stata azzerata ed è pronta per un uso immediato per soddisfare i fault di zero-on-demand. Una pagina *modificata* è una che è stata scritta da un processo e deve essere inviata al disco prima di venire allocata ad un altro processo. Le pagine in *standby* sono copie di informazioni già memorizzate su disco e possono essere pagine che non sono state modificate, pagine che sono state modificate, pagine modificate che sono già state scritte su disco, o pagine in pre-fetch (pre-caricamento) per sfruttare la località. Una pagina *difettosa* è inutilizzabile a causa del rilevamento di un errore hardware; infine, una pagina di *transizione* è quella che è in trasferimento da disco ad una struttura di pagina allocata nella memoria fisica.

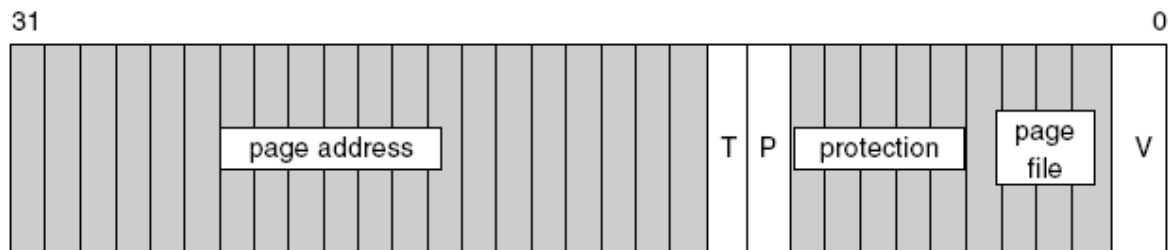
Quando in PTE il bit di validità è zero, il gestore di VM definisce il formato degli altri bit. Pagine non valide possono avere un numero di stati rappresentato dai bit in PTE. Le pagine del file di pagina che non hanno mai avuto fault sono contrassegnate come zero-on-demand. I file mappati, tramite gli oggetti della sezione, codificano un puntatore a quell'oggetto della sezione. Le pagine che sono state scritte nel file di pagina contengono abbastanza informazioni per trovare la pagina su disco.

La struttura attuale del file di pagina PTE è riportata in Figura 5. PTE contiene 5 bit di protezione della pagina, 20 bit di spiazamento del file di pagina, 4 bit per selezionare il file di paginazione e 3 bit che descrivono lo stato della pagina. Un file di pagina PTE è contrassegnato per essere un indirizzo virtuale non valido per la MMU, dal momento che il codice eseguibile e i file a memoria mappata già ne posseggono una copia su disco e non hanno bisogno di spazio in un file paginante. Se una di queste pagine non è in memoria fisica, la struttura di PTE è la seguente: si usa il bit più significativo per specificare la protezione di pagina, i successivi 28 bit si usano per indicizzare nella struttura del sistema dati quella che individua un file ed uno spiazamento entro il file relativo di pagina; i 3 bit inferiori specificano lo stato della pagina.

Indirizzi virtuali non validi possono anche trovarsi in un certo numero di stati temporanei e sono parte degli algoritmi di paginazione. Quando una pagina viene rimossa da un processo, il working set è spostato o nella lista modificata (da scrivere su disco) o direttamente nella lista standby. Se viene scritto nella lista standby, la pagina viene reclamata, senza leggerla da disco, se è ancora necessaria, prima di spostarla nella lista libera. Se possibile, il gestore di VM usa cicli di attesa della CPU nelle pagine zero nella lista libera e le sposta nella lista azzerata. Le pagine di transizione sono state allocate ad una pagina fisica, e attendono il completamento della paginazione I/O prima che il PTE sia marcato come valido.

Windows XP usa gli oggetti della sezione per descrivere le pagine che sono condivisibili fra i processi. Ogni processo ha un proprio insieme di tabelle di pagina virtuale, ma l'oggetto della sezione contiene pure un insieme di tabelle di pagina in cui si trova il master (o prototipo) PTE. Quando un PTE nella tabella di pagina di un processo è marcato come valido, esso punta alla struttura di pagina fisica che contiene la pagina, come deve avvenire nei processori IA32 in cui la MMU hardware legge le tabelle di pagina direttamente dalla memoria. Quando una pagina comune

è resa non valida, il PTE viene modificato per puntare al prototipo PTE associato con l'oggetto della sezione.



**Figura 5.** Elemento del file di pagina della tabella di pagina. Il bit di validità è zero.

page address = indirizzo di pagina  
 protection = protezione  
 page file = file di pagina

Le tabelle di pagina associate con un oggetto della sezione sono virtuali in quanto sono create e modellate in base alle necessità. L'unico prototipo PTE necessario è quello che descrive le pagine per cui vi è una visione correntemente mappata; ciò migliora notevolmente le prestazioni e permette un uso più efficiente degli indirizzi del kernel virtuale.

Il prototipo PTE contiene l'indirizzo della struttura di pagina e i bit di stato relativi alla protezione; pertanto il primo accesso, tramite un processo, ad una pagina condivisa genera un fault di pagina e, dopo il primo accesso, ulteriori accessi sono eseguiti in modo normale. Se un processo scrive in una pagina copy-on-write, marcata a sola lettura nel PTE, il gestore di VM esegue una copia della pagina e segna il PTE come scrivibile; in tal caso le pagine non hanno più una pagina condivisibile e le pagine condivise non appaiono più nel file di pagina, ma invece si trovano nel file-system.

Il gestore di VM tiene traccia di tutte le pagine della memoria fisica in un **database del pageframe** (page-frame database) in cui c'è un elemento per ogni pagina della memoria fisica nel sistema, così che possa mantenere lo stato delle pagine. Le strutture di pagina non referenziate da un PTE valido sono collegate a liste in base al tipo di pagina come azzerato, modificato, libero, ecc.

Se una pagina fisica condivisa è marcata come valida per qualsiasi processo, la pagina non può essere rimossa dalla memoria e il gestore di VM mantiene un conteggio di PTE valido per ogni pagina che si trova nel database della struttura di pagina. Quando il conteggio si azzerava, la pagina fisica può essere riutilizzata, una volta che il suo contenuto sia stato scritto su disco (era marcato come sporco).

Quando accade un fault di pagina, il gestore di VM va alla ricerca di una pagina fisica in cui tenere i dati. Per le pagine zero-on-demand, la prima scelta è di cercare una pagina che già è stata azzerata, se nessuna è disponibile, si sceglie una pagina dalla lista di quelle libere o in standby e la pagina viene azzerata prima di procedere. Se la pagina in fault è stata contrassegnata come in

transizione, o se è già stata letta dal disco, o è stata demappata o modellata, è ancora disponibile nella lista delle pagine in standby o modificate. Il thread attende che si completi l'operazione di I/O o, nel caso peggiore, reclama la pagina dalla lista appropriata.

In caso contrario, si deve eseguire un'operazione di I/O per leggere la pagina dal file paginante o dal file system. Il gestore di VM cerca di allocare una pagina disponibile o dalla lista di quelle libere o dalla lista di standby. Le pagine nella lista modificata non possono essere usate finché non vengono riscritte su disco e trasferite nella lista di standby; se non ci sono pagine disponibili, il thread si blocca finché il gestore del working set non modella le pagine in memoria, a meno che una pagina in memoria fisica non venga demappata da un processo.

Windows XP usa una politica di rimpiazzo relativa ad un processo del tipo first-in, first-out (FIFO) per ottenere pagine dai processi che usano più della dimensione minima del working set. Windows XP controlla le pagine in fault di ogni processo che si trovano alla dimensione minima del working set e aggiusta di conseguenza la dimensione di questo ultimo. Quando un processo è iniziato, gli viene assegnato di default un working set minimo di 50 pagine e il gestore di VM rimpiazza e modella le pagine in base alla loro età che viene determinata contando quanti cicli di modellazione sono avvenuti senza il PTE. Le pagine modellate vengono mosse nella lista di standby o in quella modificata, a seconda che il bit sia posto nel PTE della pagina.

Il gestore di VM non contrassegna in fault solo la pagina necessaria immediatamente. Ricerche mostrano che il riferimento di memoria di un thread tende ad avere una proprietà di **località** (locality); quando si usa una pagina; è probabile che le pagine adiacenti vengano referenziate in un prossimo futuro. (Si pensi all'iterazione di un array, o a prendere istruzioni sequenziali che formano il codice eseguibile di un thread). A causa della località, quando il gestore di VM segna in fault una pagina, segna anche alcune pagine adiacenti. Questo tipo di pre-fetch tende a ridurre il numero totale di fault di pagina; le scritture sono raggruppate per ridurre il numero di operazioni indipendenti di I/O.

Oltre alla gestione della memoria coinvolta, il gestore di VM controlla la memoria riservata di ogni processo, o lo spazio di indirizzamento virtuale; ad ogni processo è associato un albero largo e piatto che descrive i valori degli indirizzi virtuali in uso ed il relativo uso. Ciò permette al gestore di VM di segnare, in base alle necessità, in fault tabelle di pagina. Se il PTE per un indirizzo fault non esiste, il gestore di VM cerca l'indirizzo nell'albero del processo dei **descrittori virtuali di indirizzo** (virtual address descriptors:VAD) ed usa queste informazioni per riempire il PTE mancante e ritrovare la pagina. In alcuni casi una pagina della tabella di pagina può non esistere e deve essere allocata in modo trasparente ed inizializzata dal gestore di VM.

### **3.3.3 Gestore di processo**

Il gestore di processo di Windows XP fornisce servizi per la creazione, la cancellazione e l'uso di processi, thread e job, ma non conosce le relazioni padre-figlio e le gerarchie dei processi; tali raffinatezze sono lasciate al particolare sottosistema ambientale proprietario del processo. Il gestore del processo non è neppure coinvolto nella schedulazione dei processi, ad eccezione dell'individuazione delle priorità e delle affinità dei processi e dei thread, al momento della loro creazione; la schedulazione dei thread avviene nel dispatcher del kernel.

Uno o più thread sono contenuti nei processi e questi ultimi possono venire collegati insieme in grandi unità, chiamati **oggetti job** (job objects) che pongono dei limiti all'uso della CPU, alla

dimensione del working set, alle affinità del processore che controlla contemporaneamente processi multipli. Gli oggetti job sono usati per gestire grandi macchine nei centri dati.

Un esempio di creazione di un processo in Win32 è il seguente. Quando un'applicazione di Win32 chiama `CreateProcess`, viene inviato un messaggio al sottosistema Win32 per notificare la creazione del processo. `CreateProcess`, nel processo originale, chiama poi una API nel gestore del processo del codice eseguibile di NT proprio per creare il processo. Il gestore di processo chiama il gestore dell'oggetto per creare un processo oggetto, e restituisce l'handle dell'oggetto a Win32 che chiama di nuovo il gestore del processo al fine di creare un thread per il processo e restituisce gli handle al nuovo processo e al thread.

Le API di Windows XP che manipolano la memoria virtuale, i thread e duplicano gli handle, prendono un handle del processo in modo che i sottosistemi possano eseguire operazioni sulla base di un nuovo processo senza dover eseguire direttamente il contesto del nuovo processo. Una volta che si è creato un nuovo processo, si genera il thread iniziale e una APC è inviata al thread per iniziare l'esecuzione a livello del loader dell'immagine in modalità utente. Il loader è una `ntdll.dll`, che è una libreria di collegamento mappata automaticamente in ogni processo recentemente creato. Windows XP supporta inoltre una `fork()` nello stile UNIX di creazione del processo per supportare il sottosistema ambientale di POSIX. Sebbene l'ambiente Win32 chiami il gestore del processo dal processo del cliente, POSIX usa la natura di processo incrociato delle API di Windows XP per creare il nuovo processo entro il processo del sottosistema.

Il gestore del processo implementa pure l'accodamento e la consegna delle chiamate di procedura asincrona (APC) ai thread. Le APC sono usate dal sistema per iniziare l'esecuzione del thread, per il completamento degli I/O, per terminare i thread e collegarsi ai debugger. Il codice in modalità utente può anche accodare una APC ad un thread per consegnare notifiche mediante segnali. Per supportare POSIX, il gestore di processo fornisce le API che inviano allarmi ai thread per sbloccarli dalle chiamate di sistema.

Il supporto del debugger nel gestore del processo include la possibilità di sospendere e riprendere i thread, e di creare thread che iniziano in modalità sospesa. Ci sono anche API del gestore del processo che ottengono e pongono un contesto del registro del thread e accedono alla memoria virtuale di un altro processo.

I thread possono essere creati nel processo corrente; possono anche essere iniettati in un altro processo. Nell'ambito del codice eseguibile, i thread esistenti possono temporaneamente attaccarsi ad un altro processo e questo metodo è usato dai thread lavoratori che devono operare nel contesto del processo che produce una richiesta di lavoro.

Il gestore supporta inoltre la personificazione. Un thread in esecuzione in un processo, con un token di sicurezza appartenente ad un utente, può mettere un token specifico del thread che appartiene ad un altro utente. Questa possibilità è fondamentale nel modello computazionale client-server, dove i servizi devono agire in base ad una varietà di client con differenti identificatori di sicurezza.

### **3.3.4 La chiamata di procedura locale**

Windows XP usa un modello client-server; i sottosistemi di ambiente sono server che assumono personalità particolari del sistema operativo, e il modello client-server è usato per implementare un certo numero di sistemi operativi oltre ai sottosistemi di ambiente. Con questo modello si implementano altre caratteristiche, quali la gestione della sicurezza, lo spooler di stampa, i servizi

web, i file system di rete, l'autoconfigurazione (plug-and-play) e molte altre. Per ridurre l'impronta della memoria, servizi multipli sono spesso raccolti insieme in alcuni processi, che possono poi fare affidamento sulle possibilità del gruppo di thread in modalità utente per condividere i thread e aspettare i messaggi (vedere Paragrafo 3.3.3 di questo capitolo).

Il sistema operativo usa la chiamata di procedura locale (LPC) per passare le richieste ed i risultati fra i processi client e server all'interno di una singola macchina. In particolare, LPC è usato per richiedere servizi dai vari sottosistemi di Windows XP. LPC è simile, sotto molti aspetti, ai meccanismi RPC, usati da molti sistemi operativi per i processi distribuiti nelle reti, ma LPC è ottimizzato per l'uso all'interno di un singolo sistema.

L'implementazione in Windows XP di Open Software Foundation (OSF) RPC usa spesso LPC come un mezzo di trasporto sulla macchina locale.

LPC è un meccanismo di passaggio dei messaggi. Il processo del server pubblica un oggetto globalmente visibile dell'oggetto "porta di collegamento" e quando un cliente desidera i servizi da un sottosistema, apre un handle per l'oggetto "porta di collegamento" del sottosistema e invia una richiesta alla porta. Il server crea un canale e restituisce un handle al client. Il canale consiste in una coppia di porte di comunicazione private: uno per i messaggi client-server e l'altra per i messaggi server-client. I canali di comunicazione supportano un meccanismo di callback, in modo che il client ed il server possano accettare richieste quando normalmente si aspettavano una risposta.

Quando viene creato il canale LPC, si deve specificare una delle tre tecniche di passaggio del messaggio.

1. La prima tecnica è adatta a piccoli messaggi (fino a duecento byte) e, in questo caso, la coda del messaggio della porta viene usata per la memorizzazione intermedia ed i messaggi sono copiati da un processo all'altro.
2. La seconda tecnica è per messaggi più grandi. In questo caso, un oggetto sezione della memoria condivisa viene creato per il canale. I messaggi inviati tramite la coda di messaggi della porta contengono un puntatore e graduano le informazioni riferendosi all'oggetto sezione. Ciò evita la necessità di copiare grandi messaggi; il mittente mette i dati nella sezione condivisa, e il ricevente li vede direttamente.
3. La terza tecnica del passaggio di messaggi LPC usa le API che leggono/scrivono direttamente nello spazio di indirizzo del processo. LPC fornisce funzioni e sincronizzazione in modo che un server possa accedere ai dati nel client.

