

# IL SISTEMA OPERATIVO WINDOWS 2000

Microsoft Windows 2000 è un sistema operativo per elaborazione in multitasking preemptive a 32 bit, per Intel Pentium e microprocessori successivi. Il successore del sistema operativo Windows NT, fu inizialmente chiamato Windows NT versione 5.0. Gli obiettivi chiave per il sistema sono: portabilità, sicurezza, conformità con Portable Operating System Interface (POSIX o IEEE Std. 1003.1), supporto multiprocessore, estensibilità, supporto internazionale e compatibilità con MS-DOS e applicazioni di Microsoft Windows. In questo capitolo, discuteremo gli obiettivi chiave di questo sistema, l'architettura stratificata che lo rende di facile impiego, il file system, le reti e l'interfaccia di programmazione.

## 1 Storia

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

Originariamente, il gruppo di progettisti aveva pianificato che NT usasse, come ambiente nativo, le API di OS/2, ma, durante lo sviluppo, Windows NT è stato cambiato per poter usare le API a 32 bit di Windows (o 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 server avanzato. A quel tempo Windows a 16 bit era giunto alla versione 3.1, Windows NT versione 4.0 adottava l'interfaccia utente di Windows 95 e incorporava un web server per Internet e un browser web. Inoltre, le procedure dell'interfaccia utente ed il codice grafico furono spostati nel kernel per migliorare le prestazioni, con l'effetto secondario di diminuire l'affidabilità del sistema. Sebbene le precedenti versioni di NT fossero state portate su altre architetture di microprocessore, Windows 2000 interruppe quella pratica per esigenze di mercato. Pertanto, la **trasferibilità** (portabilità) ora fa riferimento solo ai sistemi dotati di architettura Intel. Windows 2000 usa un'architettura microkernel (come Mach), in modo che i miglioramenti possano essere applicati solo ad una parte del sistema operativo senza influenzare le parti rimanenti; mediante l'aggiunta di servizi terminali, Windows 2000 è ora un sistema operativo multiutente.

Windows 2000 è stato rilasciato nel 2000 ed ha incorporato cambiamenti significativi: inoltre, aggiunge un servizio di direttorio basato su X.500, un migliore supporto di rete, supporto per dispositivi plug-and-play, un nuovo file system con supporto per memorizzazione gerarchica e un file system distribuito, come pure il supporto per più processori e più memoria.

Ci sono quattro versioni del Windows 2000: la versione professionale è intesa per uso su desktop. Le altre tre sono versioni server: server, server avanzato e datacenter server. Esse differiscono principalmente per la quantità di memoria e per il numero di processori. Usano lo stesso kernel e lo stesso codice del sistema operativo, ma le versioni Windows 2000 server e server avanzato sono configurate per applicazioni client server e possono operare come server delle applicazioni su NetWare e su Microsoft LAN. Windows 2000 datacenter server ora supporta fino a 32 processori e fino a 64 GB di RAM.

Nel 1996, sono state vendute più licenze di Windows NT server delle licenze di tutte le versioni UNIX. È interessante notare che il codice di base di Windows 2000 è dell'ordine di 30 milioni di righe. Si paragoni questa dimensione con il codice base di Windows NT versione 4.0 che è di circa 18 milione di righe.

## 2 Principi progettuali

I principi progettuali che Microsoft ha enunciato per Windows 2000 includono: l'estensibilità, la portabilità, l'affidabilità, la compatibilità, le prestazioni ed il supporto internazionale.

L'**estensibilità** (extensibility) indica la capacità di un sistema operativo di mantenersi aggiornato con il progresso della tecnologia in modo che i cambiamenti siano facili al passare del tempo; i progettisti hanno realizzato Windows 2000 mediante un'architettura stratificata. Il codice eseguibile di Windows 2000, che funziona nel kernel o in modo protetto, fornisce i servizi di base del sistema. Sopra al codice eseguibile, operano parecchi sottosistemi server che funzionano in modalità utente, tra essi vi sono i **sottosistemi di ambiente** (environmental subsystems) che emulano differenti sistemi operativi; pertanto, programmi scritti per MS-DOS, Microsoft Windows e POSIX, possono funzionare in Windows 2000 nell'ambiente appropriato. (Consultare il Paragrafo 4 per maggiori informazioni sui sottosistemi di ambiente.)

A causa della struttura modulare, si possono aggiungere ulteriori sottosistemi di ambiente senza modificare il codice eseguibile; inoltre Windows 2000 usa driver caricabili nel sistema di I/O, in modo da poter aggiungere nuovi file system, nuovi dispositivi di I/O e nuovi tipi di rete mentre il sistema è in funzione. Windows 2000, analogamente al sistema operativo Mach, usa un modello client server, e supporta l'elaborazione distribuita mediante chiamate di procedura remota (RPC), definite da Open Software Foundation.

Un sistema operativo è **portabile** (portable) se può essere mosso da un'architettura hardware ad un'altra con un piccolo numero di modifiche. Windows 2000 è progettato per essere portabile. Come del resto avviene nel sistema operativo UNIX, la maggior parte del sistema è scritta in C e C++. Tutto il codice, dipendente dal processore, è isolato in una libreria di collegamento dinamico (DLL), chiamata **strato di astrazione hardware** (hardware-abstraction layer: HAL). Una DLL è un file che viene mappato nello spazio di indirizzamento dei processi in modo che qualsiasi funzione nella DLL sembri far parte del processo. Gli strati superiori di Windows 2000 dipendono da HAL, invece che dall'hardware sottostante, e ciò aiuta Windows 2000 ad essere portabile. HAL manipola l'hardware direttamente, isolando il resto di Windows 2000 dalle differenze hardware tra le piattaforme in cui funziona.

L'**affidabilità** (reliability) è la capacità di gestire condizioni di errore, compresa la capacità del sistema operativo di proteggere se stesso ed i propri utenti da software difettoso o malizioso. Windows 2000 resiste ai difetti ed agli attacchi usando la protezione hardware per la memoria virtuale e meccanismi di protezione software per le risorse del sistema operativo. Inoltre, Windows 2000, viene consegnato con un file system nativo: il **file system NTFS**, che permette di recuperare automaticamente, dopo un arresto del sistema, da molti tipi di errori del file system. Windows NT versione 4.0 ha ricevuto dal governo degli Stati Uniti una classificazione di sicurezza C-2, che indica un moderato livello di protezione contro software difettoso e contro attacchi maliziosi. Windows 2000 è attualmente sotto valutazione del governo per un'analogia classificazione. Per maggiori informazioni sulle classificazioni di sicurezza, consultare il Paragrafo 19.8.

Windows 2000 fornisce **compatibilità** (compatibility) a livello sorgente con le applicazioni che seguono lo standard IEEE 1003.1 (POSIX) e, di conseguenza, possono essere compilate per funzionare su Windows 2000 senza apportare cambiamenti al codice sorgente. Inoltre, Windows 2000 può eseguire codici binari per molti programmi compilati per architetture Intel X86 che utilizzano MS-DOS, Windows a 16 bit, OS/2, LAN Manager e Windows a 32 bit, usando i sottosistemi di ambiente accennati prima. Questi sottosistemi di ambiente supportano una certa varietà di file system, incluso il file system FAT di MS-DOS, il file system HPFS di OS/2, il file system per CD: ISO9660 e NTFS. La compatibilità di Windows 2000 con i file binari, tuttavia, non è perfetta; per esempio, in MS-DOS, le applicazioni possono accedere direttamente alle porte hardware, ma, ai fini di affidabilità e sicurezza, Windows 2000 proibisce tale accesso.

Windows 2000 è progettato per fornire buone **prestazioni** (performance). I sottosistemi che costituiscono Windows 2000 possono comunicare tra loro in modo efficace tramite la chiamata di

procedura locale (LPC) che fornisce il passaggio di messaggi con alte prestazioni. Ad eccezione del kernel, i thread nei sottosistemi di Windows 2000 possono essere prerilasciati (preempted) dai thread ad alta priorità e, quindi, il sistema può rispondere rapidamente ad eventi esterni. Inoltre, Windows 2000 è progettato per la multielaborazione simmetrica: in un computer di tipo multiprocessore, parecchi thread possono funzionare contemporaneamente. La scalabilità corrente di Windows 2000 è limitata, in confronto a quella di UNIX. A partire dalla fine del 2000, Windows 2000 ha supportato sistemi con al massimo 32 CPU, mentre Solaris arrivava a 64 processori. Le versioni precedenti del NT supportavano fino ad 8 processori.

Windows 2000 è pure progettato per un **uso internazionale**; fornisce supporto per differenti lingue locali tramite le API di **supporto della lingua nazionale** (national language support: NLS) che forniscono procedure specializzate per il formato della data, dell'ora e della moneta in accordo con le abitudini nazionali. I confronti di stringhe sono particolarizzati per tenere conto di insiemi di caratteri variabili. UNICODE è il codice nativo dei caratteri di Windows 2000 che supporta anche i caratteri ANSI: li converte in caratteri UNICODE prima di manipolarli (conversione da 8 bit a 16 bit).

### 3 Componenti di sistema

L'architettura di Windows 2000 è un sistema stratificato in moduli, come mostrato in Figura 1. Gli strati principali sono HAL, il kernel e il codice eseguibile, che funzionano tutti in modo protetto, e un'ampia collezione di sottosistemi che funzionano in modalità utente. I sottosistemi in modalità utente sono divisi in due categorie: i sottosistemi di ambiente, che emulano differenti sistemi operativi; 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 possono essere mantenute semplici. Il resto di questo paragrafo descriverà tali strati e sottosistemi.

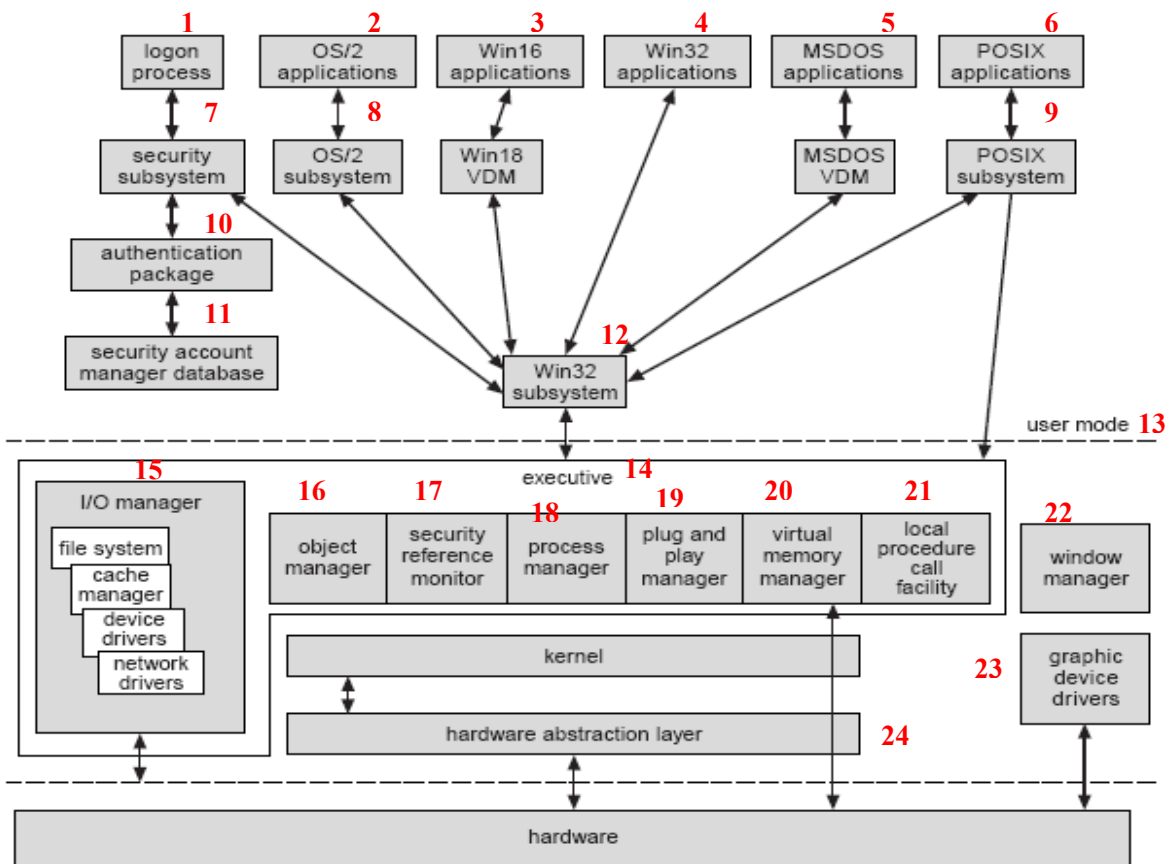
#### 3.1 Strato di astrazione hardware

HAL è lo strato software che nasconde le differenze hardware dai livelli superiori del sistema operativo, in modo da rendere Windows 2000 portabile. HAL esporta un'interfaccia di macchina virtuale che è usata dal kernel, dal codice eseguibile e dai driver. Un vantaggio di questo metodo è che è necessaria solo una singola versione di ogni driver: può funzionare su tutte le piattaforme hardware senza dover trasferire il codice del driver. HAL inoltre fornisce supporto per la multielaborazione simmetrica. Per questioni di prestazioni, i driver I/O (e driver grafici in Windows 2000) possono accedere direttamente all'hardware.

#### 3.2 Kernel

Il kernel di Windows 2000 fornisce le fondamenta al codice eseguibile e ai sottosistemi. Il kernel non è mai paginato fuori dalla memoria e l'esecuzione non avviene mai con prerilascio (preempted). È responsabile di quattro attività principali: schedulazione del thread, gestione di interrupt ed eccezioni, sincronizzazione a basso livello del processore e recupero dopo un guasto all'alimentazione.

Il kernel è orientato agli oggetti. Un **tipo oggetto** (object type) in Windows 2000 è un tipo di dati definito dal sistema e possiede un insieme di attributi (o valori dei dati) e un insieme di metodi (cioè funzioni od operazioni); un **oggetto** (object) è proprio un'istanza di un particolare tipo oggetto. Il kernel esegue il proprio job mediante l'uso di un insieme di oggetti del kernel i cui attributi memorizzano i dati del kernel e i cui metodi eseguono le attività del kernel.



**Figura 1.** Diagramma a blocchi di Windows 2000.

1. Processo di logon
2. Applicazioni OS/2
3. Applicazioni Win16
4. Applicazioni Win32
5. Applicazioni MSDOS
6. Applicazioni POSIX
7. Sottosistema di sicurezza
8. Sottosistema OS/2
9. Sottosistema POSIX
10. Pacchetto di autenticazione
11. Database del gestore degli account di sicurezza
12. Sottosistema Win32
13. Modo utente
14. Esecutivo
15. Gestore I/O - file system - gestore della cache - driver di periferiche - driver di rete
16. Gestore degli oggetti
17. Controllo del riferimento di sicurezza

18. Gestore dei processi
19. Gestore plug and play
20. Gestore della memoria virtuale
21. Software per la chiamata di procedure locali
22. Gestore della finestra
23. Driver di periferiche grafiche
24. Strato di astrazione dell'Hardware.

Il kernel usa due tipi di oggetti: il primo tipo comprende gli **oggetti del dispatcher** (dispatcher objects), i quali controllano l'invio e la sincronizzazione nel sistema; esempi di questi oggetti sono: gli eventi, i mutanti, i mutex, i thread ed i timer. L'**oggetto evento** (event object) è usato per registrare il verificarsi di un evento e per sincronizzarlo con una qualche azione. Il **mutante** (mutant) fornisce la mutua esclusione in modalità utente o in modalità kernel con la nozione di proprietà. Il **mutex**, che è disponibile solo in modalità kernel, fornisce mutua esclusione libera da stalli. Un **oggetto semaforo** (semaphore object) agisce come un contatore o gate per controllare il numero di thread che accedono ad una qualche risorsa. L'**oggetto thread** (thread object) è l'entità che è fatta funzionare dal kernel ed è associata con un **oggetto processo** (process object). Gli **oggetti timer** (timer objects) si usano per tenere traccia del tempo e per segnalare la perdita di sincronismo quando le operazioni impiegano troppo tempo e hanno bisogno di essere interrotte.

Il secondo tipo di oggetti del kernel contiene gli **oggetti di controllo** (control object), che includono: oggetti per chiamate di procedura asincrone, per interrupt, per le notifica dell'alimentazione, per lo stato dell'alimentazione, per i processi, e per i profili. Il sistema usa una chiamata di procedura asincrona (APC) per entrare in un thread in esecuzione e chiamare una procedura. L'**oggetto interrupt** (interrupt object) collega una routine di interrupt di servizio con una sorgente di interrupt. Il sistema usa l'**oggetto di notifica dell'alimentazione** (power-notify object) per richiamare automaticamente una procedura specifica dopo un guasto all'alimentazione e l'**oggetto di stato dell'alimentazione** (power-status object) per controllare se è venuta a mancare l'alimentazione. Un oggetto processo rappresenta lo spazio di indirizzo virtuale e controlla le informazioni necessarie per eseguire i thread associati con un processo. Per concludere, il sistema usa l'**oggetto profilo** (profile object) per misurare la quantità di tempo usata da un blocco di codice.

### 3.2.1 Thread e schedulazione

Come fanno molti sistemi operativi moderni, Windows 2000 usa le nozioni di processi e thread per il codice eseguibile. Il processo ha uno spazio di indirizzi nella memoria virtuale, e informazioni quali la priorità base e un'affinità con uno o più processori. Ogni processo ha uno o più thread, che sono le unità di esecuzione eseguite dal kernel. Ogni thread ha un proprio stato, che include una priorità, una affinità del processore e informazioni sull'account.

I sei stati possibili del thread sono: ready, standby, running, waiting, transition e terminated. Lo stato **ready** (pronto) significa in attesa di funzionare. Il thread ready a più alta priorità è spostato verso lo stato **standby** (attesa) che significa che sarà il thread successivo a mettersi in funzione. In un sistema multiprocessore, per ogni processore viene mantenuto un thread nella stato di standby. Un thread è **running** (funzionante) quando è in esecuzione su di un processore e funzionerà finché non verrà preilasciato (preempted) da un thread a priorità più alta, o non terminerà, o fino alla scadenza del proprio quantum di tempo, o fino ad una chiamata bloccante di sistema, come un'operazione di I/O. Un thread è in condizione **waiting** (attesa) quando aspetta un segnale, quale il completamento di un'operazione di I/O. Un nuovo thread è in condizione **transition** (transizione) mentre aspetta le risorse necessarie per l'esecuzione; un thread è nella condizione **terminated** (terminata) 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: la classe variabile, che contiene i thread che hanno priorità compresa tra 0 e 15; la classe real-time, che contiene i thread con priorità comprese tra 16 e 31. Il dispatcher usa una coda per ogni priorità schedata, e attraversa le code dalla più alta alla più bassa finché non trova un thread pronto a funzionare. Se un thread ha un'affinità particolare di processore, ma quel processore non è disponibile, il dispatcher passerà oltre e continuerà a cercare un thread pronto a funzionare; se non ne trova nessuno pronto, il dispatcher eseguirà un thread speciale, detto idle thread.

Quando il quantum di tempo del thread si esaurisce, questi viene interrotto; se il thread è nella classe a priorità variabile, la priorità viene abbassata, tuttavia senza mai scendere al di sotto di quella base. L'abbassamento di priorità del thread serve a limitare il consumo della CPU dei thread computing-bound. Quando un thread a priorità variabile viene rilasciato da un'operazione in attesa, il dispatcher ne aumenta la priorità in funzione di ciò che il thread stava aspettando; per esempio, un thread in attesa di un'operazione di I/O della tastiera otterrebbe un grande incremento di priorità, mentre un thread in attesa di un'operazione su disco ne otterrebbe uno moderato. Questa strategia tende a dare buoni tempi di risposta a thread interattivi che usano il mouse e le finestre e permette ai thread collegati ad operazioni di I/O di tenere i dispositivi di I/O occupati, mentre permette a thread collegati all'elaborazione di utilizzare cicli sparsi di CPU in background. Questa strategia è usata da parecchi sistemi operativi in timesharing, compreso UNIX; inoltre, la finestra corrente con cui l'utente interagisce riceve pure un aumento di priorità per migliorare il proprio tempo di risposta.

La schedulazione può presentarsi quando un thread entra in stato di pronto, di attesa o termina, o quando un'applicazione cambia la priorità del thread o l'affinità del processore. Se un thread real-time, a priorità più alta, diventa pronto mentre un thread a priorità più bassa è in funzione, quello a priorità più bassa verrà prerilasciato. Il prerilascio fornisce ad un thread real-time accesso preferenziale alla CPU quando il thread ne ha bisogno. Windows 2000 non è, tuttavia, un sistema operativo hard real-time, perché non garantisce che un thread real-time inizi l'esecuzione entro un particolare limite di tempo.

### 3.2.2 Eccezioni e interrupt

Il kernel fornisce anche la gestione di trap per le eccezioni e gli interrupt, generati dall'hardware o dal software. In Windows 2000 sono definite parecchie eccezioni indipendenti dall'architettura, comprese le violazioni di accesso alla memoria, overflow di un intero, overflow o underflow in virgola mobile o divisione per zero di un numero intero, divisione per zero di un numero in virgola mobile, un'istruzione illegale, il disallineamento dei dati, istruzioni privilegiate, errori di lettura della pagina, violazione della pagina di guardia (guard-page), superamento della quota dei file paginati, punto di arresto del debugger, e debugging a passo singolo.

Il gestore di trap può gestire semplici eccezioni; le altre sono gestite dal dispatcher delle eccezioni del kernel. Il **dispatcher delle eccezioni** (exception dispatcher) crea un record di eccezione che contiene il motivo dell'eccezione e trova un gestore che possa occuparsene.

Quando si presenta un'eccezione in modalità kernel, il dispatcher dell'eccezione chiama semplicemente una procedura per localizzare il gestore dell'eccezione; se non lo trova, avviene un errore fatale di sistema e l'utente viene lasciato con il malfamato "schermo blu di morte" che significa un'avaria del sistema.

La gestione delle eccezioni è più complessa per i processi in modalità utente, poiché un sottosistema di ambiente (quale POSIX) può installare una porta del debugger e una porta di eccezione per ogni processo che si crea. Se la porta del debugger è registrata, il gestore dell'eccezione invia l'eccezione a quella porta. Se non trova una porta del debugger o non è in grado di gestire quell'eccezione, il dispatcher tenta di trovare un gestore adatto. Se non trova un handler, viene di nuovo richiamato il debugger in modo da potere intercettare l'errore per eliminarlo. Se un debugger non è in funzione, viene inviato un messaggio alla porta dell'eccezione dei processi per fornire al sottosistema di ambiente una possibilità di tradurre l'eccezione. Per esempio, l'ambiente

POSIX traduce i messaggi di eccezione di Windows 2000 in segnali POSIX prima di trasmetterli al thread che ha causato l'eccezione. Per concludere, se niente altro funziona, 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 (quale un device driver) o una procedura interna del kernel. L'interrupt è rappresentato da un oggetto interrupt che contiene tutte le informazioni necessarie per gestirlo. L'uso di un oggetto interrupt rende facile associare le routine di servizio dell'interrupt con un interrupt stesso senza dovere accedere direttamente all'hardware dell'interrupt.

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, assegnando loro priorità e servendoli nell'ordine di priorità. In Windows 2000 vi sono 32 livelli di interrupt (IRQL): otto sono riservati ad uso del kernel; gli altri 24 rappresentano interrupt hardware tramite HAL, sebbene la maggior parte dei sistemi x86 usi solo 16 linee. Gli interrupt di Windows 2000 sono visualizzati in Figura 2.

livelli di interrupt	tipi di interrupt
31	controllo della macchina o errore di bus
30	avaria dell'alimentazione
29	notifica interprocessore (richiede un altro processore per agire; cioè, spedizione di un processo o aggiornamento della TLB)
28	orologio (usato per tenere traccia del tempo)
27	Profilo
3-26	interrupt del tradizionale IRQ hardware del PC
2	invio e chiamata della procedura rinviata (DPC) (kernel)
1	chiamata di procedura asincrona (APC)
0	Passivo

**Figura 2.** Livelli di richiesta di interrupt di Windows 2000.

Il kernel usa una **tabella di invio dell'interrupt** (interrupt dispatch table) per collegare ogni livello di interrupt con una routine di servizio. In un computer multiprocessore, Windows 2000 mantiene, per ogni processore, una tabella separata di invio dell'interrupt, e ogni IRQL del processore può essere posizionato in modo indipendente per mascherare gli interrupt. Tutti gli interrupt che hanno un livello minore od uguale all'IRQL di un processore vengono bloccati finché l'IRQL non è abbassato da un thread a livello kernel. Windows 2000 trae vantaggio da questa proprietà per usare gli interrupt al fine di eseguire funzioni di sistema. Per esempio, il kernel usa interrupt software per iniziare l'esecuzione del thread, per gestire i timer, e per supportare operazioni asincrone.

Il kernel usa l'interrupt del dispatcher per controllare il cambio di contesto del thread. Quando il kernel è in funzione, esso eleva il livello IRQL nel processore ad un livello sopra quello del dispatcher. Quando il kernel stabilisce che è richiesta una esecuzione del thread, genera un'interrupt di dispatch, ma questo interrupt è bloccato finché il kernel non finisce la propria attività e abbassa il livello

di IRQL; a quel punto, l'interrupt di dispatch può essere servito e, di conseguenza, il dispatcher sceglie un thread da far funzionare.

Quando il kernel decide che una qualche funzione di sistema deve essere finalmente eseguita, ma non immediatamente, esso accoda un oggetto di **chiamata rinviata di procedura** (deferred procedure call: DPC) che contiene l'indirizzo della funzione da eseguire e genera un'interrupt DPC. Quando l'IRQL del processore cala abbastanza in basso, vengono eseguiti gli oggetti DPC. L'IRQL dell'interrupt DPC è tipicamente più alto di quello dei thread dell'utente, in modo che DPC possa interrompere l'esecuzione dei thread dell'utente. Per evitare problemi, i DPC sono limitati per essere abbastanza semplici: non possono modificare la memoria di un thread; creare, acquisire, o attendere gli oggetti; chiamare servizi di sistema; o generare fault di pagina.

### 3.2.3 Sincronizzazione a basso livello del processore

Il terzo tipo di responsabilità del kernel consiste nel fornire sincronizzazione a basso livello al processore. Il meccanismo APC è simile al meccanismo DPC, ma di uso più generale. Il meccanismo APC permette ai thread di predisporre una chiamata della procedura che avverrà in un qualche tempo futuro. Per esempio, molti servizi di sistema accettano, come parametro, una procedura in modalità utente. Invece di utilizzare una chiamata di sistema sincrona che bloccherà il thread fino al completamento della chiamata, un thread dell'utente può eseguire una chiamata asincrona di sistema e fornire una APC; il thread utente continuerà a funzionare. Quando il servizio di sistema finisce, il thread utente sarà interrotto per far funzionare APC spontaneamente.

Un APC può essere accodato sia ad un thread di sistema che ad un thread utente, sebbene un APC, in modalità utente, verrà eseguito solo se il thread ha dichiarato se stesso come **avvisabile** (alertable). Un APC è più potente di un DPC, in quanto può acquisire ed aspettare gli oggetti, causare fault di pagina e chiamare servizi di sistema. Poiché un APC si esegue nello spazio degli indirizzi del thread bersaglio, il codice eseguibile di Windows 2000 usa in modo esteso APC per elaborare operazioni di I/O.

Poiché Windows 2000 può funzionare su macchine a multiprocessore simmetriche, il kernel deve impedire che due suoi thread modifichino contemporaneamente una struttura dati condivisa. Il kernel usa gli spinlock che risiedono nella memoria globale per ottenere la mutua esclusione del multiprocessore. Siccome ogni attività in un processore si ferma quando un thread tenta di acquisire uno spinlock, un thread che tiene uno spinlock non è pririlasciato, in modo da poter terminare e liberare il blocco il più rapidamente possibile.

### 3.2.4 Recupero dopo un'avaria dell'alimentazione

La quarta ed ultima responsabilità del kernel consiste nel fornire la possibilità di recupero dopo un'avaria all'alimentazione. Un interrupt all'alimentazione, che ha il secondo livello di priorità più elevato, informa il sistema operativo ogni volta che viene rilevata una perdita di potenza. L'oggetto `powernotify` fornisce un modo ad un device driver di registrare una procedura che sarà chiamata al ripristino dell'alimentazione e si assicura che i dispositivi siano riportati ad una condizione adeguata nel momento del recupero. L'oggetto `power-status` è utile per i sistemi dotati di batteria tampone. Prima di iniziare un'operazione critica, un driver esamina l'oggetto `power-status` per stabilire se l'alimentazione è venuta a mancare o meno. Se il driver determina che l'alimentazione non è venuta a mancare, aumenta il livello di IRQL del proprio processore a `powerfail`, esegue l'operazione e azzera il livello di IRQL. Questa sequenza di azioni blocca l'interrupt `powerfail` finché l'operazione critica non si completa.

### 3.3 Il codice eseguibile

Il codice eseguibile di Windows 2000 fornisce un insieme di servizi che tutti i sottosistemi di ambiente possono usare. I servizi sono raggruppati nel modo seguente: gestore dell'oggetto, gestore di memoria virtuale, gestore del processo, capacità di chiamata della procedura locale, gestore di I/O e controllo di riferimento della sicurezza.

#### 3.3.1 Gestore dell'oggetto

Analogamente ad un sistema orientato agli oggetti, Windows 2000 usa gli oggetti per tutti i propri servizi ed entità. Esempi di oggetti sono: gli oggetti direttori, gli oggetti simbolici di collegamento, gli oggetti semaforo, gli oggetti evento, gli oggetti thread, gli oggetti processo, gli oggetti porta e gli oggetti file. Il job del **gestore degli oggetti** (object manager) consiste nel sorvegliare l'uso di tutti gli oggetti. Quando un thread vuole usare un oggetto, chiama il metodo open del gestore degli oggetti per ottenere un handle per l'oggetto. Gli **handle** sono un'interfaccia standardizzata per tutti i generi di oggetti. Similmente ad un handle di un file, un handle di un oggetto è un identificatore unico per un processo che gli conferisce la capacità di accedere e manipolare una risorsa di sistema.

Siccome il gestore degli oggetti è l'unica entità che può generare un handle dell'oggetto, esso è il posto naturale per controllare la sicurezza. Per esempio, il gestore degli oggetti controlla se un processo ha il diritto di accedere ad un oggetto quando il processo cerca di aprire quell'oggetto. Il gestore degli oggetti può anche fare rispettare le quote, come la quantità massima di memoria che un processo può allocare.

Il gestore degli oggetti può tenere traccia di quali processi stanno usando ogni oggetto. Ogni intestazione dell'oggetto contiene un contatore del numero di processi che hanno un handle di quell'oggetto. Quando il contatore si azzerà, l'oggetto viene cancellato dallo spazio dei nomi, se era un nome di un oggetto temporaneo. Poiché Windows 2000, per accedere agli oggetti, usa spesso dei puntatori invece che degli handle, il gestore degli oggetti mantiene egualmente un contatore di riferimento, che incrementa, quando Windows 2000 accede ad un oggetto, e decrementa quando l'oggetto non è più necessario. Quando il contatore di riferimento di un oggetto temporaneo si azzerà, l'oggetto viene cancellato dalla memoria. Gli oggetti permanenti rappresentano entità fisiche, come i drive del disco, e non vengono cancellati quando il contatore di riferimento e quello di open-handle si azzerano.

Gli oggetti sono manipolati da un insieme standard di metodi: `create`, `open`, `close`, `delete`, `query name`, `parse` e `security`. Gli ultimi tre oggetti necessitano di alcune spiegazioni:

- `query name` viene chiamato quando un thread ha un handle in un oggetto, ma desidera conoscere il nome dell'oggetto.
- `parse` viene usato dal gestore dell'oggetto per cercare un oggetto di nome assegnato.
- `security` viene chiamato quando un processo apre o cambia la protezione di un oggetto.

Il codice eseguibile di Windows 2000 permette che sia assegnato a qualsiasi oggetto un **nome** (name). Lo spazio dei nomi è globale, in modo che un processo possa creare un oggetto con nome, e un secondo processo possa poi aprire un handle per l'oggetto e dividerlo con il primo processo. Nell'apertura di un oggetto con nome, da parte di un processo, si può chiedere che la ricerca del nome sia sensibile o non sensibile alle lettere maiuscole.

Un nome può essere permanente o temporaneo; un nome permanente rappresenta un'entità, come un disco, che rimane anche se nessun processo vi accede, un nome temporaneo esiste solo per il tempo in cui un qualche processo tiene un handle di quell'oggetto.

Sebbene lo spazio dei nomi non sia direttamente visibile in rete, il metodo `parse` del gestore degli oggetti viene usato per aiutare ad accedere ad un oggetto con un 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, che a sua volta chiama un reindirizzatore di rete per trovare l'oggetto.