Il gestore della finestra di Win32 usa una propria forma di passaggio del messaggio, indipendente dalle capacità del codice eseguibile LPC. Quando un client chiede una connessione che usa la messaggeria del gestore della finestra, il server installa tre oggetti: un thread del server dedicato per gestire le richieste, un oggetto sezione di 64 Kb e un oggetto coppia-evento. *Un oggetto coppia-evento* è un oggetto di sincronizzazione che è usato dal sottosistema di Win32 per notificare quando il thread del client ha copiato un messaggio nel server di Win32, o viceversa. L'oggetto sezione passa i messaggi e l'oggetto coppia-evento esegue la sincronizzazione. La messaggeria del gestore di Window presenta parecchi vantaggi: l'oggetto sezione elimina la copiatura dei messaggi, poiché rappresenta una regione della memoria condivisa; l'oggetto coppia-evento elimina l'overhead di usare l'oggetto porta per passare i messaggi che contengono puntatori e lunghezze. Il thread del server dedicato elimina l'overhead nel determinare quale thread client chiama il server, poiché c'è un thread del server per ogni thread del client. In conclusione, il kernel dà la preferenza di schedulazione a questi thread del server dedicati per migliorare le prestazioni.

### 3.3.5 Gestore di I/O

Il **gestore di I/O** (I/O manager) è responsabile del file system, dei driver del dispositivo e dei driver della rete. Tiene traccia di quali driver del dispositivo, driver del filtro e file system sono caricati ed inoltre controlla i buffer per le richieste di I/O. Funziona con il gestore di VM per fornire file a memoria mappata I/O e controlla il gestore della cache di Windows XP, che gestisce la cache dell'intero sistema di I/O. Il gestore di I/O è fondamentalmente asincrono; mentre l'I/O sincrono è fornito esplicitamente in attesa di un completamento di un'operazione di I/O. Il gestore di I/O fornisce parecchi modelli di completamento asincrono di I/O compresa la regolazione di eventi, l'invio degli APC al thread di inizio, e le porte di completamento di I/O che permettono ad un singolo thread di processare i completamenti di I/O da parte di molti altri thread.

I driver del dispositivo sono organizzati come una lista per ogni dispositivo (chiamato driver o stack I/O in conseguenza di come vengono aggiunti i driver del dispositivo). Il gestore di I/O converte le richieste ricevute in una forma standard che si chiama **pacchetto di richiesta di I/O** (I/O request packet: IRP) che inoltra al primo driver nello stack per processarlo. Dopo che ogni driver ha processato l'IRP, chiama il gestore di I/O per inoltrarlo al driver successivo nello stack, o, se il processo è finito, per completare l'operazione sull'IRP.

I completamenti possono avvenire in un contesto differente dalla richiesta originale di I/O. Per esempio, se un driver sta eseguendo la propria parte di un'operazione di I/O ed è forzato a bloccarsi per un lungo tempo, può accodare l'IRP al thread lavoratore per proseguire il processo nel contesto del sistema. Nel thread originale il driver restituisce uno stato che specifica che la richiesta di I/O è in corso, in modo che il thread possa continuare l'esecuzione in parallelo con l'operazione di I/O. IRP può anche essere elaborato nelle routine di interrupt del servizio e completato in un contesto arbitrario. Poiché una qualche elaborazione finale ha bisogno di avvenire nel contesto che ha iniziato l'I/O, il gestore di I/O usa una APC per eseguire il processo di completamento finale di I/O nel contesto del thread che ha dato l'avvio.

Il modello dello stack del dispositivo è molto flessibile. Mentre si costruisce uno stack del driver, vari driver hanno l'opportunità di inserirsi nello stack come **driver del filtro** (filter drivers). I driver del filtro hanno la possibilità di esaminare e potenzialmente modificare ogni operazione di I/O. La gestione del montaggio, della partizione e le operazioni di striping/mirroring sono esempi di funzionalità implementate usando i driver del filtro che operano al di sotto del file system nello stack. I driver del filtro del file system agiscono sopra il file system e sono usati per realizzare alcune funzionalità quali la gestione della memorizzazione gerarchica, la singola istanza di file per l'avvio remoto, e la conversione dinamica del formato. Terze parti usano anche un filtro del file system per implementare il rilevamento di virus.

I driver dei dispositivi di Windows XP sono scritti secondo le specifiche del modello di driver di Windows (WDM) che elenca i driver del dispositivo includendo la stratificazione dei driver del filtro, la condivisione del codice comune per gestire l'alimentazione, le richieste plug-and-play e la costruzione della corretta logica di cancellazione, ecc.

A causa della ricchezza di WDM, può essere un carico eccessivo di lavoro scrivere un driver completo WDM del dispositivo per ogni nuovo dispositivo hardware. Fortunatamente non è necessario a causa del modello di port/miniport. Per classi similari di dispositivi, quali i driver audio, i dispositivi SCSI ed i controller Ethernet, ogni istanza ad un dispositivo condivide un driver comune per quella classe, chiamato **driver port** (port driver). Il driver port realizza le operazioni

standard per la classe e poi chiama le procedure specifiche del driver nel **mini-port driver** del dispositivo per implementare funzionalità specifiche del dispositivo.

### 3.3.6 Gestore della cache

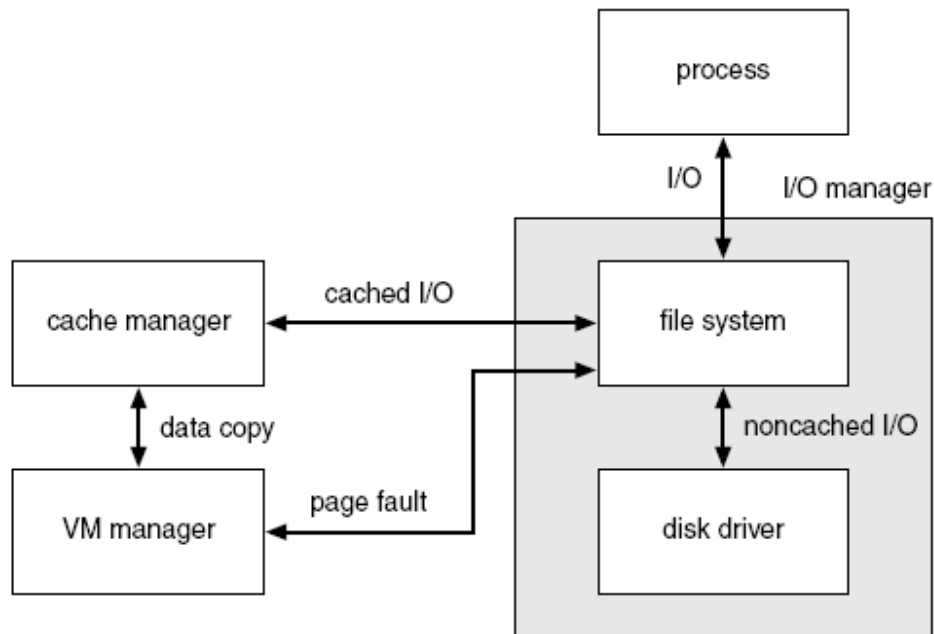
In molti sistemi operativi, l'operazione di cache è eseguita dal file system. Al contrario, Windows XP fornisce una funzione di cache centralizzata in cui il **gestore di cache** (cache manager) opera in stretto contatto con il gestore di VM per fornire servizi di cache per tutti i componenti sotto il controllo del gestore di I/O. La cache in Windows XP è basata sui file invece che sui blocchi grezzi.

La dimensione della cache cambia dinamicamente, in base alla quantità di memoria libera disponibile nel sistema. Si ricordi che i 2 GB superiori dello spazio dell'indirizzo del processo comprendono l'area di sistema che è disponibile nell'ambito di tutti i processi. Il gestore di VM assegna fino a metà di questo spazio alla cache di sistema e poi il gestore di cache mappa i file in questo spazio di indirizzo ed utilizza le capacità del gestore di VM per maneggiare il file I/O.

La cache è divisa in blocchi di 256 Kb in cui ogni blocco può contenere una vista, cioè una regione di memoria mappata di un file. Ogni blocco della cache è descritto da un **blocco di controllo dell'indirizzo virtuale** (virtual-address control block: VACB) che memorizza l'indirizzo virtuale e lo spiazzamento del file per la vista, come pure, usando tale vista, il numero di processi. I VACB risiedono in un singolo array mantenuto dal gestore della cache.

Per ogni file aperto, il gestore della cache mantiene un array-indice VACB che descrive l'operazione di cache sull'intero file; tale array ha un elemento per ogni pezzo del file da 256 KB; quindi, per esempio, un file da 2 MB dovrebbe avere un array indice ad 8 elementi VACB. Un elemento nell'array indice VACB punta al VACB stesso, se quella porzione di file è in cache; in caso contrario a null. Quando il gestore di I/O riceve la richiesta di lettura di un file a livello utente, il gestore di I/O manda un IRP allo stack del driver del dispositivo in cui il file risiede. Il file system tenta di esaminare i dati richiesti nel gestore della cache (a meno che la richiesta richieda specificamente una lettura non proveniente dalla cache). Il gestore della cache calcola quale elemento dell'array indice VACB di quel file corrisponde allo piazzamento del byte della richiesta. L'elemento o punta alla vista in cache o è non valido; in questo ultimo caso, il gestore della cache assegna un blocco della cache, e l'elemento corrispondente nell'array VACB, e mappa la vista nel blocco della cache. Il gestore della cache tenta poi di copiare i dati dal file mappato nel buffer del chiamante; se la copia riesce, l'operazione è completata.

Se la copia fallisce, ciò avviene a causa di un fault di pagina che spinge il responsabile di VM ad inviare una richiesta di lettura, non proveniente dalla cache, al gestore di I/O. Il gestore di I/O invia un'altra richiesta allo stack del driver, chiedendo, questa volta, un'operazione di *paginazione*, che esclude il gestore della cache e legge i dati direttamente dal file nella pagina allocata per il gestore della cache; al completamento, VACB è regolato per puntare alla pagina. I dati, ora in cache, sono copiati nel buffer del chiamante e l'originaria richiesta di I/O viene completata. La Figura 6 mostra una descrizione di queste operazioni.



**Figura 6.** File I/O.

Quando è possibile, per le operazioni sincrone sui file in cache, I/O è gestito dal **meccanismo di I/O veloce** (fast I/O mechanism) che mette in parallelo il normale I/O basato su IRP, ma esegue direttamente chiamate nello stack del driver invece che passare in un IRP. Siccome non ci sono IRP coinvolti, l'operazione non dovrebbe bloccarsi per un lungo periodo di tempo e non può essere accodata ad un thread lavoratore. Di conseguenza, quando l'operazione raggiunge il file system e chiama il gestore della cache, l'operazione fallisce nel caso in cui le informazioni non siano già in cache. Il gestore di I/O tenta poi l'operazione usando il percorso del normale IRP.

Un'operazione di lettura a livello kernel si comporta in modo simile, ad eccezione del fatto che ai dati si può accedere direttamente dalla cache, invece che copiarli in un buffer nello spazio dell'utente. Per usare i metadati del file system (strutture dati che descrivono il file system), il kernel, per leggere i metadati, usa l'interfaccia di mappatura del gestore della cache, mentre, per modificarli, il file system usa l'interfaccia a cui punta (pinning) il gestore della cache. Il **Pinning** (puntare) di una pagina blocca la pagina in un frame della pagina della memoria fisica, in modo che il gestore di VM non possa muoversi o paginare verso l'esterno la pagina. Dopo l'aggiornamento dei metadati, il file system chiede al gestore della cache di liberare (unpin) la pagina. Una pagina modificata è marcata come sporca, di conseguenza il gestore di VM trasferisce la pagina su disco e il metadato è memorizzato in un normale file.

Per migliorare le prestazioni, il gestore della cache mantiene una piccola storia delle richieste di lettura e dalla storia cerca di predire le richieste future. Se il gestore della cache trova un modello che assomiglia alle tre precedenti richieste, tipo un accesso sequenziale in avanti o all'indietro, esso

immette i dati dalla cache prima che l'applicazione presenti la richiesta successiva. In questo modo l'applicazione trova i propri dati già in cache senza dover aspettare l'I/O del disco. Le funzioni API di Win32 `OpenFile` e `CreateFile` possono essere passate al flag `FILE_FLAG_SEQUENTIAL_SCAN`, che è un suggerimento al gestore della cache per cercare di immettere 192 KB prima delle richieste del thread. Tipicamente, Windows XP esegue operazioni su blocchi di 64 KB o 16 pagine; e, pertanto, la read-ahead è tre volte la quantità normale.

Il gestore della cache è pure responsabile di comunicare al gestore di VM di ripulire il contenuto della cache. Il normale comportamento del gestore della cache è di porre in cache l'operazione di write-back: accumula scritture per 4-5 secondi e poi risveglia il thread di scrittura in cache. Quando è necessaria l'operazione di cache write-through, un processo può porre un flag all'apertura di un file, o può chiamare una esplicita funzione di pulitura della cache.

Un processo di scrittura veloce potrebbe potenzialmente riempire tutte le pagine libere in cache prima che il thread di scrittura in cache abbia la possibilità di risvegliare e trasferire le pagine nel disco. Lo scrittore della cache impedisce ad un processo di operare nel sistema nel modo appena descritto. Quando la quantità di memoria libera diventa piccola, il gestore della cache blocca temporaneamente i processi che tentano di scrivere i dati e risveglia il thread dello scrittore in cache per trasferire le pagine nel disco. Se il processo di scrittura veloce è attualmente un reindirizzamento della rete in un file system di rete, il bloccarlo troppo a lungo potrebbe provocare ai trasferimenti di rete la perdita del sincronismo e la ritrasmissione che, in tal caso, sprecherebbe larghezza di banda. Per impedire tale spreco, i reindirizzatori di rete possono istruire il gestore della cache a limitare l'accumulo di scritture in cache.

Siccome un file system di rete deve spostare i dati fra un disco e l'interfaccia di rete, il gestore della cache fornisce anche un'interfaccia DMA per spostare i dati direttamente, evitando di copiarli tramite un buffer intermedio.

### **3.3.7 Monitor della sicurezza**

La gestione centralizzante delle entità del sistema nel gestore dell'oggetto permette l'uso di un meccanismo uniforme per la convalida dell'accesso run-time e controlla nel sistema ogni entità accessibile dall'utente. Ogni volta che si apre un processo, un handle di un oggetto, il **security reference monitor** (controllore del riferimento di sicurezza: SRM) controlla il token di sicurezza del processo e la lista di controllo di accesso dell'oggetto per stabilire se il processo ha i diritti necessari.

SRM è inoltre responsabile della manipolazione dei privilegi nei token di sicurezza. Sono richiesti agli utenti privilegi speciali per compiere operazioni di tipo backup o restore sui file system, per superare certi controlli come amministratore, per compiere il debug di processi, ecc. I token, a causa di restrizioni nei loro privilegi, possono anche essere marcati in modo da non potere accedere ad oggetti che sono disponibili alla maggior parte degli utenti. Token con restrizioni sono principalmente usati per limitare i danni che possono essere provocati dall'esecuzione di codice non fidato.

Un'altra responsabilità di SRM è di tenere nota degli eventi di verifica della sicurezza. Una sicurezza di livello C2 richiede che il sistema abbia la capacità di rilevare ed annotare tutti i tentativi di accesso alle risorse del sistema in modo che sia più facile tenere traccia dei tentativi di accesso non autorizzato. Poiché SRM è responsabile dei controlli di accesso, esso genera la maggior parte dei record di verifica nel log dell'evento riguardante la sicurezza.

### 3.3.8 Plug-and-play e gestori dell'alimentazione

Il sistema operativo usa il **gestore plug-and-play** (plug-and-play manager: PnP) per riconoscere ed adattarsi ai cambiamenti nella configurazione hardware. Affinché PnP funzioni, sia il dispositivo che il driver devono supportare lo standard PnP; il gestore PnP riconosce automaticamente i dispositivi installati e rileva i cambiamenti nei dispositivi durante il funzionamento del sistema. Il gestore tiene anche traccia delle risorse usate da un dispositivo, come pure di risorse potenziali che potrebbero essere usate e si prende carico di caricare i driver adatti. Questa gestione delle risorse hardware, in primo luogo degli interrupt e delle aree di memoria di I/O, ha lo scopo di determinare la configurazione hardware in cui tutti i dispositivi possono funzionare.

Per esempio, se il dispositivo B può usare l'interrupt 5 ed il dispositivo A può usare il 5 od il 7, allora il gestore PnP assegnerà il 5 a B ed il 7 ad A. Nelle versioni precedenti, l'utente poteva essere costretto a rimuovere il dispositivo A e modificarlo per poter usare l'interrupt 7, prima di installare il dispositivo B. L'utente doveva quindi studiare le risorse di sistema prima di installare nuovo hardware e scoprire o ricordarsi quali dispositivi stavano usando quali risorse hardware. La proliferazione di schede PCMCIA, di stazioni di ancoraggio (dock station) per computer portatili, di periferiche USB, IEEE 1394, Infiniband ed altri dispositivi inseribili a caldo (hot-pluggable) impone anche il supporto di risorse configurabili dinamicamente.

Il gestore PnP maneggia questa riconfigurazione dinamica nel modo seguente. In primo luogo, ottiene una lista dei dispositivi da ogni driver del bus (ad esempio, PCI, USB). Carica il driver installato (o, se necessario, ne installa uno) e invia una richiesta `add-device` al driver adatto per ogni dispositivo. Il gestore di PnP calcola le assegnazioni ottimali delle risorse e invia `start-device` ad ogni driver con l'assegnazione della risorsa per il dispositivo. Se un dispositivo necessita di essere riconfigurato, il gestore di PnP manda una richiesta `query-stop`, che chiede al driver se il dispositivo può essere temporaneamente disabilitato; in caso affermativo, allora tutte le operazioni pendenti vengono completate ed a quelle nuove viene impedito di incominciare. Poi, il gestore di PnP invia una richiesta di `stop` e può riconfigurare il dispositivo con un'altra richiesta di `start-device`.

Il gestore di PnP supporta anche altre richieste, quali `query-remove`, che è usata quando l'utente diventa pronto ad espellere una PCCARD, e opera in modo simile a `query-stop`. La richiesta di `surprise-remove` è usata quando un dispositivo sbaglia o, più probabilmente, quando un utente rimuove una PCCARD senza prima fermarla. La richiesta `remove` dice al driver di terminare l'uso del dispositivo e di liberare tutte le risorse ad esso allocate.

Windows XP supporta una gestione sofisticata dell'alimentazione. Sebbene questa possibilità sia utile nei sistemi domestici per ridurre l'assorbimento di corrente di corrente, l'applicazione primaria è rivolta alla facilità di uso (accesso più rapido) ed al prolungamento della durata della batteria dei computer portatili. Il sistema ed i dispositivi individuali possono venire abilitati verso la modalità a basso consumo: chiamata modalità di attesa (`sleep`) o `standby`, quando non sono in uso, in modo che la batteria si occupi soprattutto della ritenzione dei dati nella memoria fisica (RAM). Il sistema può risvegliarsi quando vengono ricevuti dei pacchetti dalla rete, una chiamata telefonica dal modem, o un utente apre un portatile o preme un tasto funzione dell'alimentazione. Windows XP può anche ibernare un sistema memorizzando il contenuto della memoria fisica su disco ed arrestando completamente la macchina per poi ristabilire, in un secondo momento, le funzionalità del sistema prima di continuare l'esecuzione.

Sono supportate ulteriori strategie di riduzione dell'assorbimento di corrente. Piuttosto che far eseguire un ciclo al processore quando la CPU è a riposo, Windows XP sposta il sistema verso uno

stato che richiede un assorbimento inferiore di corrente. Se la CPU è sottoutilizzata, Windows XP riduce la velocità di clock della CPU, in modo da risparmiare corrente.

### 3.3.9 Registro

Windows XP mantiene molte delle proprie informazioni di configurazione in un database interno chiamato **registro** (registry). Un database del registro si chiama **hive** (alveare). Ci sono hive separati per le informazioni di sistema, per le preferenze di default dell'utente, per l'installazione del software e per la sicurezza. Poiché per poter avviare il sistema, sono richieste informazioni dal **sistema hive** (system hive), il gestore del registro è implementato come componente del codice eseguibile.

Ogni volta che il sistema si avvia con successo, esso salva il system hive come *l'ultimo ben conosciuto*. Se l'utente installa del software, quale un driver del dispositivo, che produce una configurazione del system hive che non permetterà l'avvio, l'utente può usualmente avviarlo mediante l'ultima configurazione ben conosciuta.

Il danneggiamento del system hive, dovuta ad installazione di applicazioni e di driver di terze parti, è così comune che Windows XP ha un componente chiamato **recupero del sistema** (system restore) che periodicamente salva gli hive, come pure altre configurazioni del software come i driver ed i file di configurazione eseguibili, in modo che il sistema possa essere ripristinato ad una condizione precedentemente funzionante nel caso in cui il sistema si avvii, ma non funzioni più come previsto.

### 3.3.10 Avvio

L'avvio di un PC in cui è caricato Windows XP inizia con l'alimentazione dell'hardware e quando la BIOS inizia l'esecuzione dalla ROM. La BIOS identifica il **dispositivo di sistema** (system device) da avviare, carica ed esegue il caricatore dal disco, il quale riceve informazioni sufficienti sul formato del file system per avviare il programma NTLDR dal direttorio radice del dispositivo. NTLDR viene usato per determinare quale **dispositivo di avvio** (boot device) contiene il sistema operativo; poi carica la libreria HAL, il kernel e il system hive dal dispositivo di avvio. A partire dal system hive determina quali driver del dispositivo sono necessari per avviare il sistema (*boot driver*) e li carica; Infine NTLDR comincia l'esecuzione del kernel.

Il kernel inizializza il sistema e crea due processi. Il **processo di sistema** (system process) che contiene tutti i worker thread interni e non opera mai in modalità utente. Il primo processo in modalità utente è SMSS, che è simile al processo INIT (inizializzazione) in UNIX: SMSS esegue una ulteriore inizializzazione del sistema, incluso l'avvio dei file paginanti e il caricamento dei driver dei dispositivi, e crea i processi WINLOGON e CSRSS. CSRSS è un sottosistema di Win32 e WINLOGON carica il resto del sistema, compreso il sottosistema di sicurezza LSASS ed i rimanenti servizi necessari per il funzionamento del sistema.

Il sistema ottimizza il processo di avvio precaricando i file da disco basandosi sugli avvii precedenti. I modelli di accesso al disco al momento dell'avvio vengono anche usati per riordinare i file di sistema sul disco in modo da ridurre il numero richiesto di operazioni di I/O. I processi richiesti

per avviare il sistema sono ridotti raggruppando i servizi in un unico processo; tutti questi metodi contribuiscono ad una notevole riduzione del tempo del caricamento del sistema.

D'altra parte, le capacità di sleep e di ibernazione di Windows XP permettono agli utenti di spegnere il loro computer e di riprendere rapidamente dal punto in cui lo avevano lasciato, rendendo il tempo di caricamento meno importante di prima.

## 4 Sottosistemi di Ambiente

I sottosistemi di ambiente sono processi in modalità utente stratificati sopra i servizi del codice eseguibile di Windows XP nativo per metterlo in grado di eseguire programmi sviluppati per altri sistemi operativi, incluso Windows a 16 bit, MS-DOS e POSIX. Ogni sottosistema ambientale fornisce un singolo ambiente applicativo.

Windows XP usa il sottosistema Win32 come principale sistema operativo ed inizia così tutti i processi. Quando si esegue un'applicazione, il sottosistema Win32 chiama il gestore della VM per caricare il codice eseguibile dell'applicazione. Il gestore della memoria restituisce a Win32 una condizione di stato che indica il tipo di eseguibile. Nel caso non fosse un eseguibile nativo di Win32, Win32 controlla l'ambiente per verificare se il sottosistema ambientale appropriato sta funzionando; se il sottosistema non è in funzione, viene avviato un processo in modalità utente in cui il sottosistema prende il controllo dell'avvio della applicazione.

I sottosistemi di ambiente usano la funzionalità LPC per fornire i servizi del sistema operativo ai processi client. L'architettura del sottosistema di Windows XP mantiene le applicazioni distinte dalle procedure API provenienti da ambienti diversi. Per esempio, una applicazione Win32 non può eseguire una chiamata di sistema POSIX perché solo un sottosistema ambientale può essere associato ad ogni processo.

Poiché ogni sottosistema è eseguito come processo separato in modalità utente, un arresto in uno di essi non ha conseguenze su altri processi. L'unica eccezione è Win32 che fornisce tutte le funzionalità alla tastiera, al mouse e alla visualizzazione grafica; nel caso fallisca, il sistema è effettivamente disabilitato e richiede un riavvio.

L'ambiente Win32 suddivide le applicazioni in categorie: o grafiche o basate su interfaccia a caratteri, dove un'*applicazione basata sull'interfaccia a caratteri* è tale che l'uscita interattiva vada ad una finestra basata su una interfaccia a caratteri (comando). Win32 trasforma il risultato di un'applicazione basata su interfaccia a caratteri in una rappresentazione grafica nella finestra di comando. Questa trasformazione è facile: ogni volta che si chiama una procedura di output, il sottosistema ambientale chiama, a sua volta, una procedura di Win32 per visualizzarne il testo. Poiché l'ambiente Win32 effettua questa funzione per tutte le finestre basate su interfaccia a caratteri, può trasferire il testo sullo schermo fra le finestre tramite buffer di tastiera (clipboard). Questa trasformazione funziona sia per applicazioni MS-DOS che per le applicazioni POSIX basate sulla linea di comando.

### 4.1 Ambiente MS-DOS

L'ambiente MSDOS non ha la complessità degli altri sottosistemi di ambiente di Windows XP ed è fornito da una applicazione Win32 chiamata **macchina virtuale DOS** (Virtual Dos Machine: VDM) che è un processo in modalità utente, paginato e gestito come qualunque altra applicazione

di Windows XP. La VDM ha un'**unità di istruzione-esecuzione** (instruction-execution unit) per eseguire o emulare istruzioni Intel 486. La VDM inoltre fornisce procedura per emulare la ROM BIOS di MS-DOS e i servizi di interrupt software (int 21), inoltre ha i driver del dispositivo virtuale per lo schermo, la tastiera e le porte di comunicazione. La VDM è basata sul codice sorgente di MS-DOS 5.0 e assegna almeno 620 KB di memoria per l'applicazione.

La shell di comando di Windows XP è un programma che crea una finestra che assomiglia ad un ambiente MS-DOS e può fare funzionare eseguibili sia a 16 che a 32 bit. Quando un'applicazione MS-DOS è in funzione, la shell di comando avvia un processo VDM per eseguire il programma.

Se Windows XP è in funzione su un processore compatibile IA32, l'applicazione grafica MS-DOS funziona in modalità a schermo pieno e le applicazioni basate su caratteri possono funzionare sia a schermo pieno che in una finestra, ma non tutte le applicazioni MSDOS funzionano mediante VDM. Per esempio, alcune applicazioni MS-DOS accedono direttamente all'hardware del disco, e quindi non riescono a funzionare in Windows XP perché l'accesso al disco ha delle limitazioni per proteggere il file system; in generale, le applicazioni MS-DOS che eseguono accessi diretti all'hardware non riusciranno a funzionare sotto Windows XP.

Siccome MS-DOS non è un ambiente multitasking, alcune applicazioni sono state scritte in modo tale da accaparrarsi la CPU. Per esempio, l'uso dei cicli di attesa attiva può causare ritardi o pause nell'esecuzione. Lo scheduler nel dispatcher del kernel rileva tali ritardi ed automaticamente limita l'uso della CPU, ma questo può indurre l'applicazione in questione a funzionare in modo errato.

## **4.2 Ambiente a 16-Bit di Windows**

L'ambiente di esecuzione di Win16 è fornito da una VDM che comprende software supplementare denominato *Windows on Windows* (WOW32 per applicazioni a 16 bit) che fornisce le procedure del kernel di Windows 3.1 e procedure di interfaccia (stab) per il gestore della finestra e per le funzioni dell'interfaccia grafica (GDI). Le procedure di interfaccia richiamano sottoprocedure appropriate di Win32, convertendo, indirizzi a 16 bit in indirizzi a 32 bit. Le applicazioni che fanno affidamento sulla struttura interna del gestore della finestra a 16 bit o GDI potrebbero non funzionare, poiché la realizzazione sottostante Win32 è, naturalmente, differente da una vera implementazione di Windows a 16 bit.

WOW32 può operare in multitask con altri processi in Windows XP, ma assomiglia, sotto molti aspetti, a Windows 3.1. Solo un'applicazione alla volta di Win16 può funzionare; tutte le applicazioni sono a singolo thread e risiedono nello stesso spazio di indirizzamento e tutte condividono la stessa coda di input. Queste caratteristiche implicano che un'applicazione che smette di ricevere l'input bloccherà tutte le altre applicazioni Win16, proprio come in Windows 3.x ed una sola applicazione Win16 può arrestare le altre applicazioni Win16 corrompendo lo spazio di indirizzamento. Tuttavia, possono coesistere ambienti multipli di Win16 usando, dalla linea di comando, il comando *start/separate win16application*.

Esistono relativamente poche applicazioni a 16 bit che gli utenti hanno bisogno di continuare ad usare in Windows XP, ma alcune di esse includono programmi di installazione comuni (setup). Quindi, l'ambiente WOW32 continua ad esistere principalmente perché un certo numero di applicazioni a 32 bit non possono essere installate su Windows XP senza di esso.

## **4.3 Ambiente Windows a 32-Bit su IA64**

L'ambiente nativo di Windows su IA64 usa indirizzi a 64 bit e l'insieme di istruzioni native IA64. Per eseguire programmi IA32 in questo ambiente si richiede uno strato di interfaccia per tradurre le chiamate a 32 bit di Win32 nelle corrispondenti chiamate a 64 bit, proprio come è richiesto nei sistemi IA32 per le applicazioni a 16 bit; in tal modo Windows a 64 bit supporta l'ambiente WOW64. Le realizzazioni Windows a 32 bit ed a 64 bit sono essenzialmente identiche ed il processore IA64 permette l'esecuzione diretta delle istruzioni di IA32 e, di conseguenza, WOW64 fornisce un livello di compatibilità più elevato di WOW32.

#### **4.4 Ambiente Win32**

Il sottosistema principale di Windows XP è Win32, che esegue applicazioni Win32 che controlla la tastiera, il mouse e lo schermo I/O. Siccome è l'ambiente di controllo, esso è progettato per essere estremamente robusto. A differenza dell'ambiente Win16, ogni processo di Win32 ha una propria coda di input. Il gestore della finestra spedisce ogni input al sistema nella coda di input del processo appropriato, in modo che un processo guasto non blocchi l'input di altri processi.

Il kernel di Windows XP fornisce inoltre elaborazione preemptive in multitask, per permettere all'utente di terminare applicazioni che non sono andate a buon fine o che non sono più necessarie. Inoltre, Win32 convalida tutti gli oggetti prima di usarli, per impedire arresti che potrebbero altrimenti accadere se un'applicazione cercasse di utilizzare un handle non valido o errato. Il sottosistema Win32 verifica il tipo di oggetto a cui punta un handle prima di usarlo. I conteggi di riferimento, mantenuti dal responsabile dell'oggetto, impediscono la cancellazione di oggetti mentre sono ancora in uso e di utilizzarli se sono stati cancellati.

Per ottenere un livello elevato di compatibilità con sistemi Windows 95/98, Windows XP permette agli utenti di specificare quali applicazioni individuali siano in funzione, mediante l'uso di uno **strato di adattamento** (shim layer) che modifica le API di Win32 per meglio approssimare il comportamento che ci si aspetta dalle vecchie applicazioni. Per esempio, alcune applicazioni si aspettano una particolare versione del sistema che non funziona sulle nuove versioni. Frequentemente le applicazioni hanno banchi latenti che si mettono in mostra grazie a cambiamenti nell'esecuzione. Per esempio, l'uso della memoria, dopo averla liberata, può causare guasti solo se cambia l'ordine, impartito dallo heap, di riutilizzare la memoria, o se un'applicazione può fare previsioni su quali errori possono essere restituiti da una procedura, o sul numero di bit validi in un indirizzo. Far funzionare un'applicazione con gli aggiustamenti abilitati per Windows 95/98, induce il sistema a fornire un comportamento molto più simile a Windows 95/98, sebbene con prestazioni ridotte ed interoperabilità limitata con le altre applicazioni.