I nomi degli oggetti sono strutturati come i nomi di percorso in MS-DOS e UNIX. I direttori sono rappresentati da un **oggetto direttorio** (directory object) che contiene i nomi di tutti gli oggetti del direttorio. Lo spazio dei nomi di un oggetto può ingrandirsi mediante l'aggiunta dei **domini degli oggetti** (object domains), che sono insiemi autocontenuti di oggetti. Esempi di domini di oggetti sono i floppy disk e i dischi rigidi. È facile vedere come lo spazio dei nomi degli oggetti si estenda quando si aggiunge al sistema un floppy disk: il floppy ha un proprio spazio dei nomi che si innesta sullo spazio dei nomi esistente.

I file system UNIX hanno **collegamenti simbolici** (symbolic links), in modo che soprannomi (nickname) o pseudonimi (alias) possano riferirsi allo stesso file; analogamente, Windows 2000 possiede un **oggetto di collegamento simbolico** (symbolic link object). Un modo in cui Windows 2000 usa collegamenti simbolici consiste nel mappare i nomi dei drive nelle lettere standard di MS-DOS. Le lettere dei drive sono proprio collegamenti simbolici che possono essere rimappati per soddisfare le preferenze dell'utente.

Un processo ottiene un oggetto handle creando un oggetto, aprendo un oggetto esistente, ricevendo un handle duplicato da un altro processo, o ereditando un handle da un **processo padre** (parent process), in modo simile a quello in cui UNIX ottiene un descrittore del file. Questi handle sono tutti memorizzati nella **tabella** degli oggetti (object table) dei processi. Una voce nella tabella dell'oggetto contiene i diritti di accesso all'oggetto e dichiara se l'handle deve essere ereditato dai **processi figlio** (child processes). Quando un processo termina, Windows 2000 ne chiude automaticamente gli handle aperti.

Quando un utente viene autenticato tramite il processo di login, un token di accesso viene attaccato al processo dell'utente. Il token di accesso contiene informazioni quali l'identificatore di sicurezza, gli identificatori del gruppo, i privilegi, il gruppo primario e la lista di default di controllo degli accessi: questi attributi determinano quali servizi ed oggetti possono essere usati da un certo utente.

In Windows 2000, ogni oggetto è protetto da una lista di controllo degli accessi che contiene gli identificatori di sicurezza ed i diritti di accesso assegnati ad ogni processo. Quando un processo tenta di accedere ad un oggetto, il sistema confronta l'identificatore di sicurezza nel token di accesso ai processi con la lista di controllo degli accessi per determinare se può essere consentito l'accesso. Questo controllo avviene solo quando un oggetto è aperto, in modo che i servizi interni di Windows 2000 che usano i puntatori, invece di aprire un handle di un oggetto, escludano il controllo di accesso.

In generale, il creatore dell'oggetto stabilisce, per quell'oggetto, la lista di controllo dell'accesso; se non è data esplicitamente alcuna lista, essa si può ereditare dall'oggetto del creatore, o si può ottenere una lista di default dal token di accesso dell'utente.

Un campo, nel token di accesso, controlla la contabilità dell'oggetto; le operazioni che devono essere contabilizzate vengono annotate nel diario della contabilità del sistema con un'identificazione dell'utente. Il campo della contabilità può guardare in questo diario per scoprire tentativi di penetrazione nel sistema o per accedere ad oggetti protetti.

### 3.3.2 Gestore di memoria virtuale

La porzione di memoria virtuale del codice eseguibile di Windows 2000 è il **gestore di memoria virtuale** (virtual memory manager: VM), il cui progetto presuppone che l'hardware sottostante supporti la mappatura da virtuale a fisica, un meccanismo di paginazione e la coerenza trasparente della cache in sistemi multiprocessore, e che siano permessi ingressi multipli nella tabella di pagina da mappare nello stesso frame di pagina. In Windows 2000, il gestore di VM usa uno schema di gestione con dimensione di pagina da 4 KB. Le pagine di dati che sono assegnate ad un processo ma non si trovano nella memoria fisica sono memorizzate nel **file di paginazione** su disco.

Il gestore di VM usa indirizzi a 32 bit, in modo che ogni processo abbia uno spazio di indirizzi virtuali di 4 GB; i 2 GB superiori sono identici per tutti i processi e sono usati da Windows 2000 in modalità kernel, mentre i 2 GB inferiori sono distinti per ogni processo e sono accessibili sia dai

thread che dal kernel in modalità utente. Si noti che certe configurazioni di Windows 2000 riservano solo 1 GB per il sistema operativo, permettendo ad un processo di usare 3 GB di spazio degli indirizzi.

Il gestore di VM di Windows 2000 usa un processo a due passi per allocare la memoria: il primo passo *riserva* una porzione dello spazio degli indirizzi dei processi; il secondo passo *coinvolge* l'allocazione assegnando spazio nel file di paginazione di Windows 2000. Windows 2000 può limitare la dimensione del file di paginazione, utilizzato da un processo, facendo rispettare una quota nella memoria coinvolta. Un processo può eliminare il coinvolgimento della memoria non più in uso, per liberare la propria quota di paginazione. Siccome la memoria è rappresentata da oggetti, quando un processo (genitore) crea un secondo processo (figlio), il genitore può mantenere la capacità di accesso alla memoria virtuale del figlio: questo è il modo in cui i sottosistemi di ambiente possono gestire la memoria dei propri processi client. Ai fini delle prestazioni, il gestore di VM permette ad un processo privilegiato di bloccare pagine selezionate nella memoria fisica, assicurando quindi che le pagine non vengano scambiate al di fuori del file di paginazione.

Due processi possono condividere la memoria ottenendo handle relativi allo stesso oggetto di memoria, ma questo metodo può essere inefficace poiché l'intero spazio di memoria di un oggetto deve essere reso stabile prima che ciascuno dei processi possa accedere a quell'oggetto. Windows 2000 fornisce un'alternativa, chiamata **oggetto sezione** (section object), per rappresentare un blocco della memoria condivisa. Dopo avere ottenuto un handle di un oggetto sezione, un processo può mappare solo la porzione necessaria di memoria che è chiamata **vista** (view). Il meccanismo della vista permette pure ad un processo di accedere ad un oggetto che è troppo grande da inserire nella quota del file di paginazione dei processi. Il sistema può usare la vista per vagare nello spazio di indirizzamento dell'oggetto, un pezzo per volta.

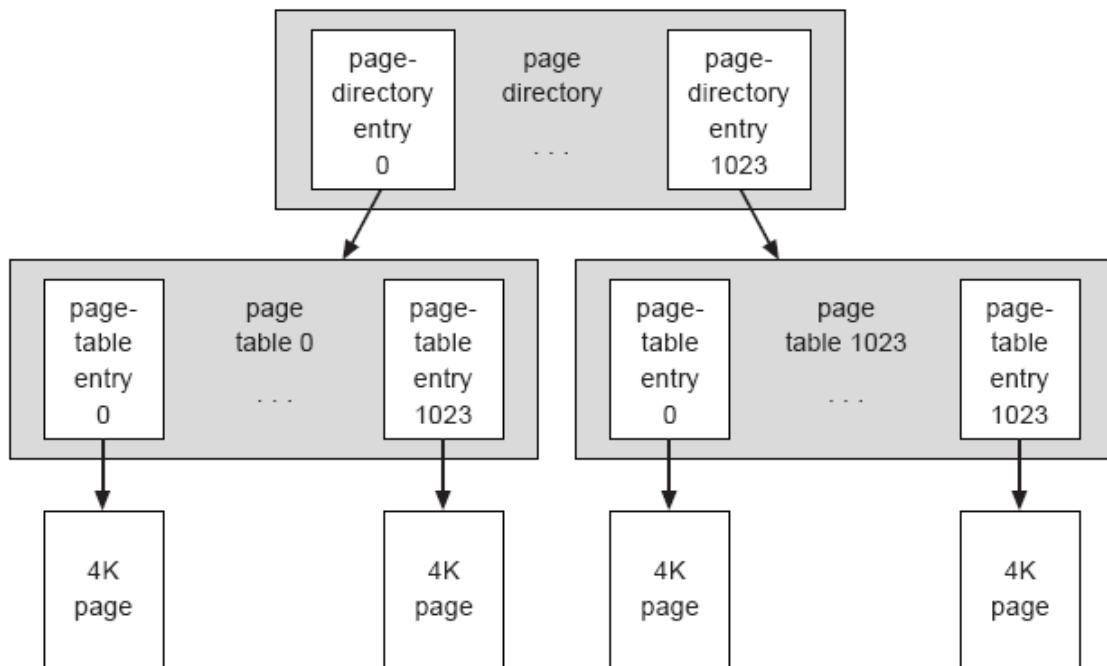
Un processo può controllare in parecchi modi l'uso di un oggetto sezione della memoria condivisa. La dimensione massima di una sezione può essere limitata. La sezione può essere aiutata dallo spazio del disco sia nel file di paginazione di sistema che in un normale file (un **file mappato in memoria**). Una sezione può essere *basata*, intendendo che la sezione appare allo stesso indirizzo virtuale per tutti i processi che vi accedono. Per concludere, la protezione di memoria delle pagine nella sezione può essere posta in sola lettura, lettura-scrittura, sola esecuzione, pagina di guardia, o copy-on-write. Le ultime due voci richiedono una qualche spiegazione:

- Una pagina di guardia solleva un'eccezione se è avvenuto un accesso; l'eccezione può essere usata, per esempio, per controllare se un programma difettoso itera oltre la fine di un array.
- Il meccanismo di copy-on-write permette al gestore di VM di salvare memoria. Quando due processi desiderano copie indipendenti di un oggetto, il gestore di VM mette solo una copia condivisa nella memoria fisica, ma abilita la proprietà di copy-on-write in quella regione di memoria. Se uno dei processi cerca di modificare i dati in una pagina copy-on-write, il gestore di VM esegue dapprima una copia privata della pagina per il processo che deve usarla.

La traduzione di indirizzo virtuale in Windows 2000 usa parecchie strutture dati; ogni processo ha un **direttorio di pagina** (page directory) che contiene 1.024 ingressi della **pagina del direttorio** (page-directory entries) di dimensione di 4 byte. Tipicamente, il direttorio di pagina è privato, ma può essere condiviso fra i processi se l'ambiente lo richiede. Ogni ingresso del direttorio di pagina punta a una **tabella di pagina** (page table) che contiene 1.024 **ingressi della tabella di pagina** (page-table entries: PTE) di dimensione di 4 byte. Ogni PTE punta ad un **frame di pagina** (page frame) di 4 Kb nella memoria fisica. La dimensione totale per un processo di tutte le tabelle di pagina è di 4 MB, in modo che il gestore di VM, in caso di necessità, scambi queste tabelle su disco. Consultare la Figura 3 per un diagramma di questa struttura.

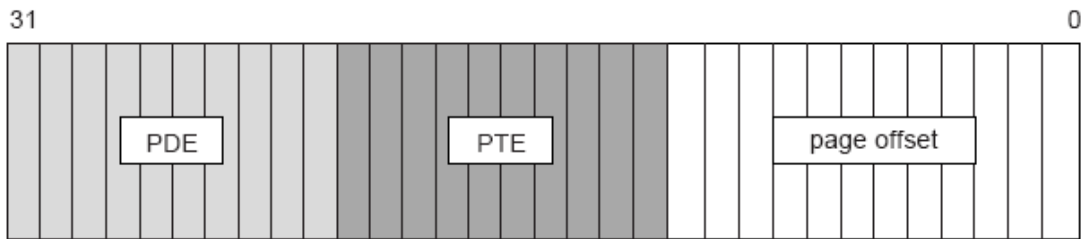
Un numero intero a 10 bit può rappresentare tutti i possibili valori tra 0 e 1.023, quindi un numero intero a 10 bit può selezionare qualsiasi ingresso nel direttorio delle pagine, o in una tabella di pagine. Questa proprietà è usata quando un puntatore all'indirizzo virtuale è tradotto in un indirizzo in byte nella memoria fisica. Un indirizzo di memoria virtuale a 32 bit è suddiviso in tre numeri interi, come mostrato in figura 4. I primi 10 bit dell'indirizzo virtuale sono usati come indici in un direttorio di pagina; questo indirizzo seleziona un ingresso nel direttorio di pagina, che punta ad

una tabella di pagina. L'unità di gestione della memoria (MMU) usa i successivi 10 bit dell'indirizzo virtuale per selezionare un PTE da quella tabella di pagina, il PTE, a sua volta, punta ad un frame di pagina nella memoria fisica. I rimanenti 12 bit dell'indirizzo virtuale puntano ad un byte specifico in quel frame di pagina. MMU crea un puntatore a quel byte specifico in memoria fisica concatenando 20 bit dal PTE, in cui i 12 bit inferiori provengono dall'indirizzo virtuale. Pertanto, PTE a 32 bit possiede 12 bit in sospenso che descrivono la pagina. PTE del Pentium riserva 3 bit ad uso del sistema operativo. I rimanenti bit specificano se la pagina è sporca, se è avvenuto un accesso, se può essere posta in cache, se è a sola scrittura, write through, in modalità kernel, o valida; descrivono quindi, lo stato della pagina in memoria. Per informazioni più generali sugli schemi paginanti, consultare il Paragrafo 9.4.



**Figura 3.** Disposizione della memoria virtuale.

- 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



**Figura 4.** Traduzione dell'indirizzo da virtuale a fisico.

page offset= spiazzamento della pagina

Una pagina può essere in uno dei sei stati: valida, libera, azzerata, in attesa, modificata, o non utilizzabile. Una pagina *valida* è usata da un processo attivo; una pagina *libera* non è referenziata in PTE; una pagina *azzerata* è una pagina libera che è stata riempita di zeri ed è pronta all'uso immediato; una pagina in attesa (*standby*) è stata rimossa dal working set di un processo; una pagina *modificata* è stata scritta, ma non è stata ancora trasferita su disco. Le pagine in attesa e quelle modificate sono considerate pagine di *transizione*. Infine, una pagina *non utilizzabile* non si può usare poiché è stato rilevato un errore hardware.

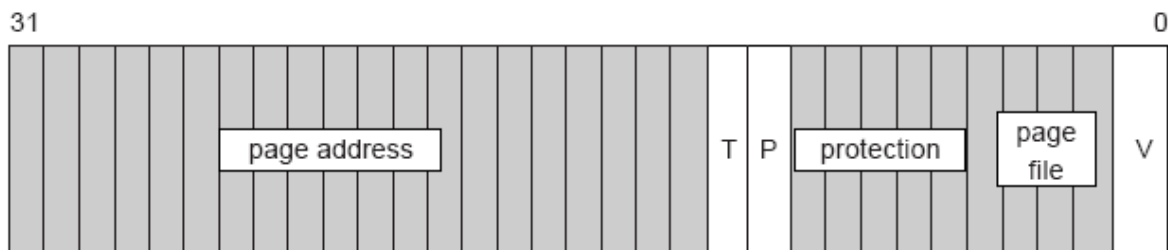
La struttura reale del file di pagina PTE è mostrata in Figura 5. PTE contiene 5 bit per la protezione della pagina, 20 bit per lo spiazzamento del file di pagina, 4 bit per selezionare il file di paginazione e 3 bit che descrivono lo stato della pagina. Il PTE del file di pagina apparirà all'hardware come pagina non valida. Siccome il codice eseguibile e i file a memoria mappata già possiedono una copia su disco, essi non hanno bisogno di spazio in un file di paginazione; se una di queste pagine non è in memoria fisica, la struttura PTE è la seguente: il bit più significativo viene usato per specificare la protezione della pagina, i 28 bit successivi vengono usati per l'indicizzazione in una struttura dati del sistema che indica un file ed uno spiazzamento entro il file per la pagina; i 3 bit di ordine più basso specificano lo stato della pagina.

È difficile condividere una pagina fra processi se ogni processo ha le tabelle del proprio insieme di pagine, dal momento che ogni processo avrà un proprio PTE per il frame di pagina. Quando una pagina condivisa è in stato di fault nella memoria fisica, l'indirizzo fisico dovrà essere memorizzato nel PTE che appartiene ad ogni processo che condivide la pagina. I bit di protezione e i bit di stato della pagina nei PTE avranno bisogno di essere registrati ed aggiornati in modo consistente. Per evitare questi problemi, Windows 2000 usa una indirectione; per ogni pagina condivisa, il processo ha un PTE che punta ad una **voce del prototipo della tabella di pagina** (prototype page-table entry), invece che al frame di pagina. Il prototipo PTE contiene l'indirizzo del frame di pagina ed i bit di protezione e di stato. Pertanto, il primo accesso di un processo ad una pagina condivisa, genera un fault di pagina; dopo il primo accesso, ulteriori accessi sono eseguiti in modalità normale. Se la pagina è contrassegnata a sola lettura, il gestore di VM esegue un copy-on-write e il processo non ha più una pagina condivisa. Le pagine condivise non appaiono mai nel file di pagina, ma si trovano nel file system.