#### **4.5 Il sottosistema POSIX**

Il sottosistema di POSIX è progettato per far funzionare applicazioni POSIX scritte in modo da seguire lo standard POSIX, che è basato sul modello UNIX. Le applicazioni POSIX possono essere avviate dal sottosistema di Win32 o da un'altra applicazione POSIX che usa il server del sottosistema POSIX: `PSXSS.EXE`, la libreria di collegamento dinamico di POSIX: `PSXDLL.DLL` ed il gestore di sessione della console POSIX: `POSIX.EXE`.

Sebbene lo standard POSIX non specifichi la stampa, nelle applicazioni POSIX si possono utilizzare stampanti in modo trasparente tramite il meccanismo di reindirizzamento di Windows XP. Le applicazioni POSIX hanno accesso a qualsiasi file system nel sistema Windows XP e l'ambiente POSIX impone permessi di tipo UNIX sugli alberi del direttorio.

A causa di problemi di schedulazione, il sistema POSIX, in Windows XP, non è consegnato con il sistema, ma è disponibile separatamente per i sistemi professionali desktop e per i server. Fornisce un più alto livello di compatibilità con le applicazioni UNIX, rispetto alle versioni precedenti di NT. Nell'ambito delle applicazioni UNIX comunemente disponibili, la maggior parte si compila e funziona senza alcun cambiamento con l'ultima versione di Interix.

## **4.6 Logon e sottosistemi di sicurezza**

Prima che un utente possa accedere agli oggetti di Windows XP, l'utente deve essere autenticato dal servizio di logon: WINLOGON che è responsabile di rispondere al comando di attenzione imperativa (Control-Alt-Delete). Il comando di attenzione imperativa è un meccanismo richiesto per evitare che un'applicazione agisca come un cavallo di Troia (Trojan Horse). Solo WINLOGON può intercettare questa sequenza per mostrare uno schermo di benvenuto (logon), cambiare le password e bloccare la workstation. Per essere autenticato, un utente deve avere un account e fornire la parola d'accesso per quell'account; in alternativa, un utente si collega usando una smart card ed un numero di identificazione personale, conformi alle politiche di sicurezza in vigore per il dominio.

Il sottosistema dell'autorità di sicurezza locale (LSASS) è il processo che genera token di accesso per descrivere gli utenti nel sistema. Chiama una **procedura di autenticazione** (authentication package) per fornire autenticazione mediante informazioni provenienti dal sottosistema di logon o del server di rete. Tipicamente, la procedura di autenticazione controlla semplicemente, in un database locale, le informazioni sull'account e sulla correttezza della password. Il sottosistema di sicurezza genera poi il token di accesso con l'identificatore dell'utente, contenente i privilegi appropriati, i limiti di quota e l'identificatore del gruppo. Ogni qualvolta l'utente tenta di accedere ad un oggetto nel sistema, come mediante l'apertura di un handle per l'oggetto, il token di accesso viene passato al controllo di sicurezza di riferimento, che controlla i privilegi e le quote. La procedura di autenticazione di default per i domini di Windows XP è Kerberos. LSASS ha inoltre la responsabilità di realizzare la politica di sicurezza come password sicure, autenticare gli utenti ed eseguire la crittografia dei dati e delle chiavi.

## **5 File System**

Storicamente, i sistemi MS-DOS hanno usato il file system della tabella di allocazione del file (FAT). Il file system FAT a 16 bit presenta parecchie limitazioni, tra cui: frammentazione interna, limitazione del formato a 2 GB e mancanza di protezione di accesso ai file. Il file system FAT a 32 bit ha risolto i problemi di frammentazione e di dimensioni, ma le prestazioni e le caratteristiche sono ancora deboli in confronto ai moderni file system. Il file system NTFS è molto migliore ed è stato progettato per includere molte caratteristiche, tra cui il recupero dei dati, la sicurezza, la tolleranza agli errori, il supporto per grandi file e file system, flussi di dati multipli, nomi UNICODE, file sparsi, crittografia, diari, copie ombra del volume e compressione dei file.

Windows 2000 XP continua, tuttavia, ad usare la FAT16 per leggere i floppy ed altri supporti rimovibili; inoltre, malgrado i vantaggi di NTFS, FAT32 continua ad essere importante per l'interoperabilità dei supporti con i sistemi Windows 95/98. Windows XP supporta ulteriori tipi di file system i formati dei supporti comuni dei CD e DVD.

## 5.1 Struttura interna di NTFS

L'entità fondamentale di NTFS è il volume che viene creato dall'utilità di gestione del disco logico di Windows XP, ed è basato su una partizione del disco logico. Un volume può occupare una porzione di disco, o un intero disco, o estendersi su più dischi.

NTFS non si occupa di singoli settori del disco, ma invece usa i cluster come unità di allocazione del disco. Un **cluster** corrisponde ad un certo numero di settori del disco ed è una potenza di 2. La dimensione del cluster è configurata quando un file system NTFS è formattato. La dimensione di default dei cluster coincide con la dimensione del settore per volumi fino a 512 MB, 1 KB per volumi fino a 1 GB, 2 KB per volumi fino a 2 GB e 4 KB per volumi maggiori. La dimensione del cluster è molto più piccola di quella dei sistemi FAT a 16 bit e tale piccolo formato riduce la quantità di frammentazione interna. Per esempio, si consideri un disco da 1.6 GB con 16.000 file; se si usa un file system FAT16, 400 MB possono andare persi a causa della frammentazione interna poiché la dimensione del cluster è di 32 KB, mentre in NTFS, solo 17 MB andranno persi se si memorizzano gli stessi file.

NTFS usa i **numeri di cluster logico** (logical cluster numbers: LCN) come indirizzi del disco e li assegna numerando i cluster dall'inizio alla fine del disco. Con questo schema, il sistema può calcolare lo spiazzamento del disco (in byte) moltiplicando LCN per la dimensione del cluster.

Un file NTFS non è un semplice flusso di byte come in MS-DOS o in UNIX; invece, è un oggetto strutturato che consiste di **attributi** (attributes) scritti. Ogni attributo di un file è un flusso indipendente di byte che può essere creato, cancellato, letto e scritto. Alcuni tipi di attributo sono standard per tutti i file, compreso il nome del file (o i nomi, se il file possiede degli pseudonimi quali un nome abbreviato in MS-DOS), la data di creazione e il descrittore di sicurezza che specifica il controllo di accesso. I dati dell'utente sono memorizzati negli *attributi dei dati*.

La maggior parte dei file tradizionali hanno un attributo *non specificato* (*unnamed*) che contiene tutti i dati del file. Tuttavia, i flussi di dati aggiuntivi possono essere creati con nomi espliciti. Per esempio, i file di Macintosh, memorizzati in un server Windows XP, sono gestiti come un flusso di dati con nome. L'interfaccia IProp del Common Object Model (Modello di Oggetto Comune, COM) usa un flusso di dati con nome per memorizzare le proprietà in file normali, incluse le miniature (thumbnails) delle immagini. In generale, gli attributi possono essere aggiunti in base alle necessità e ad essi si accede mediante la sintassi *nome del file:attributo*. NTFS restituisce la dimensione dell'attributo senza nome solo in risposta ad operazioni di file-query, come quando si esegue il comando `dir`.

Ogni file NTFS è descritto da uno o più record in un array memorizzato in un file speciale chiamato tabella del file principale (Master File Table, MFT). La dimensione di un record è determinata alla creazione del file system e varia tra 1 e 4 KB. I piccoli attributi sono memorizzati nel record stesso della MFT e sono chiamati **attributi residenti** (resident attributes). I grandi attributi, quale i dati anonimi (bulk) senza nome, chiamati **attributi non residenti** (nonresident attributes), sono memorizzati in uno o più **estensioni** (extent) contigue su disco e un puntatore in ogni estensione è memorizzato nel record MFT. Nel caso di un piccolo file, anche l'attributo dei dati

può mettere l'estensione nel record MFT. Se un file ha molti attributi, o se è notevolmente frammentato e in tal caso ha bisogno di molti puntatori che puntano a tutti i frammenti, un record MFT potrebbe non essere sufficientemente grande; in tal caso, il file è descritto da un record che si chiama il **record di base del file** (base file record), che contiene i puntatori ai record di overflow che contengono puntatori ed attributi supplementari.

Ogni file in un volume NTFS ha un'identificatore unico, chiamato **riferimento del file** (file reference) che è una quantità a 64 bit in cui 48 bit costituiscono il numero del file e 16 bit un numero progressivo. Il numero del file è il numero del record (cioè l'alloggiamento dell'array) nella MFT che descrive il file. Il numero progressivo viene incrementato ogni volta che viene riutilizzato un elemento nella MFT e tale numero permette a NTFS di eseguire controlli di consistenza interni, tipo prendere un vecchio riferimento ad un file cancellato dopo che l'elemento nella MFT era stato riutilizzato per un nuovo file.

### **5.1.1 L'albero di NTFS B+**

Come in MS-DOS ed in UNIX, lo spazio dei nomi in NTFS è organizzato con direttori gerarchici, ciascuno dei quali usa una struttura dati chiamata **albero B+** (B+ tree) per memorizzare un indice dei nomi dei file in quel direttorio. Si usa un albero B+ perché elimina il costo di riorganizzazione dell'albero ed ha la proprietà che la lunghezza di ogni cammino dalla radice dell'albero ad una foglia è la stessa. La **radice indice** (index root) di un direttorio contiene il livello superiore dell'albero B+. Per un grande direttorio, tale livello superiore contiene i puntatori alle estensioni del disco, il quale mantiene il resto dell'albero. Ogni elemento nel direttorio contiene il nome ed il riferimento del file, come pure una copia della marca di tempo e la dimensione del file presa dagli attributi residenti del file nella MFT. Copie di queste informazioni sono memorizzate nel direttorio, in modo che sia efficiente generare una lista del direttorio. Tutti i nomi del file, dimensioni e tempi in cui è avvenuto l'aggiornamento, sono disponibili dal direttorio stesso, in modo che non vi sia necessità di raccogliere questi attributi dagli elementi MFT di ogni file.

### **5.1.2 Metadata NTFS**

I metadati del volume NTFS sono memorizzati in file; il primo file è MFT, il secondo file, che viene usato durante il recupero nel caso MFT sia danneggiato, contiene una copia dei primi 16 elementi della MFT, i file successivi sono usati per scopi particolari ed includono il file di log, il file del volume, la tabella di definizione dell'attributo, il direttorio radice, il file bitmap, il file di avvio ed il file dei cluster difettosi. Il **file del volume** (volume file) contiene il nome del volume, la versione di NTFS con cui è stato formattato il volume, ed un bit che indica se il volume è rovinato e deve essere controllato ai fini della consistenza. La **tabella di definizione dell'attributo** (attribute-definition table) indica quali tipi di attributi sono usati nel volume, e quali operazioni possono essere eseguite su ognuna delle tabelle.

Il **direttorio radice** (root directory) è il direttorio di livello più elevato nella gerarchia del file system, il **file bitmap** (bitmap file) indica quali cluster nel volume sono assegnati ai file e quali sono liberi, mentre il **file di avvio** (boot file) contiene il codice di caricamento di Windows XP e deve essere situato ad un indirizzo particolare del disco in modo che possa essere ritrovato facilmente da un semplice loader di avvio in ROM. Il file di avvio contiene pure l'indirizzo fisico della MFT; infine,

il **file dei cluster difettosi** (bad-cluster file) tiene traccia di tutte le zone difettose nel volume e NTFS usa questa registrazione per il recupero dagli errori.

## 5.2 Recupero

In molti semplici file system, un guasto all'alimentazione nel momento sbagliato può danneggiare le strutture dati del file system in modo talmente grave da mescolare i dati dell'intero volume. Molte versioni di UNIX memorizzano metadati ridondanti su disco, e li recuperano in caso di guasto con il programma fsck per controllare tutte le strutture dati del file system e per ristabilirle forzatamente ad uno stato consistente; il loro ristabilimento spesso coinvolge la cancellazione di file danneggiati e la cancellazione di dati dal cluster in cui erano stati scritti assieme ai dati dell'utente, ma non correttamente registrati nelle strutture metadati del file system. Il controllo può essere un processo lento e può far perdere significative quantità di dati.

### 5.2.1 Il file di log di NTFS

NTFS adotta un approccio differente per la robustezza del file system; in NTFS, tutti gli aggiornamenti della struttura dati del file system avvengono all'interno delle transazioni. Prima di alterare una struttura dati, la transazione scrive una registrazione del log che contiene informazioni redo e undo. Dopo aver cambiato la struttura dati, la transazione scrive una registrazione commit nel log per indicare che ha avuto successo.

In seguito ad un guasto, il sistema può ristabilire le strutture dati del file system in uno stato consistente elaborando i record di log; dapprima eseguendo di nuovo le operazioni delle transazioni commit e poi annullando le operazioni per le transazioni che non hanno avuto successo prima del guasto. Periodicamente (solitamente ogni 5 secondi), viene scritta nel log una registrazione di controllo; il sistema non ha bisogno delle registrazioni di log, eseguite prima del controllo per recuperare da un guasto, esse possono essere scartate in modo che il file di log non si sviluppi senza alcun limite. La prima volta, dopo l'avvio del sistema in cui si accede ad un volume NTFS, NTFS esegue automaticamente il recupero del file system.

Questo schema non garantisce che il contenuto del file dell'utente sia corretto dopo un guasto; assicura solo che le strutture dati del file system (i file di metadati) siano intatte e riflettano un qualche stato di consistenza esistente prima del guasto. Si potrebbe estendere lo schema di transazione per comprendere i file dell'utente e la Microsoft potrebbe farlo in futuro.

Il log viene memorizzato all'inizio del volume, nel terzo file dei metadati ed è creato di dimensione fissa massima quando viene formattato il file system. È composto da due sezioni: **l'area di log** (logging area), che è una coda circolare delle registrazioni di log, e **l'area di riavvio** (restart area), che mantiene informazioni sul contesto, quali la posizione nell'area di log da cui NTFS dovrebbe far partire una lettura durante un recupero. Infatti, l'area di riavvio mantiene due copie delle informazioni correlate, in modo che sia ancora possibile il recupero nel caso che una copia venga danneggiata durante un guasto.

La funzionalità di log è fornita dal **servizio di log-file** (log-file service) di Windows XP, il quale, oltre a scrivere i record di log ed eseguire le azioni di recupero, tiene traccia dello spazio libero nel file di log. Se lo spazio libero diventa troppo piccolo, il servizio di log-file accoda le transazioni in sospenso e NTFS ferma tutte le nuove operazioni di I/O. Dopo il completamento dell'operazione,

NTFS chiama il gestore della cache per liberarsi di tutti i dati, poi azzerà il file di log ed esegue le transazioni in coda.

### **5.3 Sicurezza**

La sicurezza di un volume NTFS discende dal modello dell'oggetto di Windows XP. Ogni file NTFS fa riferimento ad un descrittore di sicurezza che contiene il token di accesso del proprietario del file e ad una lista di controllo dell'accesso che contiene i privilegi di accesso assegnati ad ogni utente che può accedere al file.

Nelle normali operazioni, NTFS non fa rispettare i permessi nell'attraversamento dei direttori dei nomi di percorso del file, tuttavia, a causa della compatibilità con POSIX, questi controlli possono essere abilitati. I controlli di attraversamento sono più costosi poiché la moderna suddivisione dei nomi di percorso dei file usa la coincidenza dei prefissi invece che aprire, componente per componente, i nomi dei direttori.

### **5.4 Gestione del volume e tolleranza agli errori**

`FtDisk` è il driver del disco tollerante agli errori (fault tolerant) di Windows XP e, una volta installato, fornisce parecchi modi per combinare lettori di dischi multipli in un unico volume logico, in modo da migliorare le prestazioni, la capacità, o l'affidabilità.

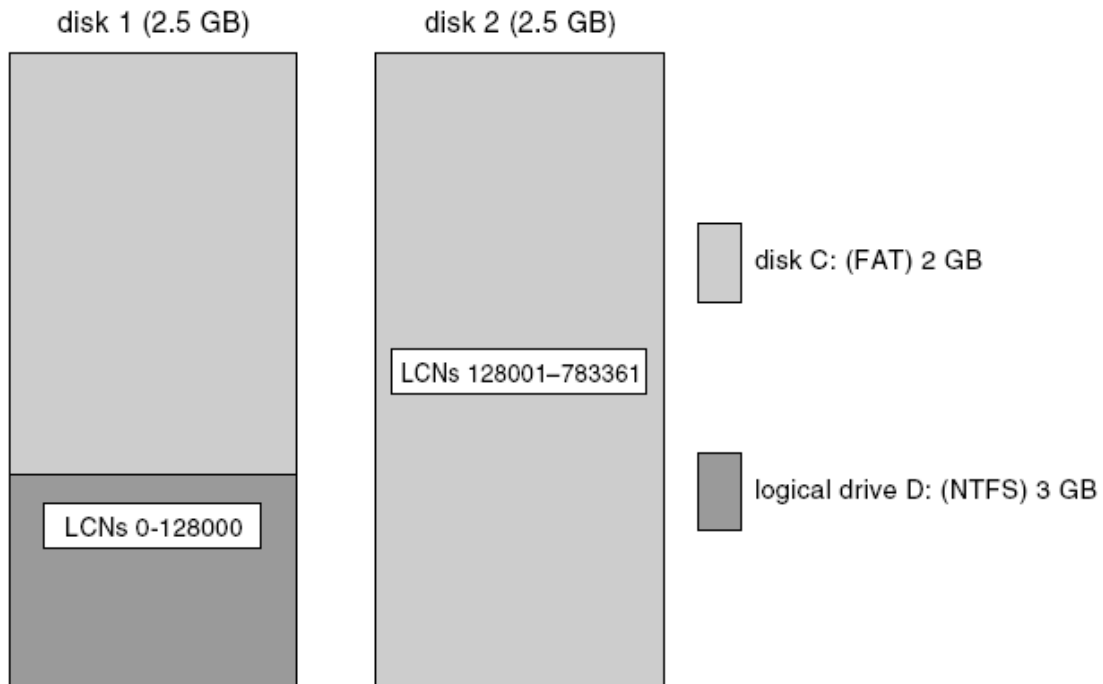
#### **5.4.1 Volume set**

Un modo per combinare dischi multipli è di concatenarli logicamente per formare un grande volume logico, come mostrato in Figura 7. In Windows XP, questo volume logico è chiamato **volume set** e può consistere fino ad un massimo di 32 partizioni fisiche. Un volume set, contenente un volume NTFS, può venire esteso senza disturbare i dati già memorizzati nel file system. I metadati di tipo bitmap nel volume NTFS vengono semplicemente estesi per coprire lo spazio appena aggiunto; NTFS continua ad usare lo stesso meccanismo LCN che viene usato per un disco fisico singolo e il driver `FtDisk` fornisce la mappa dallo spiazamento di un volume logico in quello di un particolare disco.

#### **5.4.2 Stripe set**

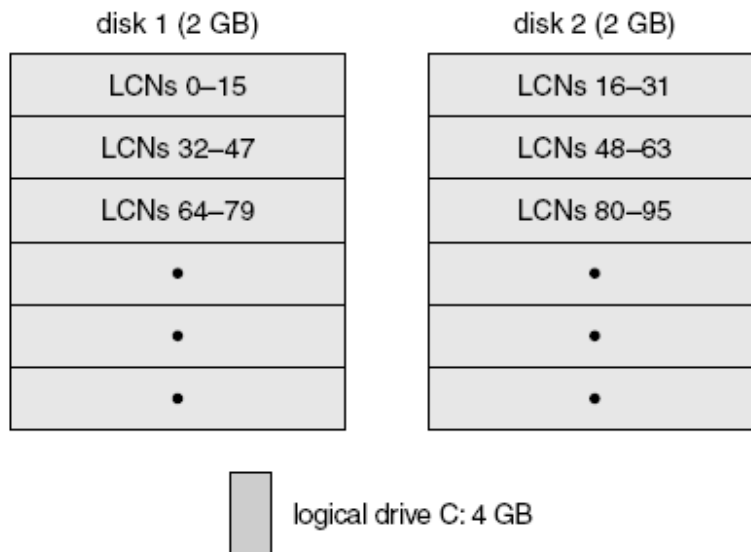
Un altro modo per combinare partizioni fisiche multiple è di intercalare i propri blocchi in modalità round-robin per formare ciò che si chiama **stripe set** (insieme di strisce), come si vede in Figura 8. Questo schema è anche detto RAID di livello 0, o **disk striping**. `FtDisk` usa una dimensione di stripe (striscia) di 64 KB: i primi 64 KB del volume logico sono memorizzati nella prima partizione fisica, i secondi 64 KB del volume logico sono memorizzati nella seconda partizione fisica e così via, finché ogni partizione non abbia contribuito a 64 KB di spazio.

Poi, l'allocazione ricopre il primo disco, allocando il secondo blocco di 64 KB. Uno stripe set forma un grande volume logico e la disposizione fisica può migliorare la larghezza di banda di I/O, poiché, in occasione di un grande I/O, tutti i dischi possono trasferire i dati in parallelo.



**Figura 7.** Volume set su due dischi.

disk= disco  
local drive = drive locale



**Figura 8.** Stripe set su due dischi.

disk= disco

local drive = drive locale

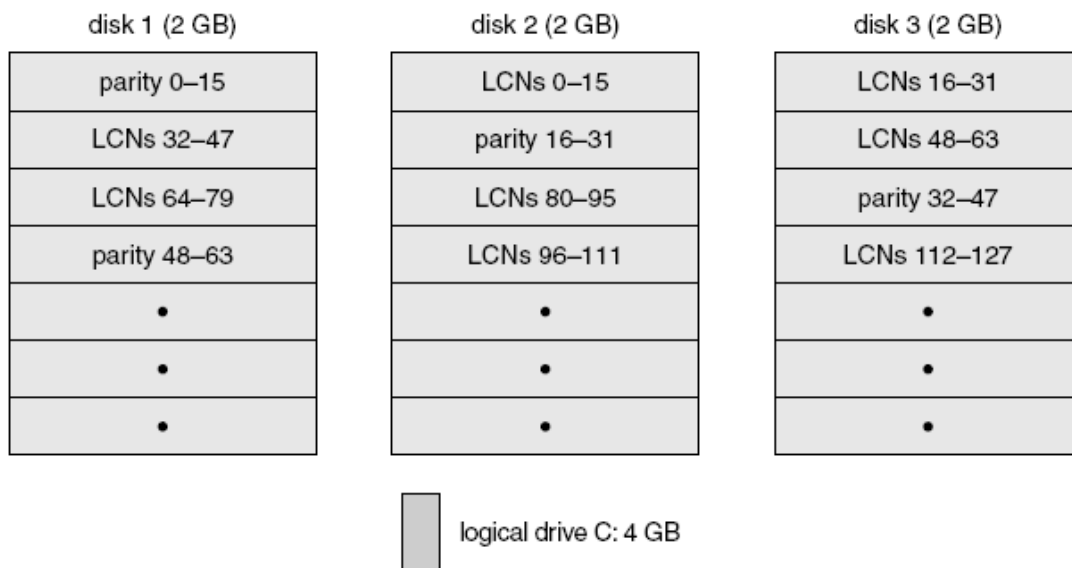
### 5.4.3 Stripe set con parità

Una variazione a questo metodo è lo **stripe set con parità** (stripe set with parity), mostrato in Figura 9. Questo schema è anche chiamato RAID livello 5. Se lo stripe set è composto da otto dischi, allora, per ognuna delle sette strisce di dati, su sette dischi separati, si trova una banda di parità sull'ottavo disco e tale banda di parità contiene l'*exclusive or*, a livello di byte, delle strisce di dati. Se una delle otto strisce viene distrutta, il sistema può ricostruire i dati calcolando l'*exclusive or* dalle sette rimanenti. Questa capacità di ricostruire i dati rende l'array di dischi meno propenso a perdere i dati nel caso di guasto ai dischi

Si noti che un aggiornamento alla striscia di dati richiede anche il ricalcolo della parità. Sette scritture concorrenti su sette differenti strisce di dati richiederebbero pure l'aggiornamento di sette strisce di parità. Se le strisce di parità fossero tutte sullo stesso disco, questo disco avrebbe un sovraccarico di I/O di sette volte. Per evitare di creare questo collo di bottiglia, si suddividono le strisce di parità fra tutti i dischi, assegnandole in modalità round-robin. Per costruire uno stripe set con parità, abbiamo bisogno di un minimo di tre partizioni uguali in tre dischi separati (Figura 9).

### 5.4.4 Mirror del disco, insiemi di mirror, RAID e insiemi duplex

Uno schema ancor più robusto è chiamato **mirror del disco** (disk mirroring) o RAID di livello 1 che è rappresentato in Figura 10. Un **insieme di mirror** (mirror set) comprende due partizioni di ugual dimensione su due dischi, tali che il contenuto dei dati sia identico. Quando un'applicazione



**Figura 9.** Stripe set con parità su tre dischi.

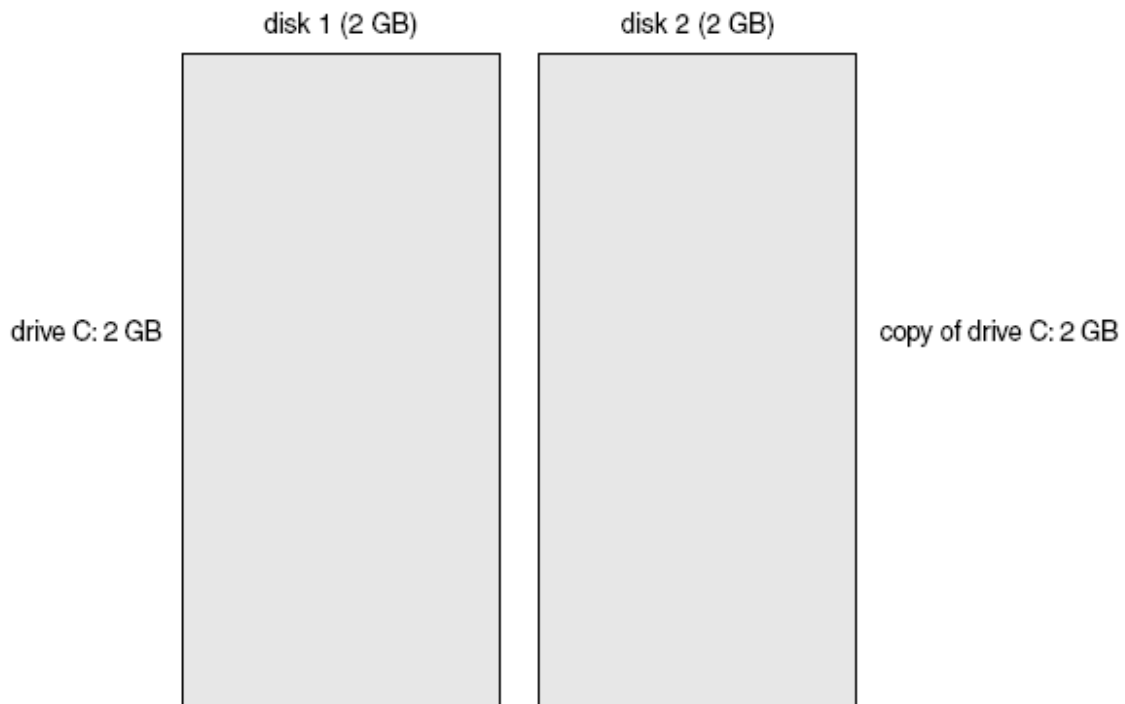
disk= disco; parity = parità; local drive = drive locale

scrive i dati in un insieme di mirror, FtDisk scrive i dati in entrambe le partizioni. Se una partizione commette un errore, FtDisk ha un'altra copia memorizzata in modo sicuro nel mirror. Gli insiemi di mirror possono anche migliorare le prestazioni, poiché le richieste di lettura possono essere suddivise fra i due mirror, assegnando ad ogni mirror metà del carico di lavoro. Per proteggerci contro i guasti di un disk controller, possiamo collegare i due dischi del mirror a due controller separati: questa combinazione si chiama **insieme duplex** (duplex set).

#### 5.4.5 Sector sparing e rimappatura del cluster

FtDisk usa una tecnica hardware chiamata sector sparing, nel caso in cui vi siano settori del disco rovinati, e NTFS usa una tecnica software chiamata rimappatura del cluster. **Sector sparing** è una possibilità hardware fornita da molti dischi. Quando un disco viene formattato, crea una mappa dai numeri del blocco logico nei settori buoni del disco e lascia, pure, settori supplementari non mappati, da usare come parti di ricambio. Se un settore viene a mancare, FtDisk istruisce il drive a sostituirlo con un settore di ricambio. La **rimappatura del cluster** (cluster remapping) è una tecnica software usata dal file system; se un blocco del disco si guasta, NTFS lo sostituisce con un blocco differente e non allocato, cambiando tutti i puntatori interessati nella MFT; inoltre tiene nota che il blocco difettoso non dovrebbe essere mai assegnato ad un qualsiasi file.

Quando un blocco del disco diventa difettoso, il risultato usuale è la perdita di dati; il sector sparing o la rimappatura del cluster possono essere combinati in volumi a tolleranza di errore per mascherare il guasto di un blocco del disco.



**Figura 10.** Mirror set su due drive.

disk = disco; drive; copy of drive = copia del drive

Se una lettura fallisce, il sistema ricostruisce i dati mancanti leggendo il mirror o calcolando *exclusive or* o la parità in uno stripe set con parità. I dati ricostruiti sono memorizzati in una nuova locazione che è ottenuta dal sector sparing o dalla rimappatura del cluster.

## **5.5 Compressione e crittografia**

NTFS può eseguire la compressione dei dati su singoli file o su tutti i file di un direttorio. Per comprimere un file, NTFS suddivide i dati del file in **unità di compressione** (compression units), che sono blocchi di 16 cluster contigui. Quando scrive in ogni unità di compressione, applica un algoritmo di compressione dei dati e, se il file risultante occupa meno di 16 cluster, viene memorizzata la versione compressa. In lettura, NTFS può stabilire se i dati erano stati compressi: in caso affermativo, la lunghezza dell'unità di compressione memorizzata è meno di 16 cluster. Per migliorare le prestazioni in lettura di unità contigue di compressione, NTFS preleva e decompime in anticipo rispetto alle richieste dell'applicazione.

Per file sparsi o per file che contengono principalmente zero, NTFS usa un'altra tecnica per risparmiare spazio: i cluster che contengono solo zero, poiché non sono mai stati scritti, non vengono assegnati o memorizzati su disco. Invece, gli intervalli (gap) sono lasciati nella sequenza dei numeri

del cluster virtuale memorizzati nell'elemento della MFT per il file. Nella lettura di un file, se si trova un intervallo nei numeri del cluster virtuale, NTFS riempie proprio con zero la porzione del buffer del chiamante; una tecnica simile è usata anche da UNIX.

NTFS supporta la crittografia dei file; singoli file, o interi direttori, possono essere specificati per venire crittografati. Il sistema di sicurezza gestisce le chiavi usate, e un servizio di recupero della chiave è disponibile per ritrovare le chiavi perse.

## **5.6 Punti di montaggio**

I punti di montaggio sono, in NTFS, una forma di collegamento simbolico ai direttori e forniscono un meccanismo più flessibile per gli amministratori nell'organizzare i volumi del disco, piuttosto che fornire semplicemente i nomi globali (tipo le lettere dei drive). I punti di montaggio sono realizzati come collegamento simbolico ai dati associati che contengono il vero nome del volume. Infine, i punti di montaggio soppianderanno completamente le lettere dei drive e ci sarà una lunga transizione dovuta alla dipendenza di molte applicazioni dallo schema lettera associata al drive.

## **5.7 Change journal**

NTFS mantiene un giornale (journal) in cui descrive tutti i cambiamenti che sono avvenuti nel file system. I servizi in modalità utente possono ricevere notifiche di cambiamenti nel giornale e poi identificare quali file sono stati cambiati. Il servizio di indicizzazione del contenuto usa questa modalità per individuare i file che devono essere indicizzati di nuovo. Il servizio di replica dei file usa il giornale dei cambiamenti per identificare i file che devono essere replicati in rete.