Il gestore della VM tiene traccia di tutte le pagine della memoria fisica in un **database dei frame di pagina** (pageframe database) in cui compare una voce per ogni frame di pagina. I punti di ingresso a PTE puntano al frame di pagina, in modo che il gestore di VM possa mantenere informazioni sullo stato della pagina. I frame di pagina sono collegati per formare, per esempio, la lista delle pagine azzerate e la lista delle pagine libere.

Quando capita un fault di pagina, il gestore di VM segnala un guasto nella pagina mancante, mettendo quella pagina nel primo frame della lista libera, ma non si ferma qui. Ricerche mostrano che il riferimento alla memoria di un thread tende ad avere una proprietà di **località** (località): quando si usa una pagina, è probabile che le pagine adiacenti siano referenziate nell'immediato futuro. (Si pensi all'iterazione di un array, o al prelevamento sequenziale di istruzioni dalla memoria che formano il codice eseguibile per un thread). A causa della località, quando il gestore di VM segnala un fault in una pagina, lo segnala anche per le poche pagine adiacenti; e l'errore nelle pagine adiacenti tende a ridurre il numero totale di fault di pagina. Per maggiori informazioni sulla località, consultare il Paragrafo 10.6.1.

Se non vi sono frame di pagina disponibili nella lista libera, Windows 2000 usa una politica di rimpiazzo FIFO per ciascun processo, per prendere pagine dai processi che ne usano più della dimensione minima del proprio working-set. Windows 2000 controlla i fault di pagina di ogni processo che si trova alla minima dimensione del working-set, e lo aggiusta di conseguenza. In particolare, quando un processo è iniziato in Windows 2000, gli viene assegnato un working-set di default di 30 pagine; Windows 2000 verifica periodicamente la dimensione, rubando una pagina valida al processo. Se il processo continua l'esecuzione senza generare un fault per la pagina rubata, il working-set del processo è ridotto di una unità, e la pagina viene aggiunta alla lista di quelle libere.



**Figura 5.** Ingresso della tabella di pagina del file.

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

### 3.3.3 Gestore di processo

Il gestore di processo di Windows 2000 fornisce servizi per la creazione, la cancellazione e l'uso di thread e processi. Non conosce le relazioni tra genitore e figlio o le gerarchie del processo; queste raffinatezze sono lasciate al particolare sottosistema di ambiente che ha la proprietà del processo.

Un esempio di creazione di un processo nell'ambiente Win32 è il seguente: quando una applicazione di Win32 chiama `CreateProcess`, viene inviato un messaggio al sottosistema Win32, che a sua volta chiama il gestore del processo per creare un processo. Il gestore del processo chiama il gestore dell'oggetto per creare un oggetto processo, e poi restituisce un oggetto handle a Win32 che, di nuovo, chiama ancora il gestore del processo per creare un thread per il processo e alla fine Win32 ritorna gli handle al nuovo processo e al thread.

### 3.3.4 Funzione di chiamata della procedura locale

Il sistema operativo usa la funzione LPC per passare le richieste e i risultati fra i processi client e server in una macchina singola e, in particolare, usa LPC per richiedere servizi dai vari sottosistemi di Windows 2000. Sotto molti aspetti, LPC è simile ai meccanismi RPC che sono usati da molti sistemi operativi per l'elaborazione distribuita attraverso le reti, ma LPC è ottimizzato per l'uso in Windows 2000.

LPC è un meccanismo di passaggio di messaggi. Il processo server pubblica un oggetto globalmente visibile della porta di connessione. Quando un client desidera i servizi di un sottosistema, apre un handle dell'oggetto della porta di connessione del sottosistema e poi manda una richiesta di collegamento a quella porta. Il server crea un canale e restituisce un handle al client. Il canale consiste in una coppia di porte di comunicazione private: una per i messaggi client server e l'altra per i messaggi server client. I canali di comunicazione supportano un meccanismo di richiamata, in modo che il client e il server possano accettare richieste, anche nel caso stiano normalmente aspettando una risposta.

Quando si crea un canale LPC, si deve specificare una delle tre tecniche di passaggio del messaggio.

1. La prima tecnica è adatta a piccoli messaggi (fino a 256 byte). In questo caso, viene usata la coda di messaggi della porta per la memorizzazione intermedia, e i messaggi sono copiati da un processo all'altro.
2. La seconda tecnica è per messaggi di grandi dimensioni. In questo caso, un oggetto della sezione della memoria condivisa è creato come canale. I messaggi inviati attraverso la coda di messaggi della porta contengono un puntatore e informazioni sulla dimensione che si riferiscono all'oggetto della sezione; in tal modo si evita la necessità di copiare grandi messaggi: il mittente mette i dati nella sezione condivisa in cui il ricevente può vederli direttamente.
3. La terza tecnica di passaggio del messaggio LPC, chiamata **LPC rapido** (quick LPC), è usata da porzioni della visualizzazione grafica del sottosistema di Win32. Quando un client chiede una connessione che userà LPC rapido, il server installa tre oggetti: un thread dedicato del server per gestire le richieste, un oggetto della sezione di 64 KB e un oggetto evento-coppia. Un *oggetto evento-coppia* è un oggetto di sincronizzazione che è usato dal sottosistema Win32 per fornire la notifica quando un thread del client ha copiato un messaggio nel server di Win32, o viceversa. I messaggi LPC sono passati nell'oggetto della sezione e la sincronizzazione è effettuata dall'oggetto evento-coppia. LPC presenta parecchi vantaggi: l'oggetto della sezione elimina il dover copiare il messaggio, poiché rappresenta una regione di memoria condivisa. L'oggetto evento-coppia elimina l'overhead di usare l'oggetto-porta per passare messaggi che contengono puntatori e lunghezze. Il thread del server dedicato elimina l'overhead di determinare quale thread del client sta chiamando il server, poiché c'è un thread del server per ogni thread del client. In conclusione, per migliorare le prestazioni, il kernel assegna la preferenza di schedulazione a questi thread dedicati del server. Lo svantaggio è che LPC rapido usa più risorse degli altri due metodi; pertanto il sottosistema Win32 usa LPC rapido solo per le interfacce del gestore di finestre e del dispositivo grafico.

### 3.3.5 Gestore di I/O

**Il gestore di I/O** (I/O manager) è responsabile dei file system, della gestione della cache, dei driver dei dispositivi e dei driver di rete. Tiene traccia di quali file system installabili sono caricati e gestisce i buffer per richieste di I/O. Lavora con il gestore di VM per fornire file I/O a memoria mappata e controlla il gestore della cache di Windows 2000, che maneggia la messa in cache dell'intero sistema di I/O. Il gestore di I/O supporta operazioni sia sincrone che asincrone, fornisce i timeout per i driver e ha meccanismi che permettono ad un driver di chiamarne un altro.

Il gestore di I/O converte le richieste ricevute in una forma standard chiamata **pacchetto di richiesta di I/O** (I/O request packet: IRP) e poi inoltra, per l'elaborazione IRP, al driver corretto. Quando l'operazione finisce, il gestore di I/O riceve l'IRP dal driver che ha eseguito l'operazione più recente, e completa la richiesta.

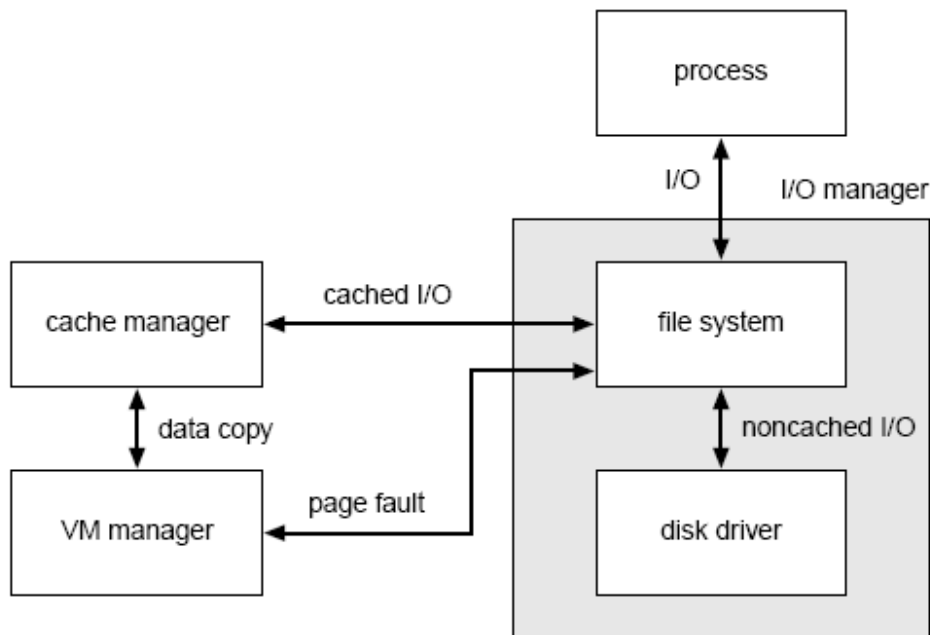
In molti sistemi operativi, la messa in cache è realizzata dal file system. Windows 2000 è, invece, dotato di una funzione centralizzata di cache e il gestore fornisce molto attentamente i servizi di cache per tutti i componenti sotto il controllo del gestore di I/O, e lavora a stretto contatto con il gestore di VM. La dimensione della cache cambia dinamicamente, a seconda di quanta memoria libera sia disponibile nel sistema. Si ricordi che i 2 GB superiori dello spazio degli indirizzi dei processi comprendono l'area di sistema che è identica per tutti i processi. Il gestore di VM alloca fino a a metà di questo spazio alla cache di sistema. Il gestore della cache mappa i file in questo spazio di indirizzamento, e usa le capacità del gestore di VM per maneggiare il file di I/O.

La cache è divisa in blocchi di 256 KB e ogni blocco della cache può contenere una vista (cioè, una regione a 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 spiazamento del file per quella vista, come pure il numero di processi che stanno usando la vista. VACB risiede in un array singolo che è mantenuto dal gestore della cache.

Per ogni file aperto, il gestore della cache mantiene un array indice separato, VACB; e questo array ha un elemento per ogni pezzo da 256 KB del file; quindi, per esempio, un file da 2 MB avrebbe un array indice formato da 8 elementi. Un elemento nell'array indice di VACB punta a VACB se quella porzione di file è in cache; in caso contrario a null.

Quando il gestore di I/O riceve una richiesta di lettura a livello utente, il gestore invia un IRP al gestore della cache (a meno che la richiesta non richieda in modo specifico una lettura non in cache). Il gestore della cache calcola quale elemento dell'array indice VACB di quel file corrisponde allo spiazamento di byte della richiesta: l'ingresso, o punta alla vista in cache o a null. Se è null, il gestore della cache assegna un blocco della cache (e l'ingresso corrispondente nell'array di VACB), e mappa la vista in quel blocco di cache. Il gestore della cache tenta poi di copiare i dati dal file mappato nel buffer del chiamante; se la copia riesce, l'operazione viene completata, mentre se la copia fallisce, a causa di un fault di pagina, provoca che il gestore di VM mandi una richiesta di lettura, non dalla cache, al gestore di I/O. Il gestore di I/O chiede al dispositivo del driver appropriato di leggere i dati, e li restituisce al gestore di VM, che li carica in cache. I dati, ora in cache, vengono copiati nel buffer del chiamante, e, così, si completa la richiesta di I/O. La Figura 6 fornisce una descrizione di tutte queste operazioni. Quando è possibile, le operazioni di I/O sincrone, in cache, non bloccanti, sono gestite dal **meccanismo I/O veloce** (fast I/O mechanism). Questo meccanismo semplicemente copia i dati direttamente da/verso le pagine in cache e utilizza il gestore della cache per eseguire le operazioni necessarie di I/O.

Un'operazione di lettura a livello kernel si svolge in modo simile, salvo che ai dati si può accedere direttamente dalla cache, invece di copiarli in un buffer nello spazio dell'utente.



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

process = processo

I/O manager =gestore I/O

cache manager = gestore di cache

data copy = copia dei dati

VM manager = gestore di VM

cached I/O = I/O in cache

page fault = errore di pagina

file system

non cached I/O = I/O non in cache

disk driver = dispositivo del disco

Per usare i metadati del file system, o le strutture dati che descrivono il file system, il kernel usa l'interfaccia di mappatura del gestore della cache per leggere i metadati.

Per modificare i metadati, il file system usa l'interfaccia di forzatura (pinning) in memoria del gestore della cache. Il **pinning** di una pagina blocca la pagina in una frame di pagina della memoria fisica, in modo che il gestore di VM non possa muovere o scambiare la pagina. Dopo aver aggiornato i metadati, il file system chiede al gestore della cache di sbloccare (unpin) la pagina. Siccome la pagina è stata modificata, essa è marcata come sporca, e il gestore di VM trasferirà la pagina su disco. Si noti che i metadati sono attualmente memorizzati in un normale file.

per migliorare le prestazioni, il gestore della cache mantiene una piccola storia delle richieste di lettura e tenta di predire future richieste. Se il gestore della cache può trovare un modello nelle precedenti tre richieste, tipo un accesso sequenziale in avanti o all'indietro, può riporre in anticipo i dati in cache prima che avvenga la richiesta successiva da parte di una applicazione; quindi l'applicazione può trovare i propri dati già nella cache e può non aver bisogno di aspettare le

operazioni di I/O del disco. L'API di Win32 `OpenFile` e le funzioni `CreateFile` possono essere passate dal flag `FILE_FLAG_SEQUENTIAL_SCAN`, che è un suggerimento al gestore della cache di cercare di riporre in anticipo in cache 192 KB prima delle richieste del thread. Tipicamente, Windows 2000 effettua operazioni di I/O in blocchi di 64 KB o di 16 pagine; pertanto, la lettura anticipata è il triplo della quantità normale.

Il gestore della cache ha anche la responsabilità di comunicare al gestore di VM di trasferire il contenuto della cache. Il comportamento di default del gestore della cache è di write-back in cache: si accumulano scritture per 4 o 5 secondi ed poi il thread dello scrittore della cache si risveglia. Quando è necessaria l'operazione di cache write-through, un processo può posizionare un flag all'apertura di un file, o può chiamare, se necessario, una funzione esplicita di trasferimento in cache.

Un processo di scrittura veloce potrebbe potenzialmente riempire tutte le pagine libere della cache prima che il thread di scrittura in cache abbia la possibilità di svegliarsi e trasferire le pagine su disco. Lo scrittore della cache impedisce ad un processo di arrecare danni al sistema, nel seguente modo: quando l'ammontare di memoria libera in cache diventa bassa, il gestore della cache blocca temporaneamente i processi che tentano di scrivere dati e risveglia il thread dello scrittore della cache al fine di trasferire le pagine su disco. Se il processo a scrittura veloce è un reindirizzatore di rete per un file system di rete, il bloccarlo troppo a lungo potrebbe causare che i trasferimenti di rete vadano fuori sincronismo e quindi di doverli ritrasmettere con conseguente spreco di banda. Per impedire tale spreco, i reindirizzatori di rete possono chiedere al gestore della cache di non lasciare una grande quantità di scritture in cache.

Poiché un file system di rete deve spostare dati fra un disco e l'interfaccia di rete, il gestore della cache fornisce pure un'interfaccia DMA per spostare i dati direttamente, evitando, in tal modo, di copiare i dati attraverso un buffer intermedio.

### 3.3.6 Security Reference Monitor

La natura di Windows 2000, orientata agli oggetti, permette l'uso di un meccanismo uniforme per realizzare la convalida run-time degli accessi e i controlli degli account per ogni entità nel sistema. Ogni qualvolta un processo apre un handle di un oggetto, il **controllo di riferimento di sicurezza** (security reference monitor) controlla il token di sicurezza dei processi e la lista di controllo degli accessi per stabilire se il processo ha i necessari diritti.