## **5.8 Copie ombra del volume**

Windows XP ha la capacità di portare un volume in uno stato conosciuto per poi creare una copia ombra che può essere usata per salvare una parte consistente del volume. Realizzare una copia ombra di un volume è una forma di copy-on-write, in cui i blocchi modificati, dopo la creazione di una copia ombra, hanno il proprio contenuto nascosto (stashed) durante la copia. Per ottenere uno stato consistente per il volume si richiede la cooperazione delle applicazioni dal momento che il sistema non può sapere quando i dati, usati dall'applicazione, sono in uno stato stabile e quando l'applicazione potrebbe venire di nuovo eseguita in modo sicuro.

La versione server di Windows XP usa copie ombra per mantenere efficientemente le vecchie versioni dei file memorizzate sui server. Ciò permette agli utenti di vedere i documenti memorizzati sui server nel momento in cui sono usciti in istanti precedenti. L'utente può usare questa caratteristica per recuperare file che sono state cancellati per errore, o semplicemente vedere una versione precedente del file, senza accedere ad un nastro di salvataggio.

## 6 Rete

Windows XP supporta sia la connessione peer-to-peer che quella client-server e possiede funzionalità di gestione della rete. I componenti di rete in Windows XP forniscono il trasporto dei dati, la comunicazione tra processi, la condivisione di file in rete, e la capacità di inviare job di stampa verso stampanti remote.

### 6.1 Interfacce di rete

Per descrivere la connessione di rete in Windows XP, si fa riferimento a due interfacce di rete interne chiamate **specifica dell'interfaccia del dispositivo di rete** (network device interface specification: NDIS) e **interfaccia del driver di trasporto** (transport driver interface: TDI). L'interfaccia NDIS è stata sviluppata nel 1989 da Microsoft e da 3Com per separare gli adattatori di rete dai protocolli di trasporto, in modo che uno di essi possa essere cambiato senza interferire con l'altro. NDIS risiede nell'interfaccia fra il controllo di collegamento dei dati e gli strati di controllo di accesso al supporto nel modello OSI e permette a molti protocolli di operare su molti adattatori differenti di rete. In termini di modello OSI, TDI è l'interfaccia fra lo strato di trasporto (strato 4) e lo strato di sessione (strato 5); tale interfaccia permette a qualsiasi componente dello strato di sessione di usare un qualunque meccanismo di trasporto disponibile. (Una modalità simile conduceva, in UNIX, al meccanismo dei flussi). TDI supporta sia il trasporto basato sulla connessione che quello senza connessione e ha le funzioni per trasmettere qualunque tipo di dati.

### 6.2 Protocolli

Windows XP realizza i protocolli di trasporto come driver che possono essere caricati e scaricati dinamicamente dal sistema, anche se in pratica il sistema deve essere riavviato dopo un cambiamento. Windows XP viene fornito con parecchi protocolli di rete.

#### 6.2.1 Blocco del messaggio del server

Il protocollo del **blocco del messaggio del server** (server-message-block: SMB) è stato per la prima volta introdotto in MS-DOS 3.1. Il sistema usa il protocollo per inviare richieste di I/O in rete. Il protocollo SMB ha quattro tipi di messaggio: i messaggi `Session control` sono comandi che iniziano e terminano una connessione reindirizzata (redirector) verso una risorsa condivisa nel server. Un redirector usa i messaggi `File` per accedere ai file del server. Il sistema usa i messaggi `Printer` per inviare dati ad una coda di stampa remota e per ricevere informazioni di stato; il messaggio `Message` è usato per comunicare con un'altra workstation. Il protocollo SMB è stato pubblicato con il nome di **Common Interface File System** (CIFS) ed è supportato da un certo numero di sistemi operativi.

### **6.2.2 Sistema di base di Input/Output di rete**

Il **sistema base di input/output di rete** (network basic input/output system: NetBIOS) è un'interfaccia di astrazione hardware per le reti, analoga all'interfaccia di astrazione hardware progettata per i PC su cui funziona MS-DOS. NetBIOS, sviluppata all'inizio degli anni 80, è divenuta una interfaccia standard di programmazione della rete, ed è usata per assegnare nomi logici in rete, per stabilire connessioni logiche o **sessioni** (sessions) fra due nomi logici in rete e per supportare trasferimenti di dati affidabili in una sessione tramite richieste di NetBIOS o di SMB.

### **6.2.3 Interfaccia utente estesa di NetBIOS**

L'**interfaccia utente estesa NetBIOS** (NetBIOS extended user interface: NetBEUI) è stata introdotta dall'IBM nel 1985 come semplice ed efficiente protocollo di rete in grado di supportare fino a 254 macchine. È il protocollo di default per la rete peer di Windows 95 e di Windows per Workgroups. Windows XP usa NetBEUI quando vuole condividere risorse con queste reti. Fra le limitazioni di NetBEUI si annovera il fatto che usa, come indirizzo, il nome reale di un computer e che non supporta l'instradamento.

### **6.2.4 Protocollo di Controllo della Trasmissione/Protocollo Internet**

La suite Transmission Control Protocol/Internet Protocol (protocollo di controllo della trasmissione/protocollo Internet: TCP/IP), usati in Internet, è divenuto lo standard di fatto dell'infrastruttura di rete. Windows XP usa TCP/IP per connettersi ad un'ampia varietà di sistemi operativi e di piattaforme hardware. Il pacchetto TCP/IP di Windows XP include il semplice protocollo di gestione della rete (SNMP), il protocollo di configurazione dell'host dinamico (DHCP), il servizio del nome di Internet di Windows (WINS) ed il supporto NetBIOS.

### **6.2.5 Protocollo di tunneling punto-a-punto**

Il **protocollo di tunneling punto-a-punto** (point-to-point tunneling protocol: PPTP) è un protocollo fornito da Windows XP per permettere la comunicazione fra i moduli del server di accesso remoto, funzionanti su macchine Windows XP Server, con altri sistemi client che sono connessi ad Internet. I server di accesso remoto possono cifrare i dati inviati sulla connessione e supportano in Internet **reti private virtuali** (virtual private networking: VPN).

### **6.2.6 Protocolli Novell NetWare**

I protocolli Novell NetWare (servizio di datagramma IPX sullo strato di trasporto SPX) sono ampiamente usati nelle LAN dei PC. Il protocollo NWLink di Windows XP collega le reti NetBIOS alle reti NetWare. Congiuntamente ad un reindirizzatore (quale il servizio client di Microsoft per Netware o il client di Novell Netware per Windows), questo protocollo permette ad un client di Windows XP di connettersi ad un server NetWare.

## **6.2.7 Presentazioni multimediali distribuite su Web e protocollo di versione**

Le presentazioni multimediali distribuite su Web e la gestione delle versioni (WebDAV) sono un protocollo basato su http per creare presentazioni collaborative in rete. Windows XP costruisce un reindirizzatore WebDAV nel file system in cui viene sviluppato direttamente e può funzionare assieme ad altre funzionalità, quali la crittografia. I file personali possono in tal modo essere memorizzati in modalità sicura in un luogo pubblico.

## **6.2.8 Protocollo Appletalk**

Il **protocollo Appletalk** è stato progettato come connessione a basso costo da Apple per permettere ai calcolatori Macintosh di condividere i file. I sistemi Windows XP possono condividere i file e le stampanti con computer Macintosh tramite Appletalk, se un server Windows XP in rete fa funzionare i servizi di Windows per il pacchetto Macintosh.

## **6.3 Meccanismi di elaborazione distribuita**

Sebbene Windows XP non sia un sistema operativo distribuito, esso supporta applicazioni distribuite. I meccanismi che supportano l'elaborazione distribuita su Windows XP includono: NetBIOS, redirezioni con nome e mailslot, socket di windows, RPC e scambio di dati dinamico in rete (NetDDE).

### **6.3.1 NetBIOS**

In Windows XP, le applicazioni NetBIOS possono comunicare in rete usando NetBEUI, NWLink, o TCP/IP.

### **6.3.2 Named pipe**

Le named pipe (pipe con nome) sono un meccanismo di connessione orientato alla messaggistica e furono originariamente sviluppate come interfaccia ad alto livello per le connessioni NetBIOS in rete. Un processo può anche usare le named pipe per comunicare con altri processi sulla stessa macchina. Poiché alle named pipe si accede tramite l'interfaccia del file system, i meccanismi di sicurezza usati per gli oggetti del file si applicano anche ad esse.

Il nome di una named pipe è definito con un formato chiamato **convenzione della denominazione uniforme** (uniform naming convention: UNC). Un nome UNC assomiglia ad un tipico nome di file remoto. Il formato di un nome UNC è `\\server_name\share_name\x\y\z`, dove `server_name` identifica un server in rete; `share_name` identifica qualsiasi risorsa che viene resa disponibile agli utenti della rete, quali direttori, file, named pipe e stampanti; e la parte `x\y\z` è un normale percorso del file.

### **6.3.3 Mailslot**

Le **mailslot** sono un meccanismo di messaggistica senza connessione. Sono inaffidabili quando si accede attraverso la rete, in quanto un messaggio inviato ad una mailslot può andare perso prima che il vero destinatario lo riceva. Le mailslot sono usate in applicazioni relative alle trasmissioni, come trovare componenti in rete; sono anche usate dal servizio browser dei computer di Windows.

### **6.3.4 Winsock**

**Winsock** è la API per i socket di Windows XP ed è un'interfaccia dello strato di sessione che in gran parte è compatibile con i socket UNIX, ma con alcune estensioni aggiuntive di Windows XP; fornisce un'interfaccia standardizzata a molti protocolli di trasporto che possono avere differenti schemi di indirizzamento, in modo che qualsiasi applicazione Winsock possa funzionare su qualsiasi stack del protocollo compatibile con Winsock.

### **6.3.5 Chiamata di procedura remota**

Una chiamata di procedura remota (RPC) è un meccanismo client-server che permette ad una applicazione su una macchina di eseguire una chiamata della procedura da codificare su un'altra macchina. Il client chiama una procedura locale: una **procedura madre** (stub routine) che impacchetta i propri argomenti in un messaggio e li invia, attraverso la rete, ad un particolare processo del server. Dal lato client, la procedura madre poi si blocca e, nel frattempo, il server spacchetta il messaggio, chiama la procedura, impacchetta i risultati ottenuti in un messaggio e li rispedisce alla procedura madre del client. La procedura madre si sblocca, riceve il messaggio, spacchetta i risultati di RPC e li restituisce al chiamante. Un tale impacchettamento di argomenti è talvolta chiamato **mettere in ordine** (marshalling). Il meccanismo RPC di Windows XP segue l'ambiente standard ampiamente usato di elaborazione distribuita per i messaggi RPC, in modo che i programmi scritti per RPC di Windows XP siano altamente portabili. Lo standard RPC è descritto con ampio dettaglio; esso nasconde molte differenze di architettura fra i computer, come le dimensioni dei numeri binari e l'ordine dei byte e dei bit nelle word di un computer, specificando formati di dati standard per i messaggi RPC.

Windows XP può trasmettere messaggi RPC tramite NetBIOS, o Winsock su reti TCP/IP, o named pipe su reti Lan Manager. La funzione LPC, discussa precedentemente, è simile a quella RPC, tranne che in LPC i messaggi sono passati fra due processi che funzionano sullo stesso computer.

### **6.3.6 Linguaggio di definizione dell'interfaccia di Microsoft**

È noioso e propenso agli errori scrivere codice per fare il marshal e trasmettere argomenti nel formato standard, eseguire l'unmarshal ed eseguire la procedura remota, eseguire il marshal e inviare risultati ottenuti, ed infine eseguire l'unmarshal e restituire l'esito al chiamante. Fortunatamente, tuttavia, molto di questo codice può essere generato automaticamente mediante una semplice descrizione degli argomenti per poi restituire i risultati.

Windows XP fornisce il **linguaggio per la definizione di interfacce Microsoft** (Microsoft interface definition language) che descrive i nomi delle procedure remote, gli argomenti ed i risultati. Il compilatore del linguaggio genera le intestazioni dei file che dichiarano gli stub delle procedure remote, i tipi di dati per gli argomenti e i messaggi con il valore di ritorno; genera, pure, il codice sorgente per gli stub delle procedure, usate sul versante client, per eseguire l'unmarshal e per spedirle al server. Quando l'applicazione è collegata (linked), sono inclusi anche gli stub delle procedure e quando l'applicazione esegue lo stub dell'RPC, il codice generato gestisce il resto.

### **6.3.7 Modello di oggetto comune**

Il **modello dell'oggetto componente** (component object model: COM) è un meccanismo di comunicazione tra processi che è stato sviluppato per Windows. Gli oggetti COM forniscono un'interfaccia ben definita per gestire i dati nell'oggetto. Per esempio, COM è l'infrastruttura usata dalla tecnologia di **object linking and embedding** (OLE) per inserire fogli di calcolo elettronici nei documenti di WORD. Windows XP possiede un'estensione distribuita denominata **DCOM** che può essere usata in una rete che utilizza RPC per fornire un metodo trasparente di sviluppo per le applicazioni distribuite.

## **6.4 Reindirizzatori e server**

In Windows XP, un'applicazione può usare le API I/O per accedere ai file da un computer remoto come se fossero locali, una volta che quello remoto faccia funzionare un server CIFS come quello che si trova in Windows XP o nei sistemi precedenti di Windows. Un **reindirizzatore** (redirector) è un oggetto sul versante client che inoltra richieste I/O ai file remoti che verranno soddisfatte da un server. Per questioni di prestazioni e sicurezza, i reindirizzatori e i server funzionano in modalità kernel.

In maggior dettaglio, l'accesso ad una file remoto avviene nel modo seguente:

- L'applicazione chiama il gestore di I/O per richiedere che un file venga aperto con un nome del file nel formato standard UNC.
- Il gestore di I/O costruisce un pacchetto di richiesta I/O, come è descritto nel Paragrafo 3.3.5 di questo capitolo.
- Il gestore di I/O riconosce che l'accesso è per un file remoto, e chiama il driver **fornitore della convenzione di denominazione universale multipla** (multiple universal-naming-convention provider: MUP).
- MUP invia il pacchetto di richiesta di I/O in modo asincrono a tutti i reindirizzatori registrati.

- Un reindirizzatore che può soddisfare alla richiesta risponde al MUP. Per evitare di ripetere in futuro la stessa domanda a tutti i reindirizzatori, MUP usa una cache per ricordare quale reindirizzatore può maneggiare questo file.
- Il reindirizzatore invia la richiesta di rete al sistema remoto.
- I driver del sistema remoto di rete ricevono la richiesta e la passano al driver del server.
- Il driver del server passa la richiesta al driver locale adeguato al file system.
- Il driver appropriato del dispositivo viene chiamato per accedere ai dati.
- I risultati vengono restituiti al driver del server, che restituisce i dati al reindirizzatore richiedente; poi il reindirizzatore ritorna i dati all'applicazione chiamante tramite il gestore di I/O.

Un processo simile si presenta per applicazioni che usano le API di rete di Win32 invece dei servizi UNC, salvo che viene utilizzato un modulo chiamato router multi-provider, al posto di MUP.

Per questioni di portabilità, i reindirizzatori ed i server usano le API TDI per il trasporto in rete. Le richieste stesse sono espresse in un protocollo a più alto livello, che per default è il protocollo SMB menzionato nel Paragrafo 6.2 di questo capitolo. La lista dei reindirizzatori è mantenuta nel database del sistema di registro.

#### **6.4.1 File system distribuito**

I nomi UNC non sono sempre convenienti poiché i server di file multipli possono essere disponibili per servire lo stesso contenuto, e i nomi UNC includono esplicitamente il nome del server. Windows XP supporta un protocollo di **file system distribuito** (distributed file system: DFS) che permette ad un amministratore di rete di servire file provenienti da server multipli utilizzando un singolo spazio dei nomi distribuito.

#### **6.4.2 Reindirizzamento della cartella e messa in cache da parte del client**

Per migliorare l'uso del PC, per gli utenti che operano nel campo degli affari e che passano frequentemente da un computer ad un altro, Windows XP permette agli amministratori di assegnare agli utenti un **profilo mobile** (roaming profile) che mantiene le preferenze dell'utente e altre assegnazioni sui server. Il **reindirizzamento delle cartelle** (folder redirection) è quindi usato per memorizzare automaticamente i documenti e altri file dell'utente in un server. Ciò funziona bene finché uno dei computer non è più collegato alla rete, come un portatile in un aereo. Per fornire agli utenti un accesso non in linea ai file reindirizzati, Windows XP usa la **cache dalla parte del client** (client-side caching: CSC). CSC è usato quando è in linea per mantenere copie dei file del server al fine di avere prestazioni migliori. I file sono inviati al server non appena cambiano e se il computer si disconnette, i file sono ancora disponibili e l'aggiornamento del server è rinviato alla volta successiva in cui il computer è in linea con un collegamento di rete con buone prestazioni.

## 6.5 Domini

Molti ambienti in rete hanno gruppi naturali di utenti, quali gli studenti di un laboratorio di informatica in una scuola, o gli impiegati di un reparto commerciale. Frequentemente, si desidera che tutti i membri del gruppo possano accedere alle risorse condivise nei vari computer del gruppo. Per gestire i diritti di accesso globali in tali gruppi, Windows XP usa il dominio. Precedentemente, tali domini non avevano alcuna relazione con il sistema del nome del dominio DNS che mappa i nomi host di Internet in indirizzi IP; ora, invece, sono strettamente collegati.

Nello specifico, un dominio di Windows XP è un gruppo di workstation e server di Windows XP che condividono una politica comune di sicurezza e una database degli utenti. Dal momento che Windows XP ora usa il protocollo Kerberos per l'autenticazione e il trusting (certificazione autorizzazioni), un dominio Windows XP è equivalente ad un dominio Kerberos. Le precedenti versioni di NT usavano l'idea dei controllori primari e di salvataggio del dominio; ora tutti i server di un dominio sono controllori del dominio stesso. Inoltre, le versioni precedenti richiedevano la predisposizione di fiduciari a senso unico fra i domini. Windows XP usa un metodo gerarchico basato sul DNS e permette fiduciari transitivi che possono fluire su e giù nella gerarchia. Questo metodo riduce il numero dei fiduciari richiesti per gli  $n$  domini da  $n * (n - 1)$  a  $O(n)$ . Le workstation nel dominio si fidano del controllore del dominio e forniscono informazioni corrette sui diritti di accesso di ogni utente (tramite il token di accesso dell'utente). Tutti gli utenti hanno la capacità di limitare l'accesso alle proprie workstation, indipendentemente da ciò che il controllore del dominio possa dire.

### 6.5.1 Alberi e foreste del dominio

Siccome una ditta può avere molti reparti e una scuola può avere molte classi, è necessario gestire domini multipli entro una singola organizzazione. Un **albero del dominio** (domain tree) è una gerarchia contigua di nomi DNS. Per esempio, [bell-labs.com](http://bell-labs.com) potrebbe essere la radice dell'albero, con [research.bell-labs.com](http://research.bell-labs.com) e [pez.bell-labs.com](http://pez.bell-labs.com) come figli: cioè i domini *research* e *pez*. Una **foresta** (forest) è un insieme di nomi non contigui; un esempio potrebbero essere gli alberi [bell-labs.com](http://bell-labs.com) e/o [lucent.com](http://lucent.com). Una foresta, tuttavia, può essere fatta solo da un albero del dominio.

### 6.5.2 Relazioni fiduciarie

Le relazioni fiduciarie possono essere instaurate fra i domini in tre modi: a senso unico, transitive e a collegamento incrociato. Le versioni di NT fino alla versione 4.0 hanno permesso solo relazioni fiduciarie a senso unico. Una **relazione fiduciaria unidirezionale** (one-way trust) è esattamente ciò che il nome implica: si dice che il dominio A può fidarsi del dominio B; tuttavia, B non può fidarsi di A a meno che non sia configurata un'altra relazione. In una **relazione fiduciaria transitiva** (transitive trust), se A ha fiducia in B e B ha fiducia in C, allora A, B e C si fidano uno dell'altro poiché la fiducia transitiva è per default bidirezionale. Le fiducie transitive sono abilitate per default nei nuovi domini in un albero e possono essere configurate solo fra domini entro una foresta. Il terzo tipo: la **relazione fiduciaria a collegamento incrociato** (cross-link trust) è utile per ridurre il traffico di autenticazione. Supponiamo che i domini A e B siano nodi della foglia e che gli utenti di A usino spesso risorse di B. Se si usa una fiducia standard transitiva, le richieste di autenticazione devono attraversare fino all'antenato comune dei due nodi della foglia, ma se A e B

hanno stabilito una relazione di fiducia a collegamento incrociato, l'autenticazione potrà essere inviata direttamente all'altro nodo.

## **6.6 Direttorio attivo**

Il **direttorio attivo** (active directory) è la realizzazione da parte di Windows XP dei servizi di **protocollo leggero di accesso al direttorio** (lightweight directory-access protocol:LDAP). Il direttorio attivo memorizza le informazioni di topologia del dominio, mantiene gli utenti basati sul dominio, gli accrediti dei gruppi, le password e fornisce un deposito basato sul dominio per tecnologie come le **politiche** ed **intellimirror di gruppo**.

Gli amministratori usano politiche di gruppo per stabilire gli standard delle preferenze e del software per i desktop. Per molti gruppi di tecnologia dell'informazione, l'uniformità riduce drasticamente il costo di elaborazione. Intellimirror viene usato insieme alle politiche di gruppo per specificare quale software dovrebbe essere a disposizione per ogni categoria di utenti, installandolo anche automaticamente su richiesta di un sever della ditta.

## **6.7 Risoluzione del nome nelle reti TCP/IP**

In una rete IP, la **risoluzione dei nomi** (name resolution) è il processo mediante il quale si converte il nome di un computer in un indirizzo IP, come, per esempio, la risoluzione di [www.bell-labs.com](http://www.bell-labs.com) in 135.104.1.14. Windows XP fornisce parecchi metodi di risoluzione dei nomi, compreso il servizio dei nomi di Internet di Windows (WINS), la risoluzione dei nomi trasmesso, il sistema dei nomi dei domini (DNS), un file host ed un file LMHOSTS. Qui descriveremo solo WINS, anche se la maggior parte di questi metodi è usata da molti sistemi operativi.

Nel metodo WINS, due o più server WINS mantengono un database dinamico dei collegamenti tra nome e indirizzo IP, ed un software client per interrogare i server. Si utilizzano almeno due server, in modo che il servizio WINS possa sopravvivere al guasto di un server e in modo che il carico di lavoro di risoluzione dei nomi possa essere suddiviso tra più computer.

### **6.7.1 Protocollo dinamico di configurazione degli host**

WINS usa il protocollo dinamico di configurazione degli host (DHCP). DHCP aggiorna automaticamente le configurazioni degli indirizzi nel database WINS, senza intervento dell'amministratore o dell'utente. Opera nel modo seguente: quando un client DHCP si avvia, trasmette un messaggio discover; ogni server DHCP, che riceve il messaggio, risponde con un messaggio offer che contiene le informazioni di configurazione e un indirizzo IP per il client. Il client sceglie una delle configurazioni e invia un messaggio request al server DHCP selezionato. Il server DHCP risponde con le informazioni di configurazione e con l'indirizzo IP fornito in precedenza e con un **affitto** (lease) per quel indirizzo. L'affitto dà al cliente il diritto di usare l'indirizzo IP per un periodo di tempo specificato, A metà del quale, il client tenta di rinnovarlo. Se l'affitto non viene rinnovato, il client deve ottenerne uno nuovo.

## 7 Interfaccia del programmatore

Win32 API è l'interfaccia fondamentale rivolta alle capacità di Windows XP. Questo paragrafo descrive cinque aspetti principali di Win32 API: accesso agli oggetti del kernel, condivisione degli oggetti fra processi, gestione del processo, comunicazione tra processi e gestione della memoria.

### 7.1 Accesso agli oggetti del kernel

Il kernel di Windows XP fornisce molti servizi che possono essere usati dai programmi applicativi; i programmi applicativi ottengono questi servizi manipolando gli oggetti del kernel. Un processo può accedere ad un oggetto del kernel, detto XXX, chiamando la funzione `CreateXXX` per aprire un handle in XXX. Questo handle, nell'ambito del processo, è unico. In base all'oggetto che si apre, se la funzione `create` non va a buon fine, può restituire 0, o può restituire un costante speciale chiamata `INVALID_HANDLE_VALUE`. Un processo può chiudere qualsiasi handle mediante la chiamata della funzione `Close_Handle`, in tal modo il sistema può cancellare l'oggetto se il numero di processi che usano l'oggetto cala a 0.

### 7.2 Condivisione degli oggetti fra i processi

Windows XP fornisce tre modi per condividere gli oggetti fra i processi: nel primo modo il processo figlio eredita un handle dell'oggetto, il processo padre chiama la funzione `CreateXXX`, il padre fornisce una struttura `SECURITY_ATTRIBUTES` con il campo `bInheritHandle` posto su `TRUE`. Questo campo crea un handle ereditabile. Dopo che si crea il processo figlio, si passa un valore `TRUE` a `CreateProcess` (argomento della funzione `bInheritHandle`). La Figura 11 mostra un campione del codice che genera un handle del semaforo ereditato da un processo figlio.

```
...
SECURITY_ATTRIBUTES sa;
sa.nlength = sizeof(sa);
sa.lpSecurityDescriptor = NULL;
sa.bInheritHandle = TRUE;
Handl_a_semaphore = CreateSemaphore (&sa,1,1,
NULL); char command_line[132 ] ;
ostream ostring (command_line,
sizeof(command_line));
ostring << a_semaphore <<ends;
CreateProcess("another process.exe",command_line,
NULL,NULL, TRUE,...);
...
```

**Figura 11.** Codice figlio per condividere un oggetto che eredita un handle.

Assumendo che il processo figlio conosca quali handle siano condivisi, il padre ed il figlio possono realizzare la comunicazione tra processi tramite gli oggetti condivisi. Nell'esempio di Figura 11, il processo figlio ottiene il valore di handle dal primo argomento della riga di comando e poi condivide il semaforo con il processo padre.

Il secondo modo di condividere gli oggetti è che un processo assegni all'oggetto un nome al momento della creazione, e che il secondo processo apra il nome. Questo metodo ha due svantaggi: Windows XP non fornisce un modo per controllare se un oggetto con il nome scelto esiste già, dato che lo spazio dei nomi degli oggetti è globale, senza aver cura del tipo di oggetto. Per esempio, due applicazioni possono creare un oggetto chiamato *pipe*, ma si desiderano due oggetti distinti, e, possibilmente differenti.

Gli oggetti con nome hanno il vantaggio che processi indipendenti possono prontamente condividerli; il primo processo chiama una delle funzioni `CreateXXX` e fornisce un nome nel parametro `lpzName`. Il secondo processo ottiene un handle per condividere l'oggetto chiamando `OpenXXX` (o `CreateXXX`) con lo stesso nome, come mostrato nell'esempio di Figura 12.

```
//processo A
...
Handle_a_semaphore = CreateSemaphore(NULL,1,1,"MySEM1");
...
//processo B
...
Handle_b_semaphore = OpenSemaphore(SEMAPHORE_ALL_ACCESS,
                                   FALSE, "MySEM1");
...
```

**Figura 12.** Codice per condividere un oggetto dando uno sguardo al nome.

Il terzo modo di condividere gli oggetti è tramite la funzione `DuplicateHandle`; questo metodo richiede qualche meccanismo di comunicazione tra processi per passare l'handle duplicato. Una volta assegnato un handle ad un processo ed il valore di un handle entro quel processo, un secondo processo può ottenere un handle dello stesso oggetto e quindi condividerlo. Un esempio di questo metodo è mostrato in Figura 13.

```
...
//Il processo A vuole permettere al processo B
l'accesso ad un semaforo
//processo A
Handle_a_semaphore = CreateSemaphore(NULL,1,1,NULL);
//invia il valore del semaforo al processo B
//usa un messaggio o la memoria condivisa
...
//processo B
```

```

Handle b_semaphore;
DuplicateHandle(process_a, a_semaphore,
    GetCurrentProcess(), &b_semaphore,
    0, FALSE, DUPLICATE_SAME_ACCESS);
//usa b_semaphore per accedere al semaforo
...

```

**Figura 13.** Codice per la condivisione di un oggetto mediante il passaggio di un handle.

## 7.3 Gestione del processo

In Windows XP, un **processo** è una richiesta di esecuzione di un'applicazione e un **thread** è un'unità del codice che può essere schedulata dal sistema operativo; pertanto, un processo contiene uno o più thread. Un processo si inizia quando un qualche altro processo chiama la procedura `CreateProcess`.

Questa procedura carica tutte le librerie di collegamento dinamico usate dal processo e crea un **thread primario** (primary thread); thread supplementari possono venire creati dalla funzione `CreateThread`. Ogni thread è creato con il proprio stack, che usualmente è di 1 MB a meno che non sia specificato diversamente in un argomento di `CreateThread`. Siccome le funzioni run-time del C mantengono lo stato in variabili statiche, quali `errno`, un'applicazione multithread deve salvaguardarsi contro accessi non sincronizzati. La funzione avvolgente (wrapper) `beginthreadex` fornisce la sincronizzazione adatta.

### 7.3.1 Handle di richiesta

Ogni libreria di collegamento dinamico o ogni file eseguibile, caricato nello spazio di indirizzo di un processo, è identificata da un **handle di istanza** (instance handle). Il valore dell'handle di istanza è attualmente l'indirizzo virtuale dove è caricato il file. Un'applicazione può ottenere un handle per un modulo nel proprio spazio di indirizzi passando il nome del modulo a `GetModuleHandle`; se viene passato come nome `NULL`, viene restituito l'indirizzo base del processo. I 64 KB inferiori dello spazio degli indirizzi non vengono usati, in modo che un programma difettoso, che tenti di dereferenziare un puntatore `NULL`, ottenga una violazione di accesso.

Le priorità nell'ambiente di Win32 sono basate sul modello di schedulazione di Windows XP, ma non si possono scegliere tutti i valori di priorità. Win32 usi quattro classi di priorità: `IDLE_PRIORITY_CLASS` (livello di priorità 4), `NORMAL_PRIORITY_CLASS` (livello di priorità 8), `HIGH_PRIORITY_CLASS` (livello di priorità 13) e `REALTIME_PRIORITY_CLASS` (livello di priorità 24). I processi sono tipicamente membri della `NORMAL_PRIORITY_CLASS` a meno che il padre non fosse nella `IDLE_PRIORITY_CLASS`, o che un'altra classe non fosse stata specificata al momento della chiamata di `CreateProcess`. La classe di priorità di un processo può essere cambiata con la funzione `SetPriorityClass`, o tramite un argomento passato al comando `START`. Per esempio, il comando `START/REALTIME cbserver.exe` farà funzionare il programma `cbserver` in `REALTIME_PRIORITY_CLASS`. Solo gli utenti con il

privilegio di *priorità aumentata di schedulazione* possono muovere un processo in `REALTIME_PRIORITY_CLASS`; gli amministratori e gli utenti posseggono questo privilegio per default.

### 7.3.2 Regola di schedulazione

Quando un utente fa funzionare un programma interattivo, il sistema deve fornire delle prestazioni particolarmente buone per il processo. Per questo motivo, Windows XP ha una regola speciale di schedulazione per i processi nella `NORMAL_PRIORITY_CLASS` e Windows XP fa distinzione fra un processo ad alta priorità, che è attualmente selezionato sullo schermo, e processi a bassa priorità che non sono correntemente selezionati. Quando un processo si muove verso una priorità alta, Windows XP aumenta tipicamente il quantum di schedulazione di un qualche fattore, tipicamente 3. (Questo fattore può essere cambiato tramite l'opzione di prestazione nella sezione di sistema del pannello di controllo.) Questo aumento dà al processo in primo piano tre volte il tempo di funzionamento prima che avvenga una pre-emption in un sistema time-sharing.