### 3.3.7 Gestore Plug-and-Play

Il sistema operativo usa un **gestore plug-and-play** (plug-and-play –PnP– manager) per riconoscere e adattarsi ai cambiamenti di configurazione hardware. Affinchè PnP funzioni, sia il dispositivo che il driver devono supportare lo standard PnP. Il gestore PnP riconosce automaticamente i dispositivi installati e rileva 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 cura di caricare i driver adatti. Questa gestione di risorse hardware, principalmente interrupt e intervalli di memoria I/O, ha lo scopo di determinare una configurazione hardware in cui possano funzionare tutti i dispositivi. Per esempio, se il dispositivo B può usare solo l'interrupt 5, ma il dispositivo A può usare il 5 o il 7, allora il gestore di PnP potrebbe assegnare a B il 5 e ad A il 7. Nelle versioni precedenti, l'utente avrebbe dovuto rimuovere il dispositivo A e modificarlo per poter usare l'interrupt 7, prima di installare il dispositivo B. L'utente, quindi, doveva studiare le risorse di sistema prima di installare un nuovo hardware e scoprire o ricordare quali dispositivi stavano usando quali risorse hardware. La proliferazione di dispositivi PCCARD e USB impone anche che sia supportata la configurazione dinamica delle risorse.

Il gestore di PnP gestisce questa riconfigurazione dinamica nel modo seguente: dapprima, ottiene una lista dei dispositivi da ogni driver di bus (per esempio, PCI, USB), poi carica il driver installato (o

ne installa uno in caso di necessità) e invia un comando `add-device` al driver appropriato per ogni dispositivo. Il gestore PnP configura le assegnazioni ottimali della risorsa e poi invia un comando `start-device` per ogni driver con l'assegnazione della risorsa per quel dispositivo. Se un dispositivo deve essere riconfigurato, il gestore PnP invia un comando `query-stop`, che chiede al driver se può temporaneamente disabilitare il dispositivo. Se il driver risponde positivamente, allora tutte le operazioni pendenti vengono completate e non è permesso di iniziarne di nuove. Successivamente, il gestore di PnP invia un comando di stop; a questo punto può riconfigurare il dispositivo con un altro comando `start-device`. Il gestore di PnP supporta anche altri comandi, quali `query-remove`: questo comando viene usato quando l'utente è pronto ad espellere un dispositivo PCCARD e opera in modo simile a `query-stop`. Il comando `surprise-remove` si usa quando l'operazione di un dispositivo non va a buon fine o, più probabilmente, quando un utente rimuove un dispositivo PCCARD senza usare il programma di utilità che arresta la PCCARD. Il comando `remove` richiede l'arresto del driver tramite il dispositivo e il rilascio di tutte le risorse che sono state allocate al dispositivo.

## 4 Sottosistemi di ambiente

I sottosistemi di ambiente sono processi in modalità utente stratificati sopra i servizi del codice eseguibile nativo di Windows 2000 per permettergli di fare funzionare programmi sviluppati per altri sistemi operativi, incluso Windows a 16 bit, MS-DOS, POSIX e applicazioni OS/2 a 16 bit basate sui caratteri. Ogni sottosistema di ambiente fornisce una API o un ambiente applicativo.

Windows 2000 usa il sottosistema Win32 come ambiente operativo principale, e quindi iniziare tutti i processi. Quando viene eseguita un'applicazione, il sottosistema Win32 chiama il gestore di VM per caricare il codice eseguibile dell'applicazione; il gestore di memoria restituisce uno stato a Win32 che indica il tipo di codice eseguibile. Se non è un eseguibile nativo di Win32, l'ambiente Win32 controlla se è in funzione un sottosistema di ambiente adatto, ed, in caso negativo, lo avvia come un processo in modalità utente; poi, Win32 crea un processo per fare funzionare l'applicazione, e passa il controllo al sottosistema di ambiente.

Il sottosistema di ambiente usa la funzione LPC di Windows 2000 per ottenere i servizi del kernel per il processo; questo metodo aiuta Windows 2000 ad essere robusto, perché i parametri passati ad una chiamata di sistema possono essere controllati se sono corretti, prima che sia invocata la procedura attuale del kernel. Windows 2000 proibisce alle applicazioni di mescolare le procedure API provenienti da ambienti diversi. Per esempio, un'applicazione Win32 non può chiamare una procedura POSIX.

Dal momento che ogni sottosistema viene fatto funzionare come processo separato in modalità utente, un arresto in uno di essi non ha alcun effetto sugli altri; l'eccezione è Win32, che fornisce le funzioni alla tastiera, al mouse e al video: in caso di guasto, il sistema viene completamente disabilitato.

L'ambiente Win32 suddivide in categorie le applicazioni: o grafiche o basate sui caratteri, ove una *applicazione basata sui caratteri* crede che l'uscita interattiva vada ad un display ASCII 80 x 24. Win32 trasforma l'uscita di un'applicazione basata sui caratteri in una rappresentazione grafica in finestra. Questa trasformazione è facile: ogni volta che si chiama una procedura di output, il sottosistema di ambiente chiama una procedura di Win32 per visualizzarne il testo. Siccome l'ambiente di Win32 esegue questa funzione per tutte le finestre basate sui caratteri, si può trasferire il testo dello schermo fra le finestre tramite clipboard. Questa trasformazione funziona per le applicazioni MS-DOS, come pure per le applicazioni POSIX a riga di comando.

## 4.1 Ambiente MS-DOS

L'ambiente MS-DOS non ha la complessità degli altri sottosistemi di ambiente di Windows 2000; è fornito da un'applicazione di Win32 chiamata **macchina virtuale DOS** (virtual DOS machine: VDM). Essendo VDM proprio un processo in modalità utente, esso è paginato e inviato come qualsiasi altro thread di Windows 2000. VDM ha un'**unità di esecuzione delle istruzioni** (instruction-execution unit) per eseguire o emulare istruzioni Intel 486; VDM fornisce anche procedure per emulare la ROM BIOS di MS-DOS e i servizi di interrupt software "int 21", ed è dotata di device driver virtuali per lo schermo, la tastiera e le porte di comunicazione. VDM è basata sul codice sorgente di MS-DOS 5.0 e mette a disposizione delle applicazioni almeno 620 KB di memoria.

La shell di comando di Windows 2000 è un programma che crea una finestra che assomiglia ad un ambiente MS-DOS; si possono fare funzionare eseguibili sia a 16 che a 32 bit. Quando un'applicazione MS-DOS è in funzione, la shell di comando inizia un processo VDM per eseguire il programma.

Se Windows 2000 funziona in un processore x86, le applicazioni grafiche MS-DOS sono in modalità a schermo intero e le applicazioni basate sui caratteri possono funzionare sia a schermo intero che in finestra. Se Windows 2000 funziona su un'architettura di processore differente, tutte le applicazioni MS-DOS funzioneranno in finestra. Alcune applicazioni MS-DOS accedono direttamente all'hardware del disco, ma non riescono a funzionare in Windows 2000 poiché l'accesso al disco è privilegiato per proteggere il file system. In generale, le applicazioni MS-DOS che accedono direttamente all'hardware non riusciranno a funzionare in Windows 2000.

Siccome MS-DOS non è un ambiente di elaborazione in multitasking, sono state scritte alcune applicazioni impegnare la CPU, per esempio, mediante l'uso di cicli di operazioni per provocare ritardi temporali o pause nell'esecuzione. Il meccanismo di priorità del dispatcher di Windows 2000 rileva tali ritardi ed automaticamente abbassa il consumo della CPU (e provoca che l'applicazione offendentente operi in modo errato).

## 4.2 Ambiente di Windows a 16 bit

L'ambiente di esecuzione di Win16 è fornito da una VDM che incorpora software supplementare, chiamato *Windows on Windows*, che fornisce le procedure del kernel di Windows 3.1 e le procedure stub per il gestore delle finestre e le funzioni di GDI. Le procedure stub chiamano le sottoprocedure appropriate di Win32, convertendo (*thunking*) indirizzi a 16 bit in quelli a 32 bit. Applicazioni che fanno affidamento sulla struttura interna del gestore di finestra a 16 bit o su GDI possono non funzionare, perché Windows on Windows non realizza completamente le API a 16 bit.

Windows on Windows può funzionare in multitask con altri processi in Windows 2000, ma assomiglia, sotto molti aspetti, a Windows 3.1. Solo un'applicazione alla volta di Win16 può essere in funzione, tutte le applicazioni sono a singolo thread e risiedono nello stesso spazio di indirizzi, condividendo la stessa coda di input. Queste caratteristiche implicano che un'applicazione che si ferma per ricevere l'input bloccherà tutte le altre applicazioni di Win16, proprio come in Windows 3.x ed un'applicazione di Win16 può arrestare tutte le altre applicazioni di Win16 corrompendo lo spazio degli indirizzi. Ambienti multipli di Win16 possono, tuttavia, coesistere, a tale scopo si usa, da riga di comando, *start/separate win16application*.

### **4.3 Ambiente di Win32**

Il sottosistema principale di Windows 2000 è Win32 che fa funzionare applicazioni Win32 e gestisce completamente la tastiera, il mouse e lo schermo I/O. Siccome è l'ambiente di controllo, esso è progettato per essere estremamente robusto e parecchie caratteristiche di Win32 contribuiscono a tale robustezza. A differenza dell'ambiente di Win16, ogni processo di Win32 ha la propria coda di input. Il gestore delle finestre spedisce tutti gli input di sistema alla coda di input dei processi appropriati, in modo che un processo guasto non blocchi l'input degli altri. Il kernel di Windows 2000 fornisce anche l'elaborazione in multitasking con prerilascio (preemptive), che permette all'utente di terminare le applicazioni che sono andate a buon fine o che non sono più necessarie. Win32 convalida anche tutti gli oggetti prima di usarli, per prevenire arresti che potrebbero altrimenti accadere se un'applicazione cercasse di usare un handle non valido o errato. Il sottosistema di Win32 verifica il tipo dell'oggetto a cui un handle punta prima di usarlo. I contatori dei riferimenti, mantenuti dal gestore dell'oggetto, impediscono la cancellazione degli oggetti mentre sono ancora in uso, e pure l'uso dopo la cancellazione.

### **4.4 Sottosistema POSIX**

Il sottosistema POSIX è progettato per far funzionare applicazioni POSIX che sono conformi allo standard POSIX.1, che è basato sul modello UNIX. Le applicazioni POSIX possono essere iniziate dal sottosistema Win32 o da un'altra applicazione POSIX. Le applicazioni POSIX usano il server di sottosistema PSXSS.EXE, la libreria di collegamento dinamico PSXDLL.DLL ed il gestore della sessione della console di POSIX: POSIX.EXE.

Sebbene lo standard POSIX non fornisca specifiche sulla stampa, le applicazioni POSIX possono usare le stampanti in modo trasparente tramite il meccanismo di reindirizzamento di Windows 2000. Le applicazioni POSIX hanno accesso a qualsiasi file system in Windows 2000. L'ambiente POSIX fa rispettare permessi di tipo UNIX sugli alberi del direttorio. Parecchie funzioni di Win32 non sono supportate dal sottosistema POSIX, inclusi i file a memoria mappata, la rete, la grafica e lo scambio di dati dinamico.

### **4.5 Sottosistema OS/2**

Sebbene originariamente, Windows 2000 fosse stato progettato per fornire un ambiente robusto del sistema operativo OS/2, il successo di Windows ha portato ad un cambiamento durante lo sviluppo iniziale di Windows 2000; l'ambiente di Windows si è trasformato in ambiente di default. Di conseguenza, Windows 2000 fornisce soltanto funzioni limitate del sottosistema di ambiente di OS/2. Le applicazioni di OS/2 1.x, basate sui caratteri, possono funzionare solo sul Windows 2000 montato sui calcolatori Intel x86. Le applicazioni in real-mode di OS/2 possono funzionare su tutte le piattaforme che usano l'ambiente MS-DOS. Applicazioni collegate, che hanno doppio codice sia per MS-DOS che per OS/2, funzionano nell'ambiente OS/2 a meno che l'ambiente OS/2 non sia disabilitato.

### **4.6 Sottosistemi di logon e di sicurezza**

Un utente deve essere autenticato dal sottosistema di logon in Windows 2000 prima di poter accedere agli oggetti di Windows 2000; per essere autenticato, un utente deve avere un conto e fornire la password di quell'account.

Il sottosistema di sicurezza genera dei token di accesso per visualizzare gli utenti nel sistema. Chiama un **pacchetto di autenticazione** (authentication package) per fornire autenticazione tramite informazioni provenienti dal sottosistema di logon o dal server di rete. Tipicamente, il pacchetto di autenticazione controlla semplicemente le informazioni dell'account in un database locale e la correttezza della password. Il sottosistema di sicurezza genera poi il token di accesso relativo all'identificatore dell'utente, contenente i privilegi appropriati, i limiti di quota e gli identificatori del gruppo. Ogni volta che l'utente tenta di accedere ad un oggetto nel sistema, quale l'apertura di un handle dell'oggetto, il token di accesso viene passato al controllo di riferimento di sicurezza, che controlla i privilegi e le quote. Kerberos è il pacchetto di autenticazione di default per i domini di Windows 2000.

## 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 della dimensione a 2 GB e mancanza di protezione dell'accesso per i file. Il file system FAT a 32 bit ha risolto i problemi di frammentazione e di dimensione, ma le sue prestazioni e caratteristiche sono ancora deboli in confronto con i moderni file system. Il file system NTFS è molto migliore: è stato progettato per includere molte caratteristiche, tra cui il recupero dei dati, la sicurezza, la tolleranza agli errori, grandi file e grandi file system, flussi di dati multipli, nomi UNICODE e compressione del file. Per questioni di compatibilità, Windows 2000 fornisce supporto sia per la FAT che per HPFS di OS/2.

### 5.1 Disposizione interna

L'entità fondamentale in NTFS è un volume che viene creato dal programma di utilità dell'amministratore del disco di Windows 2000 ed è basato su una partizione logica del disco. Il volume può occupare una porzione di un disco, o l'intero disco, o può estendersi su parecchi dischi.

NTFS non si occupa dei singoli settori di un disco, ma invece usa i cluster come unità di allocazione. Un cluster è un certo numero di settori del disco che è una potenza di 2. La dimensione del cluster è configurata quando viene formattato il file system NTFS, e la dimensione di default del cluster coincide con la dimensione del settore per volumi fino a 512 MB; è di 1 Kb per volumi fino a 1 GB; è di 2 KB per volumi fino a 2 GB, e di 4 KB per volumi più grandi. La dimensione del cluster è molto più piccola di quella del file system FAT a 16 bit e questa piccola dimensione riduce la quantità di frammentazione interna. Come esempio, si consideri un disco da 1.6 GB con 16.000 file; se si usa un file system FAT-16, 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 potrebbero venire persi quando si memorizzano gli stessi file.

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

Un file in NTFS non è un semplice flusso di byte, come in MS-DOS o in UNIX; piuttosto, è un oggetto strutturato di **attributi** (attributes) in cui ogni attributo di un file è un flusso indipendente di byte che può essere creato, cancellato, letto e scritto. Alcuni attributi sono standard per tutti i file, compreso il nome del file (o i nomi, se il file ha degli pseudonimi), l'ora di creazione e il descrittore di sicurezza che specifica il controllo di accesso. Altri attributi sono specifici di determinati tipi di file. Una directory ha attributi che realizzano un indice per i nomi dei file in essa. In generale, gli attributi possono essere aggiunti in base alle necessità e vi si accede usando la nomenclatura *nome del file: attributo*. I file di dati più tradizionali hanno un attributo di dati *non specificato* che

contiene tutti i dati del file. Si noti che NTFS restituisce la dimensione di un attributo non specificato solo in risposta ad operazioni di interrogazione del file, come quando si esegue il comando `dir`. Chiaramente, alcuni attributi sono piccoli mentre altri sono grandi.

Ogni file in NTFS è descritto da uno o più record in un array memorizzato in un file speciale chiamato tabella principale del file (MFT). La dimensione di un record è determinata nel momento in cui si crea un file system e varia tra 1 e 4 Kb. Piccoli attributi sono memorizzati nel record stesso di MFT e sono chiamati **attributi residenti** (resident attributes). Grandi attributi, quali dati anonimi non specificati, chiamati **attributi non residenti** (nonresident attributes), sono memorizzati in uno o più **extent** (estensioni) contigui su disco, e un puntatore ad ogni estensione è memorizzato nel record MFT. Per un file piccolino, anche l'attributo dei dati può trovare posto nel record MFT. Se un file ha molti attributi, o è molto frammentato e quindi sono necessari molti puntatori per individuare tutti i frammenti, un record in MFT potrebbe non essere sufficientemente grande; in tal caso il file è descritto da un record chiamato **record di base del file** (base file record), che contiene i puntatori ai record di overflow che contengono ulteriori puntatori e attributi.

Ogni file in un volume NTFS ha un identificatore unico chiamato **riferimento del file** (file reference) che è una quantità a 64 bit costituita da un numero a 48 bit del file e da un numero progressivo a 16 bit. Il numero del file è il numero del record (cioè, la posizione nell'array) nella MFT che descrive il file. Il numero progressivo è incrementato ogni volta che un ingresso nella MFT è riutilizzato e tale incremento permette a NTFS di effettuare controlli di consistenza interna, come prendere un vecchio riferimento ad un file cancellato dopo che l'ingresso in MFT è stato riutilizzato per un nuovo file.