### 7.3.3 Priorità dei thread

Un thread incomincia con una priorità iniziale determinata dalla sua classe che può essere alterata dalla funzione `SetThreadPriority`. Questa funzione prende un argomento che specifica una priorità relativa alla priorità di base della propria classe:

- `THREAD_PRIORITY_LOWEST`: base - 2
- `THREAD_PRIORITY_BELOW_NORMAL`: base - 1
- `THREAD_PRIORITY_NORMAL`: base + 0
- `THREAD_PRIORITY_ABOVE_NORMAL`: base + 1
- `THREAD_PRIORITY_HIGHEST`: base + 2

Si usano altre due indicazioni per aggiustare la priorità. Si ricordi dal Paragrafo 3.2.1 di questo capitolo che il kernel ha due classi di priorità: 16 -31 per la classe in tempo reale e 0 -15 per la classe a priorità variabile. `THREAD_PRIORITY_IDLE` pone la priorità a 16 per i thread in tempo reale ed a 1 per quelli a priorità variabile. `THREAD_PRIORITY_TIME_CRITICAL` pone la priorità a 31 per i thread in tempo reale ed a 15 per quelli a priorità variabile.

Come abbiamo discusso nel Paragrafo 3.2.1 di questo capitolo, il kernel aggiusta dinamicamente la priorità di un thread a seconda che sia collegato all'I/O o alla CPU. Le API di Win32 forniscono un metodo per disabilitare questa regolazione, tramite le funzioni `SetProcessPriorityBoost` e `SetThreadPriorityBoost`.

### 7.3.4 Sincronizzazione dei thread

Un thread può essere creato in **stato sospeso** (suspended thread): quel thread non va in esecuzione finché un altro thread non lo rende eleggibile tramite la funzione `ResumeThread`; la funzione

`SuspendThread` opera in modo opposto. Queste funzioni regolano un contatore, in modo che se un thread è sospeso per due volte, deve essere ripreso per due volte prima di poter funzionare. Per sincronizzare l'accesso concorrente agli oggetti condivisi dai thread, il kernel fornisce oggetti di sincronizzazione, quali semafori e mutex.

Inoltre, la sincronizzazione dei thread può essere ottenuta usando le funzioni `WaitForSingleObject` o `WaitForMultipleObjects`. Un altro metodo di sincronizzazione nelle API di Win32 è la sezione critica. Una sezione critica è una regione sincronizzata del codice che può essere eseguita solo da un thread alla volta. Un thread stabilisce una sezione critica chiamando `InitializeCriticalSection`. L'applicazione deve chiamare `EnterCriticalSection` prima di entrare nella sezione critica e `LeaveCriticalSection` dopo l'uscita dalla sezione critica. Queste due procedure garantiscono che, se thread multipli tentano di entrare in modo concorrente nella sezione critica, è consentita la prosecuzione solo ad un thread alla volta, e gli altri aspettano la procedura `EnterCriticalSection`. Il meccanismo della sezione critica è più veloce dell'uso di oggetti di sincronizzazione del kernel, poiché evita l'allocazione degli oggetti del kernel finché non incontra per la prima volta una disputa per la sezione critica.

### 7.3.5 Fibre

Una **fibra** (fiber) è un codice in modalità utente che viene schedato secondo un algoritmo di schedazione definito dall'utente. Un processo può avere al suo interno fibre multiple, proprio come può avere thread multipli. La principale differenza fra i thread e le fibre è che i thread possono essere eseguiti in modo concorrente, ma è consentita l'esecuzione di solo una fibra alla volta, anche nell'hardware multiprocessore. Questo meccanismo è incluso in Windows XP per facilitare la portabilità delle applicazioni ereditate da UNIX e che sono state scritte per un modello di esecuzione della fibra.

Il sistema crea una fibra chiamando o `ConvertThreadToFiber` o `CreateFiber`; la differenza principale fra queste funzioni è che `CreateFiber` non inizia l'esecuzione della fibra creata. Per iniziare l'esecuzione, l'applicazione deve chiamare `SwitchToFiber`. L'applicazione può terminare una fibra chiamando `DeleteFiber`.

Creazioni/cancellazioni ripetute di thread possono essere costose per le applicazioni e per i servizi che svolgono singolarmente piccole quantità di lavoro. Il gruppo di thread fornisce programmi in modalità utente mediante tre servizi: una coda in cui le richieste di lavoro possono essere sottoposte (tramite l'API: `QueueUserWorkItem`), una API (`RegisterWaitForSingleObject`) che può essere usata per collegare chiamate ripetute agli handle in attesa, e una API per collegare le chiamate ripetute alle interruzioni (`CreateTimerQueue` e `CreateTimerQueueTimer`).

L'obiettivo del pool di thread è di incrementare le prestazioni; i thread sono relativamente costosi e un processore può eseguire solo una cosa alla volta indipendentemente da quanti thread si usano. Il pool di thread cerca di ridurre il numero di thread in un processo, ritardando un poco le richieste di lavoro (riutilizzando ogni thread per più richieste), mentre fornisce abbastanza thread per utilizzare efficacemente la CPU della macchina.

Le API di attesa e i richiami del temporizzatore permettono al pool di thread di ridurre il numero di thread in un processo, usandone molto meno di quanti non sarebbero necessari se un processo dovesse dedicare un thread per servire ogni handle o interruzione.

## **7.4 Comunicazione tra processi**

Le applicazioni unidirezionali di Win32 gestiscono la comunicazione tra processi condividendo gli oggetti del kernel. Un altro modo consiste nel passare i messaggi: un metodo che è particolarmente popolare nelle applicazioni della GUI di Windows. Un thread può inviare un messaggio ad un altro thread o ad una finestra chiamando `PostMessage`, `PostThreadMessage`, `SendMessage`, `SendThreadMessage`, o `SendMessageCallback`. La differenza fra *spedire per posta* e *inviare* un messaggio consiste nel fatto che le procedure di invio per posta sono asincrone: ritornano immediatamente e il thread chiamante non sa quando il messaggio è stato realmente consegnato. Le procedure di invio sono sincrone: bloccano il chiamante fino a quando il messaggio non è stato consegnato ed elaborato.

In aggiunta all'invio di un messaggio, un thread può pure inviare dei dati insieme al messaggio. Siccome i processi hanno gli spazi degli indirizzi separati, i dati devono essere copiati. Il sistema li copia chiamando `SendMessage` per inviare un messaggio di tipo `WM_COPYDATA` con una struttura dati `COPYDATASTRUCT` che contiene la lunghezza e l'indirizzo dei dati da trasferire. Quando il messaggio è inviato, Windows XP copia i dati in un nuovo blocco di memoria e assegna l'indirizzo virtuale del nuovo blocco al processo ricevente.

### **7.4.1 Code di input individuali**

A differenza dell'ambiente di Windows a 16 bit, ogni thread Win32 ha una propria coda di input da cui il thread riceve i messaggi. (ogni input è ricevuto tramite messaggi.) Questa struttura è più affidabile della coda di input condivisa dalle finestre a 16 bit, poiché, con code separate, un'applicazione bloccata non può bloccare l'input di altre applicazioni. Se un'applicazione Win32 non chiama `GetMessage` per gestire gli eventi nella propria coda di input, la coda si riempie, e dopo circa 5 secondi il sistema segna l'applicazione come "non rispondente".

## **7.5 Gestione della memoria**

Le API di Win32 forniscono ad un'applicazione parecchie possibilità di usare la memoria: memoria virtuale, file a memoria mappata, heap, e memorizzazione locale ai thread.

### **7.5.1 Memoria virtuale**

Un'applicazione chiama `VirtualAlloc` per riservare o impegnare la memoria virtuale e `VirtualFree` per rilasciare o liberare la memoria. Queste funzioni permettono all'applicazione di specificare l'indirizzo virtuale in cui la memoria è allocata e operano su multipli del formato della dimensione della pagina di memoria, e l'indirizzo di partenza di una regione allocata deve essere più grande di `0x10000`. Esempi di queste funzioni sono mostrati in Figura 14.

```

...
//allocare 16 MB in cima al nostro spazio di indirizzo
void*buf = VirtualAlloc (0,0x1000000, MEM_RESERVE/ MEM_TOP_DOWN,
    PAGE_READWRITE);
//impegnare gli 8 MB superiori dello spazio allocato
VirtualAlloc(buf + 0x800000,0x800000, MEM_COMMIT,PAGE_READWRITE);
//compie un certo lavoro con essa
//rilascia (la memoria)
VirtualFree(buf + 0x800000,0x800000, MEM_DECOMMIT);
//rilascia tutto lo spazio di indirizzo allocato
VirtualFree(buf,0, MEM_RELEASE);
...

```

**Figura 14.** Frammenti di codice per l'allocazione della memoria virtuale.

Un processo può bloccare alcune delle pagine impegnate nella memoria fisica chiamando `VirtualLock`. Il numero massimo di pagine che un processo può bloccare è 30, a meno che non venga prima chiamato `SetProcessWorkingSetSize` per aumentare la dimensione massima del working-set.

### **7.5.2 File a memoria mappata**

Un altro modo in cui un'applicazione usa la memoria è tramite la mappatura della memoria di un file nel proprio spazio di indirizzo. La mappatura della memoria è anche un modo conveniente di condivisione della memoria fra due processi: entrambi i processi mappano lo stesso file nella propria memoria virtuale. La mappatura della memoria è un processo a più stadi, come si può vedere nell'esempio di Figura 15.

Se un processo desidera mappare un qualche spazio di indirizzo per condividere una regione di memoria con un altro processo, non c'è bisogno di alcun file. Il processo chiama `CreateFileMapping` con una handle del file di `0xffffffff` e una dimensione particolare. L'oggetto risultante del file mappato può essere condiviso per eredità, per ricerca del nome, o per duplicazione.

```

...
//aprire il file o crearlo se non esiste
HANDLE hfile = CreateFile ("somefile", GENERIC_READ /
    GENERIC_WRITE,
    FILE_SHARE_READ / FILE_SHARE_WRITE, NULL, OPEN_ALWAYS,
    FILE_ATTRIBUTE_NORMAL,
    NULL);
//creare un file che mappa uno spazio di 8 MB
HANDLE hfile = CreateFileMapping(hfile,PAGE_READWRITE,

```

```

    SEC_COMMIT,0,0x800000, "SHM_1");
//dare uno sguardo allo spazio mappato
Void* buf = MapViewOfFile (hmap, FILE_MAP_ALL_ACCESS, 0, 0, 0
times800000);
//eseguire qualcosa con esso
//demappare il file
UnmapViewOfFile (buf);
CloseHandle(hfile);...

```

**Figura 15.** Frammenti di codice per la mappatura della memoria di un file.

### 7.5.3 Heap

Il terzo modo per le applicazioni di usare la memoria è un heap che, nell'ambiente Win32, è una regione dello spazio di indirizzo riservata. Quando un processo di Win32 viene inizializzato, esso è creato per default con un heap di 1 MB. Poiché molte funzioni di Win32 usano lo heap di default, l'accesso è sincronizzato per proteggere le strutture dati dello spazio di allocazione dello heap, al fine di evitare danneggiamenti da parte di aggiornamenti concorrenti eseguiti da thread multipli.

Win32 fornisce parecchie funzioni di gestione dello heap in modo che un processo possa allocare e gestire un heap privato. Queste funzioni sono: `HeapCreate`, `HeapAlloc`, `HeapRealloc`, `HeapSize`, `HeapFree` e `HeapDestroy`. La API di Win32 fornisce pure le funzioni di `HeapLock` e `HeapUnlock` per permettere a un thread di ottenere accesso esclusivo ad un heap. A differenza di `VirtualLock`, queste funzioni permettono solo la sincronizzazione; non bloccano le pagine nella memoria fisica.

### 7.5.4 Memorizzazione locale di un thread

Il quarto modo di usare la memoria da parte delle applicazioni è un meccanismo di memorizzazione locale di un thread. Le funzioni che si basano su dati globali o statici non funzionano correttamente in un ambiente a più thread. Per esempio, la funzione run-time `strtok` del C usa una variabile statica per tenere traccia della propria posizione corrente mentre analizza una stringa. Affinchè due thread concorrenti eseguano correttamente `strtok`, hanno bisogno di variabili separate della *posizione corrente*. Il meccanismo di memorizzazione locale ai thread assegna memoria globale sulla base del singolo thread. Fornisce sia metodi dinamici che statici per creare memoria per il thread locale. Il metodo dinamico è mostrato in Figura 16.

```

//riserva uno spazio per una variabile
DWORD var_index = TlsAlloc();
//la pone ad un certo valore
TlsSetValue(var_index,10);
//gli ritorna il valore
int var = TlsGetValue(var_index);

```

```
//rilascia l'indice  
TlsFree(var_index);
```

**Figura 16.** Codice per la memorizzazione dinamica di un thread locale.

Per usare una variabile statica del thread locale, l'applicazione dichiara la variabile nel modo seguente, al fine di assicurarsi che ogni thread abbia la propria copia privata:

```
__declspec(thread) DWORD cur_pos = 0;
```

## 8 Sommario

Microsoft ha progettato Windows XP in modo che sia un sistema operativo estensibile e portabile, capace di trarre vantaggio dalle nuove tecniche e dall'hardware. Windows XP supporta una molteplicità di ambienti operativi e la multielaborazione simmetrica, sia per i processori a 32 bit che a 64 bit e i computer NUMA. L'uso di oggetti del kernel per i servizi di base ed il supporto per elaborazione client-server, permette a Windows XP di supportare un'ampia varietà di ambienti applicativi. Per esempio, Windows XP può far funzionare programmi compilati per MS-DOS, Win16, Windows 95, Windows XP e/o POSIX. Fornisce memoria virtuale, uso della cache integrato e schedulazione con rilascio anticipato. Windows XP supporta un modello di sicurezza più robusto di quello dei sistemi operativi precedenti di Microsoft ed include caratteristiche di internazionalizzazione. Windows XP funziona su un'ampia varietà di computer, in modo che gli utenti possano scegliere ed aggiornare l'hardware per accoppiare le disponibilità economiche con i requisiti di prestazioni senza avere bisogno di cambiare le applicazioni utilizzate.

## Esercizi

- 1 Quale tipo di sistema operativo è Windows XP ? Descrivere due caratteristiche importanti.
- 2 Elencare gli obiettivi progettuali di Windows XP. Descriverne due in dettaglio.
- 3 Descrivere il processo di avvio di Windows XP.
- 4 Descrivere i tre strati principali dell'architettura di Windows XP.
- 5 Quale è il job del gestore dell'oggetto?
- 6 Che cosa è un handle e come fa un processo ad ottenere un handle?
- 7 Descrivere lo schema di gestione della memoria virtuale. Come fa il gestore di VM a migliorare le prestazioni?
- 8 Quali tipi di servizi fornisce il gestore del processo? Che cosa è una procedura di chiamata locale?

- 9 Quali sono le responsabilità del gestore di I/O?
- 10 Che cosa gestisce la cache in Windows XP? Come è gestita la cache?
- 11 Windows XP fornisce processi in modalità utente che gli permettono di far funzionare programmi sviluppati per altri sistemi operativi? Descrivere due di questi sottosistemi.
- 12 Quali tipi di reti supporta Windows XP? Come fa Windows XP a realizzare i protocolli di trasporto? Descrivere due protocolli di rete.
- 13 Come è organizzato lo spazio del nome in NTFS? Descriverlo.
- 14 In quale modo NTFS gestisce le strutture dati? Come fa NTFS a recuperare da un arresto del sistema? Che cosa è sicuro dopo che è avvenuto un recupero?
- 15 Cosa è un processo e come è controllato in Windows XP?
- 16 In quale modo Windows XP assegna la memoria all'utente?
- 17 Descrivere alcuni dei modi in cui un'applicazione può usare la memoria tramite le API di Win32.

## Note Bibliografiche

Solomon e Russinovich [2000] danno una descrizione di Windows XP e parecchi dettagli tecnici sulle strutture interne e i componenti del sistema. Tate [2000] è un buon riferimento sull'uso di Windows XP. Microsoft Windows XP Server Resource Kit (Microsoft [2000b]) è un gruppo di sei volumi utili per l'uso e le spiegazioni di Windows XP. Microsoft Developer Network Library (Microsoft [2000a]) è pubblicato trimestralmente; fornisce ampie informazioni su Windows XP e su altri prodotti Microsoft.

Iseminger [2000] fornisce un buon riferimento sul direttorio attivo di Windows XP. Richter [1997] presenta una discussione dettagliata sulla scrittura di programmi che usano le API di Win32. Silberschatz et al. [2001] contiene una buona discussione sugli alberi B+.