Come in MS-DOS e in UNIX, lo spazio del nome in NTFS è organizzato come gerarchia di direttori. Ogni direttorio usa una struttura dati chiamata **albero B+ (B+ tree)** per memorizzare un indice dei file in quel direttorio. Si usa un B+ tree 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 dell'indice** di un direttorio contiene il livello superiore del B+ tree. Nel caso di un direttorio ampio, il livello superiore contiene i puntatori alle estensioni del disco che contengono il resto dell'albero. Ogni voce nel direttorio contiene il nome e il riferimento del file, come pure una copia della marca di tempo aggiornata 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 generarne una lista: tutti i nomi dei file, le dimensioni e gli orari aggiornati sono disponibili nel direttorio stesso, in modo che non sia necessario recuperare per ogni file questi attributi dalle voci nella MFT.

Tutti i metadati del volume NTFS sono memorizzati in file; il primo file è la MFT, il secondo file, che è usato in caso di recupero se la MFT è danneggiata, contiene una copia dei primi 16 ingressi nella MFT; anche i pochi file successivi sono particolari e sono chiamati file di log, file del volume, tabella di definizione degli attributi, direttorio radice, file bitmap, file di avvio, e file dei cluster difettosi. Il *file di log* (Paragrafo 5.2) registra tutti gli aggiornamenti ai metadati nel file system. Il **file del volume** (volume file) contiene il nome del volume, la versione NTFS con cui è stato formattato il volume e un bit che indica se il volume può essere stato corrotto e ha bisogno di essere controllato per la consistenza. La **tabella di definizione degli attributi** (attribute-definition table) indica quali tipi di attributo sono usati nel volume e quali operazioni si possono compiere su ognuno di essi. Il **direttorio radice** (root directory) è l'indice di più alto livello nella gerarchia del file system. Il **file bitmap** (bitmap file) indica quali cluster nel volume sono allocati ai file e quali sono liberi. Il **file di avvio** (boot file) contiene il codice avvio di Windows 2000 e deve essere situato ad un indirizzo particolare del disco in modo da poterlo ritrovare facilmente mediante un semplice caricatore di inizializzazione della ROM; il file di avvio contiene l'indirizzo fisico della MFT. Da ultimo, il **file dei cluster difettosi** (bad-cluster file) tiene traccia di qualsiasi zona difettosa nel volume; NTFS usa questo record per il recupero dagli errori.

## 5.2 Recupero

In molti file system semplici, un guasto all'alimentazione nel momento sbagliato può danneggiare le strutture dati del file system al punto che i dati nell'intero volume potrebbero venire mescolati alla rinfusa. Molte versioni di UNIX memorizzano dei metadati ridondanti sul disco e poi li recuperano dopo un guasto mediante il programma `fsck` per controllare tutte le strutture dati del file system e per riportarle in modo forzoso in uno stato consistente. Il loro ristabilimento spesso coinvolge la cancellazione di file danneggiati e la liberazione di cluster di dati che erano stati scritti con i dati dell'utente, ma non erano stati registrati correttamente nelle strutture dei metadati del file system; questo controllo può essere un processo lento e può far perdere un numero significativo di dati.

NTFS usa un approccio differente riguardo la robustezza del file system, infatti tutti gli aggiornamenti della struttura dati avvengono all'interno di transazioni. Prima di alterare una struttura dati, la transazione scrive un record di log che contiene informazioni per il redo e undo; dopo il cambiamento della struttura dati, la transazione scrive un record di commit nel log per indicare il successo della transazione. Dopo un blocco, il sistema può ristabilire le strutture dati del file system ad uno stato consistente elaborando i record di log; dapprima, rifacendo le operazioni delle transazioni eseguite, poi eliminando le operazioni delle transazioni che non si sono svolte con successo prima del blocco. Periodicamente (di norma ogni 5 secondi), un record di controllo (checkpoint) viene scritto nel log; il sistema non ha bisogno dei record del log prima del punto di controllo per recuperare da un crash; essi possono essere scartati, in modo che il file di log non cresca a dismisura. Dopo l'avvio, la prima volta in cui si accede ad un volume NTFS, questi esegue automaticamente il recupero del file system.

Questo schema non garantisce che tutti i contenuti del file utente siano corretti dopo un blocco; assicura solo che le strutture dati del file system (i file metadati) non siano danneggiate e riflettano un certo stato consistente che esisteva prima del blocco. Si potrebbe estendere lo schema di transazione ai file utente, ma l'overhead potrebbe alterare le prestazioni del file system.

Il log è memorizzato nel terzo file di metadati all'inizio del volume, ed è creato con una dimensione massima fissa, al momento della formattazione del file system. Consiste di due sezioni: l'**area di log** (logging area), che è una coda circolare dei record di log e l'**area di riavviamento** (restarting area) che mantiene le informazioni di contesto, quale la posizione nell'area di log da cui NTFS dovrebbe incominciare a leggere durante un recupero; infatti, l'area di riavviamento, mantiene due copie delle proprie informazioni, in modo che il recupero sia ancora possibile se una copia viene danneggiata durante un blocco.

La funzionalità di log è fornita dal **servizio del file di log** (log-file service) di Windows 2000. In aggiunta alla scrittura dei record di log e alle azioni di recupero, il servizio del file di log tiene traccia dello spazio libero nel file di log. Se lo spazio libero diventa troppo basso, il servizio del file di log accoda le transazioni in attesa e NTFS ferma tutte le nuove operazioni di I/O. Dopo che le operazioni avviate si completano, NTFS chiama il gestore della cache per trasferire tutti i dati, poi ripristina il file di log ed esegue le transazioni in coda.

## 5.3 Sicurezza

La sicurezza di un volume NTFS è derivata dal modello di oggetto di Windows 2000. Ogni file oggetto ha un attributo descrittore di sicurezza memorizzato nel proprio record MFT. Questo attributo contiene il token di accesso del proprietario del file e una lista di controllo di accesso che elenca i privilegi assegnati ad ogni utente che può accedere al file.

## 5.4 Gestione del volume e tolleranza agli errori

FtDisk è il driver del disco di Windows 2000 tollerante agli errori. Una volta installato, fornisce parecchi modi per combinare i dischi in un unico volume logico, al fine di migliorare le prestazioni, la capienza, o l'affidabilità.

Un modo di combinare dischi multipli è di concatenarli logicamente per formare un grande volume logico, come si vede in Figura 7. In Windows 2000, questo volume logico è chiamato **insieme di volumi** (volume set), e può essere formato, al massimo, da 32 partizioni fisiche. Un volume set che contiene un volume NTFS può essere ampliato senza alterare i dati già memorizzati nel file system. La bitmap dei metadati nel volume NTFS viene semplicemente estesa per coprire lo spazio aggiunto di recente. NTFS continua ad usare lo stesso meccanismo LCN che usa per un singolo disco fisico, e il driver FtDisk fornisce la mappatura dallo spiazamento di un volume logico in quello di un particolare disco.

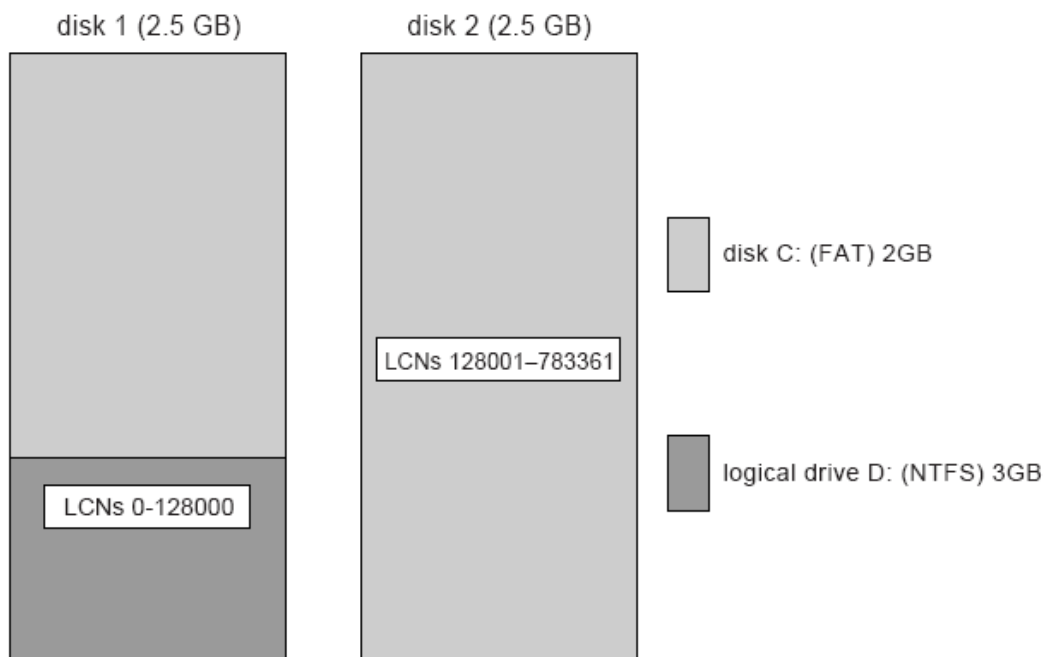
Un altro modo di combinare partizioni fisiche multiple è di numerare i loro blocchi nella modalità round-robin per formare ciò che è chiamato **stripe set** (insieme di strisce), come è mostrato in Figura 8. Questo schema è anche chiamato RAID di livello 0, o **disk striping**. FtDisk usa una dimensione 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 ha contribuito a 64 KB di spazio; poi, l'algoritmo di allocazione ritorna al primo disco, allocando il secondo blocco di 64 Kb. Uno stripe set forma un grande volume logico, ma la disposizione fisica può migliorare la larghezza di banda di I/O, poiché, nel caso di un grande I/O, tutti i dischi possono trasferire i dati in parallelo. Per maggiori informazioni su RAID, consultare il Paragrafo 14.5.

Una variante di questa idea è lo **stripe set con parità** (stripe set with parità), che è 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 stripe di dati, che si trovano su sette dischi separati, ci sarà una stripe di parità nell'ottavo disco. La stripe di parità contiene l'*exclusive or* (or esclusivo) a livello di byte delle stripe di dati; se una qualsiasi delle otto stripe viene distrutta, il sistema può ricostruirne i dati calcolando l'*or esclusivo* dalle rimanenti sette.

Questa capacità di ricostruire i dati rende il disk array molto meno incline a perdere i dati nel caso di guasto del disco. Si noti che un aggiornamento ad una stripe di dati richiede anche il ricalcolo della stripe di parità. Sette scritture concorrenti su sette bande di dati differenti richiederanno l'aggiornamento di sette bande di parità sullo stesso disco e quel disco potrebbe avere sette volte il carico di I/O dei dischi dei dati. Per evitare di creare questo collo di bottiglia, conviene sparpagliare le bande di parità su tutti i dischi, assegnando loro il round-robin. Per costruire uno stripe set con parità, c'è bisogno di almeno tre partizioni su tre dischi separati.

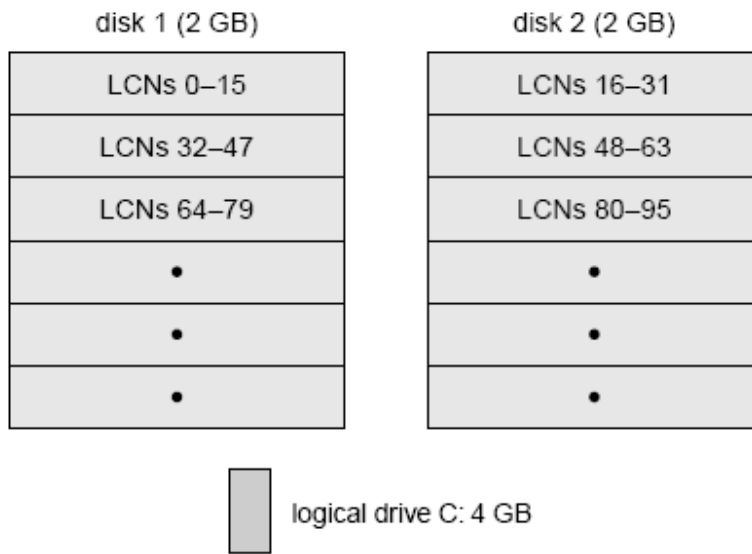
Uno schema ancora più robusto è chiamato **duplicazione del disco** (disk mirroring) o RAID di livello 1 che è presentato in Figura 10. Un **insieme duplicato** (mirror set) comprende due partizioni di equal dimensione su due dischi, tali che il contenuto dei loro dati sia identico. Quando un'applicazione scrive i dati in un mirror set, FtDisk li scrive su entrambe le partizioni. Se una partizione sbaglia, FtDisk possiede un'altra copia sicura memorizzata nel mirror set. I mirror set possono anche migliorare le prestazioni, poiché le richieste di lettura possono essere suddivise fra i due mirror, assegnando ad ognuno di essi metà del carico di lavoro. Per proteggersi contro i guasti di un controller del disco, si possono collegare i due dischi del mirror a due controller separati: questa modalità è chiamata **insieme duplex** (duplex set).

Nel caso di settori del disco che si rovinano, FtDisk usa una tecnica hardware chiamata sector sparing (risparmio dei settori) e NTFS usa una tecnica software chiamata rimappatura del cluster: il sector sparing è una possibilità hardware fornita da molti lettori di dischi; quando un disco viene formattato, crea una mappatura fra i numeri dei blocchi logici e i settori validi del disco, e lascia anche dei settori supplementari non mappati, come parti di ricambio.



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

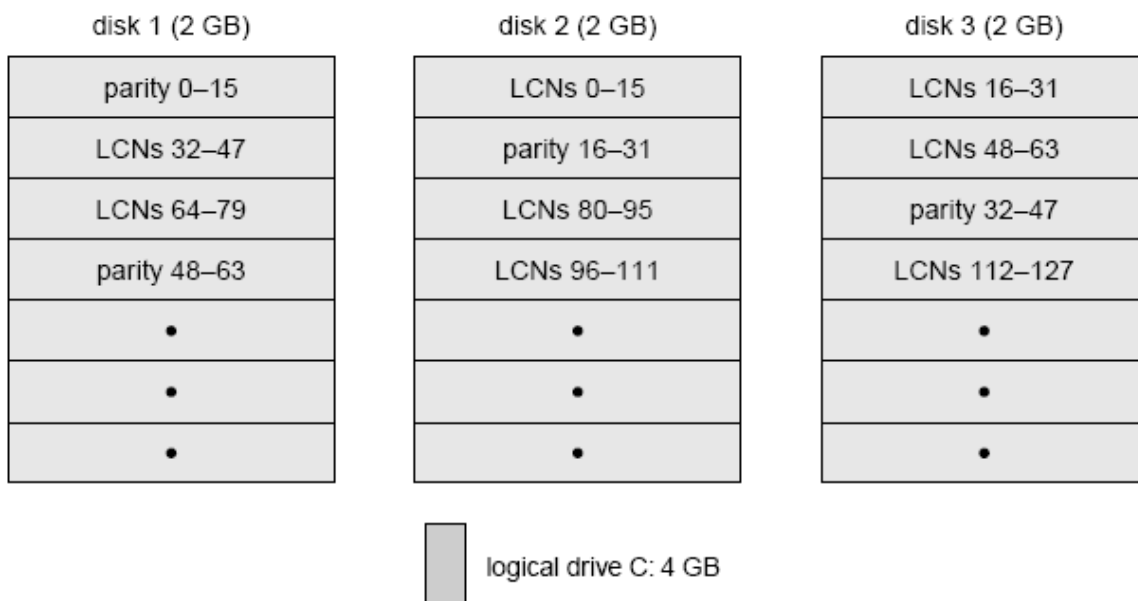
disk= disco  
local drive = drive locale



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

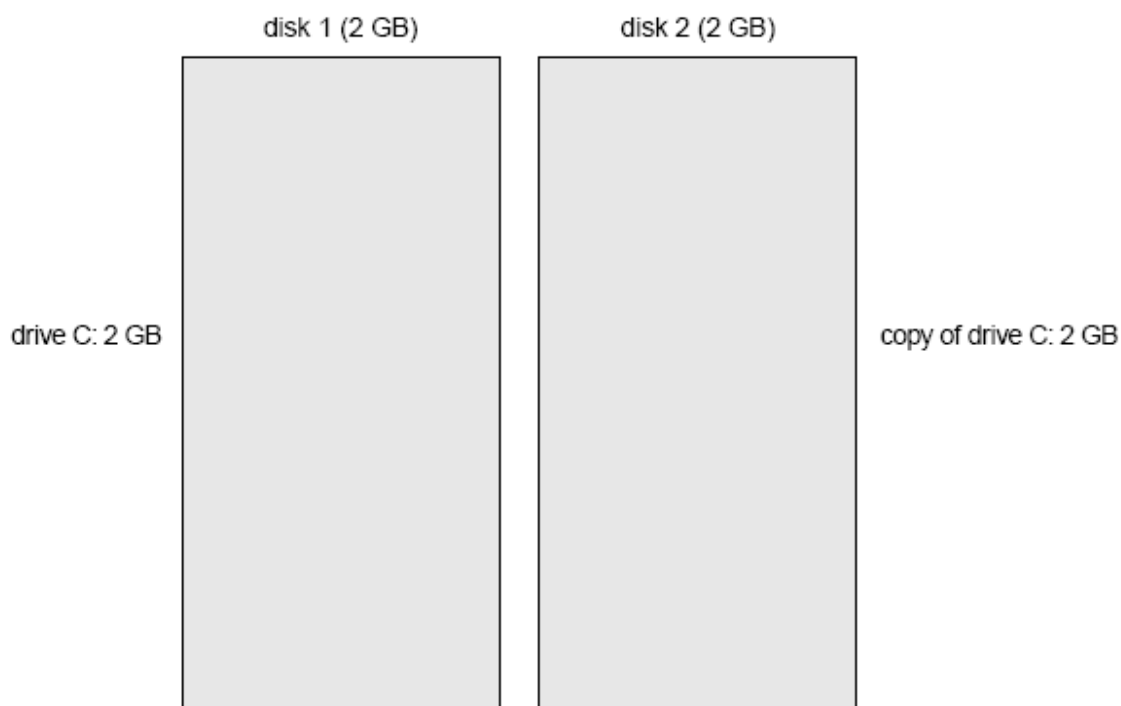
disk= disco

local drive = drive locale



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

disk= disco  
parity = parità  
local drive = drive locale



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

disk = disco  
drive  
copy of drive = copia del drive

Se un settore si danneggia, FtDisk dirà al disco di sostituirlo con una parte di ricambio. La **rimappatura del cluster** (cluster remapping) è una tecnica software usata dal file system. Se il blocco del disco fallisce, NTFS sostituirà un blocco differente e non allocato cambiando tutti i puntatori interessati nella MFT. NTFS tiene pure nota che il blocco difettoso non dovrà mai essere assegnato a qualsiasi file.

Quando un blocco del disco si rovina, il normale risultato è una perdita di dati; le tecniche del settore di scorta e della rimappatura del cluster possono essere combinate in volumi a tolleranza di errori, quali gli stripe set, per mascherare il guasto di un blocco del disco. Se una lettura fallisce, il

sistema ricostruisce i dati mancanti leggendo il mirror o calcolando la parità mediante l'or esclusivo in uno stripe set con parità; i dati ricostruiti vengono memorizzati in una nuova locazione che è ottenuta dal settore di scorta o dalla rimappatura del cluster.

## 5.5 Compressione

NTFS può eseguire la compressione dei dati su file individuali o su tutti i file di dati in un direttorio. Per comprimere un file, NTFS suddivide i dati in **unità di compressione** (compression units) che sono blocchi di 16 cluster contigui. Quando si scrive in ogni unità di compressione, si applica un algoritmo di compressione dei dati. Se il risultato occupa meno di 16 cluster, viene memorizzata la versione compressa. In lettura, NTFS può determinare se i dati sono stati compressi: in caso affermativo, la lunghezza dell'unità di compressione memorizzata è meno di 16 cluster. Per migliorare le prestazioni durante la lettura di unità di compressione contigue, NTFS preleva e decompone in anticipo, prima delle richieste da parte delle applicazioni.

Per file sparsi o file che contengono principalmente degli zero, NTFS usa un'altra tecnica per risparmiare spazio: i cluster che contengono tutti zero non sono realmente allocati o memorizzati su disco, invero gli intervalli sono lasciati nella sequenza di numeri dei cluster virtuali memorizzati nell'ingresso della MFT per il file; quando NTFS legge un file, se trova un intervallo nei numeri del cluster virtuale, riempie proprio con degli zero la porzione del buffer del chiamante. Una tecnica simile è usata anche in UNIX.

## 5.6 Punti di riscansione

I **punti di riscansione** (reparse point) sono una nuova caratteristica del file system che di fatto restituisce un codice di errore una volta che vi si accede. I dati di riscansione indicano poi al gestore di I/O che cosa fare.

I punti di montaggio sono un'altra caratteristica aggiunta a Windows 2000 e sono una forma di punti di riscansione. A differenza dei sistemi UNIX, le versioni precedenti di Windows non fornivano alcun modo logico per unire le partizioni; ad ogni partizione era assegnata una lettera del disco diversa da quella di ogni altra partizione, ciò significava, tra le altre cose, che se un file system si riempiva, la struttura delle directory doveva essere cambiata per aggiungere ulteriore spazio. I punti di montaggio permettono di creare un nuovo volume in un altro disco, di muovere vecchi dati nel nuovo volume e di montare poi il nuovo volume nel posto originale. I dati saranno quindi ancora utilizzabili dai programmi installati, poiché i dati appariranno nello stesso posto di prima. Il punto di montaggio è realizzato come punto di riscansione con i dati di riscansione che contengono il vero nome del volume.

Anche la funzione **servizi di memorizzazione remota** (remote storage service) usa i punti di riscansione; quando un file è mosso verso una memoria non connessa, i dati originali del file sono rimpiazzati con un punto di riscansione che contiene informazioni sulla localizzazione del file. Se si accede al file in un secondo tempo, esso viene ritrovato, e il punto di riscansione viene sostituito con i dati del file. Per maggiori informazioni sulla memorizzazione gerarchica, si consulti il Paragrafo 14.8.1.

## 6 Rete

Windows 2000 supporta le reti sia peer-to-peer sia client-server e possiede funzioni di gestione della rete. I componenti di rete in Windows 2000 realizzano il trasporto dei dati, la comunicazione tra processi, la condivisione dei file in rete, e la capacità di inviare job di stampa a stampanti remote.

Nella descrizione della rete in Windows 2000, faremo riferimento a due interfacce interne di rete chiamate: **specifica di 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 in 1989 da Microsoft e da 3Com per separare gli adattatori di rete dai protocolli di trasporto, in modo da poter cambiare gli uni senza influenzare gli altri. NDIS risiede a livello di interfaccia fra lo strato di controllo data link e quello di controllo di accesso al mezzo nel modello OSI, e permette a molti protocolli di operare su molti adattatori di rete differenti. In termini di modello OSI, TDI è l'interfaccia fra lo strato di trasporto (strato 4) e lo strato di sessione (strato 5); questa interfaccia permette a qualsiasi componente dello strato di sessione di usare qualsiasi meccanismo di trasporto disponibile. (Un ragionamento simile aveva condotto al meccanismo dei flussi in UNIX). TDI supporta il trasporto basato sia sulla connessione che senza connessione, e ha funzioni per inviare qualunque tipo di dati.

## 6.1 Protocolli

Windows 2000 realizza i protocolli di trasporto come driver che possono venire caricati e scaricati dinamicamente dal sistema, anche se, in pratica, il sistema deve essere avviato di nuovo dopo un cambiamento. Windows 2000 è dotato di parecchi protocolli di rete.

Il protocollo di **blocco del messaggio del server** (server message-block: SMB) è stato, per la prima volta, introdotto in MS-DOS 3.1, ed il sistema lo usa per inviare richieste di I/O in rete. Il protocollo SMB è dotato di quattro tipi di messaggio: i messaggi di `Session control` (controllo di sessione) sono comandi per iniziare e concludere una connessione del reindirizzatore con una risorsa condivisa nel server; un reindirizzatore usa messaggi `File` per accedere ai file nel server; il sistema usa messaggi `Printer` per inviare i dati a una coda di stampa remota e per ricevere a sua volta informazioni di stato; il messaggio `Message` viene usato per comunicare con un'altra workstation.

Il **sistema di input/output di base della rete** (network basic input/output system: NetBIOS) è un'interfaccia di astrazione dall'hardware per le reti, analoga all'interfaccia di astrazione dall'hardware del BIOS progettata per PC in cui funziona MS-DOS. NetBIOS, sviluppato all'inizio degli anni 80, è divenuto un'interfaccia di programmazione standard. NetBIOS viene usato per attribuire nomi logici in rete, per stabilire connessioni o **sessioni** (sessions) logiche tra due nomi logici in rete e per supportare trasferimenti di dati affidabili in una sessione tramite richieste SMB o NetBIOS.

L'**interfaccia utente estesa NetBIOS** (NetBIOS extended user interface: NetBEUI) è stata introdotta da IBM nel 1985, come semplice ed efficiente protocollo di rete per supportare fino a 254 macchine. È il protocollo di default di Windows 95 e di Windows for Workgroups. Windows 2000 usa NetBEUI quando vuole condividere risorse con queste reti. Alcune limitazioni di NetBEUI consistono nell'uso del nome reale di un computer come indirizzo e nell'impossibilità di supportare l'instradamento.

La suite di protocolli TCP/IP, usati in Internet, è divenuta lo standard di fatto dell'infrastruttura di rete, ed è ampiamente supportata. Windows 2000 usa TCP/IP per collegarsi ad un'ampia varietà di sistemi operativi e di piattaforme hardware; il pacchetto TCP/IP di Windows 2000 include il semplice protocollo di gestione della rete (SNMP), il protocollo dinamico di configurazione dell'host (DHCP), il servizio del nome Internet di Windows (WINS) e il supporto NetBIOS.

Il **protocollo di collegamento privato punto a punto** (point-to-point tunnelling protocol: PPTP) è un protocollo fornito da Windows 2000 per permettere di comunicare fra i moduli del server di accesso remoto che funzionano su macchine Windows 2000 Server ed altri sistemi client che sono connessi in Internet. I server di accesso remoto possono cifrare i dati inviati alla connessione e supportano, in Internet, reti virtuali private multiprotocollo.

I protocolli Novell NetWare (servizio di datagramma IPX dello strato di trasporto SPX) sono ampiamente usati nelle PC LAN. Il protocollo NWLink di Windows 2000 collega NetBIOS a reti NetWare. In combinazione con un reindirizzatore (tipo Microsoft's Client Service for Netware o Novell's NetWare Client per Windows 2000), questo protocollo permette ad un client di Windows 2000 di collegarsi ad un server NetWare.

Windows 2000 usa il **protocollo di controllo di collegamento dei dati** (data-link control protocol: DLC) per accedere a mainframe IBM e stampanti HP che sono collegati direttamente alla rete; questo protocollo non è invece usato dai sistemi Windows 2000.

Il **protocollo Appletalk** (AppleTalk protocol) è stato progettato come connessione a basso costo da Apple per permettere ai computer Macintosh di condividere i file; i sistemi Windows 2000 possono condividere file e stampanti con computer Macintosh tramite Appletalk se un server Windows 2000 in rete sta facendo funzionare i servizi di Windows 2000 per il pacchetto Macintosh.

## 6.2 Meccanismi di elaborazione distribuita

Sebbene Windows 2000 non sia un sistema operativo distribuito, esso supporta applicazioni distribuite; i meccanismi che supportano l'elaborazione distribuita in Windows 2000 includono NetBIOS, pipe con nome (named pipe) e caselle di posta (mailslot), socket, RPC e scambio dinamico dei dati in rete (NetDDE).

In Windows 2000, le applicazioni di NetBIOS possono comunicare in rete tramite NetBEUI, NWLink, o TCP/IP.

Le **named pipe (pipe con nome)** sono un meccanismo di messaggistica orientato alla connessione e sono state originariamente sviluppate come interfaccia ad alto livello per le connessioni in rete NetBIOS. Un processo può anche usare le named pipe per comunicare con altri processi nella stessa macchina. Siccome le named pipe si raggiungono 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 ha un formato che si chiama **convenzione di denominazione uniforme** (uniform naming convention: UNC) e 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 resa disponibile agli utenti della rete, come i direttori, i file, le named pipe e le stampanti; la parte x \ y \ z è un normale percorso del nome del file.

**Mailslot** (casella di messaggi) è un meccanismo di messaggi senza connessione. Non è affidabile, in quanto un messaggio inviato ad un mailslot può andare perso prima che il destinatario in questione lo riceva. Mailslot è usato in applicazioni di radiodiffusione, quali la ricerca di componenti in rete ed è usato anche dal servizio Windows 2000 Computer Browser.

**Winsock** è l'API dei socket di Windows 2000 ed è un'interfaccia dello strato di sessione che è in gran parte compatibile con i socket UNIX, ma con alcune estensioni di Windows 2000. Fornisce un'interfaccia standardizzata a molti protocolli di trasporto che possono avere schemi di indirizzamento differenti, in modo che qualsiasi applicazione Winsock possa funzionare su qualunque stack del protocollo Winsock compatibile.

Un **RPC** è un meccanismo client-server che permette, ad un'applicazione su una macchina, di eseguire una chiamata della procedura da codificare su un'altra macchina. Il client chiama una procedura locale - una procedura stub - che impacchetta i propri argomenti in un messaggio e li invia in rete ad un particolare processo del server. La procedura stub, sul versante client, poi si blocca e, nel frattempo, il server spacchetta il messaggio, chiama la procedura, impacchetta i risultati ottenuti in un messaggio e li trasmette al client stub che si sblocca, riceve il messaggio, spacchetta i risultati di RPC e li restituisce al chiamante. Questo impacchettamento di argomenti è talvolta chiamato **marshalling**.

In Windows 2000, il meccanismo RPC segue lo standard, ampiamente usato dall'ambiente di elaborazione distribuita, per i messaggi RPC in modo che i programmi scritti per RPC di Windows

2000 siano altamente portabili. Lo standard RPC è descritto in dettaglio e nasconde molte delle differenze di architettura dei calcolatori, quali le dimensioni dei numeri binari e l'ordine dei byte e dei bit nelle word di un calcolatore, specificando formati di dati standard per i messaggi RPC.

Windows 2000 può inviare messaggi RPC usando NetBIOS, o Winsock su reti TCP/IP, o named pipe su reti LAN Manager. La funzione LPC, discussa prima, è simile a quella RPC, a parte che in LPC i messaggi vengono passati fra due processi in funzione nello stesso computer.

È noioso e incline agli errori scrivere il codice per ordinare e trasmettere gli argomenti in formato standard, per poi metterli in disordine ed eseguire la procedura remota, riordinare ed inviare i risultati ottenuti, e poi metterli in disordine e restituirli al chiamante. Per fortuna, tuttavia, molto di questo codice può essere generato automaticamente mediante una semplice descrizione degli argomenti e dei risultati da restituire.

Windows 2000 mette a disposizione il **linguaggio di definizione delle interfacce di Microsoft** (Microsoft interface definition language) per descrivere i nomi, gli argomenti ed i risultati della procedura remota. Il compilatore di questo linguaggio genera le intestazioni dei file che dichiarano gli stub per le procedure remote e i tipi di dati per l'argomento e i messaggi che restituiscono i valori. Genera pure il codice sorgente per le procedure stub usate dalla parte del client e, dal lato server, un unmarshaller e un dispatcher. Quando l'applicazione è collegata, vengono incluse le procedure stub e quando l'applicazione esegue lo stub di RPC, il codice generato gestisce il resto.

**COM** è un meccanismo per la comunicazione tra processi che è stato sviluppato per Windows; gli oggetti COM forniscono un'interfaccia ben definita per manipolare i dati nell'oggetto. Windows 2000 possiede un'estensione chiamata **DCOM** che può essere usata in una rete che utilizza il meccanismo RPC per fornire un metodo trasparente di sviluppo per le applicazioni distribuite.

## 6.3 Redirezionatori e server

In Windows 2000, un'applicazione può usare le API di I/O di Windows 2000 per accedere ai file da un computer remoto, come se fossero locali, a patto che il computer remoto abbia un server MS-NET in funzione, come quello di cui è provvisto Windows 2000 o Windows for Workgroups. Un **redirezionatore** (redirector) è l'oggetto del lato client che inoltra le richieste di I/O ai file remoti che saranno soddisfatte da un server. I redirezionatori e i server funzionano in modalità kernel, per questioni di prestazioni e di sicurezza.

Più in dettaglio, l'accesso ad un file remoto avviene nel seguente modo:

- L'applicazione chiama il gestore di I/O per chiedere che un file venga aperto con un nome nel formato standard UNC.
- Il gestore di I/O costruisce un pacchetto di richiesta di I/O, come descritto nel Paragrafo 3.3.5.
- Il gestore di I/O riconosce che l'accesso è per un file remoto e chiama un driver che si chiama **fornitore multiplo della convenzione universale dei nomi** (multiple universal-naming-convention provider: MUP).
- MUP invia il pacchetto di richiesta di I/O in modo asincrono a tutti i redirezionatori registrati.
- Un redirezionatore, che può soddisfare la richiesta, risponde a MUP. Per evitare di rivolgere in futuro la stessa domanda a tutti i redirezionatori, MUP usa la cache per ricordare quale redirezionatore può maneggiare questo file.
- Il redirezionatore invia la richiesta di rete al sistema remoto.
- I driver di rete del sistema remoto ricevono la richiesta e la passano al driver del server.
- Il driver del server passa la richiesta al proprio driver locale del file system.
- Viene chiamato un device driver adatto per accedere ai dati.
- I risultati sono restituiti al driver del server, che invia di nuovo i dati al redirezionatore richiedente che restituisce, poi, i dati all'applicazione chiamante tramite il gestore di I/O.

Un processo simile si presenta per le applicazioni che usano l'API di rete di Win32, invece dei servizi UNC, salvo che viene usato un modulo chiamato router multiprovider, anziché MUP.

Per la portabilità, i redirezionatori e i server usano l'API TDI per trasporto di rete. Le richieste stesse sono espresse da un protocollo di livello più elevato, che per default è il protocollo SMB menzionato al Paragrafo 6.1. La lista dei redirezionatori è conservata nel database del registro del sistema.

## 6.4 Domini

Molti ambienti in rete hanno gruppi naturali di utenti, come gli studenti di un laboratorio di informatica in una scuola, o gli impiegati in un reparto commerciale. Spesso, si desidera che tutti i membri del gruppo possano accedere alle risorse condivise tra i vari computer del gruppo. Per gestire i diritti di accesso globali entro tali gruppi, Windows 2000 usa il dominio. In precedenza, questi domini non avevano alcuna relazione con il sistema di dominio del nome (DNS) che mappa i nomi dell'host di Internet negli indirizzi IP; ora, tuttavia, sono strettamente collegati: nello specifico, un dominio di Windows 2000 è un gruppo di workstation e server di Windows 2000 che condivide una politica comune di sicurezza e un database dell'utente. Siccome Windows 2000 usa attualmente il protocollo Kerberos per fiducia e autenticazione, un dominio di Windows 2000 è come il regno di Kerberos. Le precedenti versioni di NT usavano i controllori del dominio primari e di riserva; ora tutti i server di un dominio sono controllori del dominio. Inoltre, le versioni precedenti richiedevano la messa a punto di fiduciari unidirezionali fra i domini. Windows 2000 usa un approccio gerarchico basato su DNS e permette fiducia transitiva che può fluire verso l'alto o verso il basso nella gerarchia; questo metodo riduce il numero di fiducie richieste per  $n$  domini da  $n \sim (n - 1)$  a  $O(n)$ . Le workstation nel dominio si fidano del controllore del dominio nel dare informazioni corrette sui diritti di accesso di ogni utente, tramite il token di accesso dell'utente. Tutti gli utenti mantengono la capacità di limitare l'accesso alle proprie workstation, indipendentemente da ciò che possa affermare il controllore del dominio.

Poiché una ditta può avere molti reparti e una scuola molte classi, spesso è necessario gestire domini multipli all'interno di una singola organizzazione. Un **albero dei domini** (domain tree) è una gerarchia di nomi DNS contigua. 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 domini figlio. 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 può, tuttavia, essere costituita solo da un albero del dominio.

Le relazioni di fiducia possono essere regolate tra i domini in tre modi: unidirezionali, transitive e a collegamento incrociato. Le versioni di NT fino alla versione 4.0 permettevano solo l'installazione di fiducie unidirezionali. Una **fiducia unidirezionale** (one-way trust) è esattamente ciò che indica il nome: si dice che il settore A può fidarsi del settore B, ma che B non può fidarsi di A, a meno che non venga configurata un'altra relazione. In una **fiducia transitiva** (transitive trust), se A si fida di B e B si fida di C, allora A, B e C si fidano uno dell'altro poiché per default le fiducie transitive sono bidirezionali. Fiducie transitive sono abilitate per default per i nuovi domini in un albero e possono essere configurate solo fra domini all'interno di una foresta. Il terzo tipo: **fiducia a collegamento incrociato** (cross-link trust) è utile per ridurre il traffico di autenticazione. Supponiamo che i settori A e B siano vertici e che gli utenti di A usino spesso le risorse di B; se si usa una fiducia transitiva standard, le richieste di autenticazione devono attraversare fino all'antenato comune dei due vertici, ma se A e B hanno stabilito una fiducia a collegamento incrociato, le autenticazioni saranno inviate direttamente all'altro vertice.

## 6.5 Risoluzione dei nomi nelle reti TCP/IP

In una rete IP, la **risoluzione dei nomi** (name resolution) è il processo di conversione del nome di un computer in un indirizzo IP, tipo la risoluzione di [www.bell-labs.com](http://www.bell-labs.com) in 135.104.1.14. Windows 2000 fornisce parecchi metodi di risoluzione dei nomi, compreso il servizio dei nomi di Internet di Windows (WINS), la risoluzione dei nomi di broadcast, il sistema dei nomi dei domini (DNS), un file degli host ed un file LMHOSTS; la maggior parte di questi metodi è usata da molti sistemi operativi, ma nel seguito descriveremo solo WINS.

Nell'ambiente WINS, due o più server WINS mantengono un database dinamico dei collegamenti: nome, indirizzo IP e il software del client per interrogare i server; si usano almeno due server, in modo che il servizio WINS possa sopravvivere al guasto di un server e che il carico di lavoro di risoluzione del nome possa essere suddiviso tra macchine multiple.

WINS usa il protocollo dinamico di configurazione degli host (DHCP). DHCP aggiorna automaticamente le configurazioni dell'indirizzo nel database WINS, senza alcun intervento dell'utente o dell'amministratore, nel seguente modo: quando un client DHCP si avvia, trasmette un messaggio di scoperta e ogni server DHCP che riceve il messaggio risponde con un messaggio di offerta che contiene un indirizzo IP e le informazioni di configurazione, fornite in precedenza, per il client. Il client poi sceglie una delle configurazioni e manda un messaggio di richiesta al server DHCP selezionato; il server DHCP risponde con un indirizzo IP, con le informazioni di configurazione fornite precedentemente e con un **contratto d'affitto** (lease) per quell'indirizzo. Il contratto d'affitto dà al client il diritto di usare l'indirizzo IP per un periodo di tempo specificato. Quando il tempo del contratto d'affitto è trascorso per metà, il client cercherà di rinnovare il contratto per quell'indirizzo e se non viene rinnovato, il client dovrà ottenerne uno nuovo.

## 7 Interfaccia del programmatore

API di Win32 è l'interfaccia fondamentale per sfruttare le capacità di Windows 2000. Questo paragrafo descrive cinque aspetti principali delle API di Win32: 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 2000 fornisce molti servizi che possono essere usati dai programmi applicativi e tali servizi si ottengono manipolando gli oggetti del kernel. Un processo ottiene l'accesso ad un oggetto del kernel, detto XXX, chiamando la funzione `CreateXXX` per aprire un handle per XXX che è unico per il processo. A seconda di quale oggetto si sta aprendo, se la funzione di creazione non va a buon fine, può restituire 0, o una costante speciale detta `INVALID_HANDLE_VALUE`. Un processo può chiudere qualsiasi handle mediante la funzione `CloseHandle` e il sistema può cancellare l'oggetto se il numero di processi che usano l'oggetto cala a 0.

Windows 2000 fornisce tre modi per condividere gli oggetti fra i processi. Il primo modo consiste nell'ereditare un handle per l'oggetto dal processo figlio. Quando il padre chiama la funzione `CreateXXX`, esso fornisce una struttura di `SECURITY_ATTRIBUTES` con il campo `bInheritHandle` posto a `TRUE`; tale campo crea un handle ereditabile. Poi, il processo figlio può venire creato, passando il valore `TRUE` a `CreateProcess`, argomento della funzione `bInheritHandle`. La Figura 11 mostra un campione del codice che crea un handle semaforo che è ereditato da un processo figlio.

Supponiamo che il processo figlio conosca quali handle sono condivisi, il padre e il figlio possono realizzare la comunicazione tra i processi tramite gli oggetti condivisi. Nell'esempio di Figura 11, il processo figlio otterrà il valore dell'handle dal primo argomento della linea di comando, e potrà poi condividere il semaforo con il processo padre.

Il secondo modo di condivisione è che un processo dia un nome all'oggetto quando l'oggetto viene creato e che il secondo processo lo apra con quel nome. Questo metodo ha due limitazioni: la prima è che Windows 2000 non fornisce un modo per controllare se un oggetto, con il nome scelto, esiste già; la seconda limitazione è che lo spazio del nome dell'oggetto è globale, senza riguardo al tipo di oggetto. Per esempio, due applicazioni possono creare un oggetto chiamato *pipe* se si desiderano due oggetti distinti e possibilmente differenti.

Gli oggetti con nome presentano il vantaggio che i processi non collegati possono dividerli prontamente. Il primo processo chiamerà una delle funzioni `CreateXXX` e fornirà un nome nel parametro `lp.szName`; il secondo processo può ottenere un handle per condividere questo oggetto chiamando `OpenXXX` (o `CreateXXX`) con lo stesso nome, secondo quanto indicato nell'esempio di Figura 12.

Il terzo modo di condivisione degli oggetti avviene tramite la funzione `DuplicateHandle`: questo metodo richiede qualche altro metodo di comunicazione fra processi per passare l'handle duplicato. Se si assegna un handle ad un processo, e un valore dell'handle in quel processo, un secondo processo può ottenere un handle dello stesso oggetto e quindi dividerlo. Un esempio di questo metodo è indicato in Figura 13.

```

...
SECURITY_ATTRIBUTES sa;
sa.nlength = sizeof (sa);
sa.lpSecurityDescriptor = NULL;
sa.bInheritHandle = TRUE;
Handle a semaphore = CreateSemaphore(&sa,1,1,NULL);
char command line [132];
ostream
    ostring (command line, sizeof (riga di comando));
ostring << a semaphore << ends;
CreateProcess (" another process.exe",
    command line, NULL, NULL,TRUE, ...);
...

```

**Figura 11.** Codice di un processo figlio per condividere un oggetto ereditato da un handle.

```

//processo A
...
Handle a semaphore =
    CreateSemaphore (NULL, 1,.1, "MySEM1,");
...
//processo B
...
Handle b semaphore =
    OpenSemaphore (SEMAPHORE ALL ACCESS, FALSO, "MySEM1");
...

```

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

```

...
//Il processo A desidera dare al processo B
//accesso a un semaforo
//
//processo A
Handle a semaphore = CreateSemaphore (NULL, 1,1,NULL);
//invia il valore del semaforo al processo B
//usando un messaggio o la memoria condivisa...
//
//processo B
Handle process a =
    OpenProcess (PROCESS ALL ACCESS, FALSE, process id of A);
Handle b semaphore;
DuplicateHandle (process a,a semaphore,
    GetCurrentProcess (),&b semaphore,
    0, FALSE, DUPLICATE SAME ACCESS);
//usare semaforo b per accedere al semaforo
...

```

**Figura 13.** Codice per condividere un oggetto passando un handle.

## 7.2 Gestione dei processi

In Windows 2000, un processo è una istanza in esecuzione di un'applicazione, e un thread è un'unità di codice che può essere schedulata dal sistema operativo; di conseguenza, un processo contiene uno o più thread. Un processo viene iniziato quando qualche altro processo chiama la procedura `CreateProcess` che carica tutte le librerie di collegamento dinamico che sono usate dal processo e crea un **thread primario** (primary thread). Ulteriori thread possono essere creati dalla funzione `CreateThread`. Ogni thread viene creato con il proprio stack, che per default è di 1 MB, a meno che non venga diversamente specificato in un argomento di `CreateThread`. Poiché alcune funzioni run-time in C mantengono lo stato in variabili statiche, quali `errno`, un'applicazione multithread deve stare attenta ad un accesso non sincronizzato. La funzione wrapper `beginthreadex` fornisce la sincronizzazione adatta.

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

Le priorità nell'ambiente Win32 sono basate sul modello di schedulazione di Windows 2000, ma non si possono scegliere tutti i valori di priorità. Win32 usa 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 del processo non fosse in `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 si può cambiare con la funzione `SetPriorityClass`, o mediante il passaggio di un argomento al comando `START`. Per esempio, il comando `START/REALTIME cbsvr.exe` eseguirà il programma `cbsvr` in `REALTIME_PRIORITY_CLASS`. Si noti che soltanto gli utenti con il privilegio di priorità di schedulazione aumentata possono muovere un processo in `REALTIME_PRIORITY_CLASS`. Gli amministratori e gli utenti hanno questo privilegio per default.

Quando un utente fa funzionare un programma interattivo, il sistema deve fornire prestazioni particolarmente buone per quel processo; per questo motivo, Windows 2000 ha una regola speciale di schedulazione per i processi in `NORMAL_PRIORITY_CLASS`. Windows 2000 fa distinzione tra un processo in primo piano che è attualmente selezionato sullo schermo, e i processi in secondo piano che non sono attualmente selezionati. Quando un processo si muove verso il primo piano, Windows 2000 aumenta il quantum di schedulazione di un qualche fattore, tipicamente di 3. (Questo fattore può essere alterato tramite l'opzione prestazioni nella sezione opzioni avanzate del pannello di controllo). Tale incremento fornisce al processo in primo piano tre volte il tempo di funzionamento prima che accada una interruzione per condivisione del tempo.

Un thread incomincia con una priorità iniziale determinata dalla propria classe, ma tale priorità può essere alterata dalla funzione `SetThreadPriority`, che prende un argomento che specifica una priorità relativa rispetto a quella 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 che il kernel ha due classi di priorità: 16–31 per la classe real-time e 0–15 per la classe a priorità variabile. `THREAD_PRIORITY_IDLE` pone la priorità a 16 per i thread in real-time e a 1 per i thread a priorità variabile. `THREAD_PRIORITY_TIME_CRITICAL` pone la priorità a 31 per i thread in real-time e a 15 per i thread a priorità variabile.

Come è stato discusso nel Paragrafo 3.2, il kernel aggiusta dinamicamente la priorità di un thread a seconda che il thread sia I/O bound o CPU bound. L'API di Win32 fornisce un metodo per disabilitare questo aggiustamento, tramite le funzioni `SetProcessPriorityBoost` e `SetThreadPriorityBoost`.

Un thread può essere creato in uno **stato sospeso** (suspended state): il thread non funzionerà finché un altro thread non lo renderà eleggibile tramite la funzione `ResumeThread`; la funzione `SuspendThread` opera in modo opposto. Queste funzioni regolano un contatore, in modo che se un thread viene sospeso due volte, esso deve essere ripreso due volte prima che di poter funzionare.

Per sincronizzare l'accesso concorrente agli oggetti condivisi dai thread, il kernel fornisce oggetti di sincronizzazione, quali i semafori e i mutex. Inoltre, la sincronizzazione dei thread può essere realizzata dalle funzioni `WaitForSingleObject` o `WaitForMultipleObjects`. Un altro metodo di sincronizzazione nelle API di Win32 è la sezione critica che è una regione sincronizzata di 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, solo ad un thread alla volta sarà permesso di continuare e gli altri aspetteranno nella procedura `EnterCriticalSection`. Il meccanismo della sezione critica è un po' più veloce degli oggetti di sincronizzazione del kernel.

Una **fibra** è codice in modalità utente che viene schedulato in base ad un algoritmo di schedulazione definito dall'utente. Un processo può avere fibre multiple, così come thread multipli. La differenza maggiore tra thread e fibre consiste nel fatto che i thread possono essere in esecuzione in modo concorrente, ma è consentita l'esecuzione solo di una fibra per volta, anche se l'hardware è multiprocessore. Questo meccanismo è incluso in Windows 2000 per facilitare la portabilità delle applicazioni compatibili con UNIX e che sono state scritte per un modello di esecuzione su fibra.

Il sistema crea una fibra chiamando `ConvertThreadToFiber` o `CreateFiber`. La differenza principale fra queste funzioni è che `CreateFiber` non comincia l'esecuzione della fibra che è stata creata; per iniziare l'esecuzione, l'applicazione deve chiamare `SwitchToFiber`, mentre può terminare l'esecuzione chiamando `DeleteFiber`.

### 7.3 Comunicazione tra processi

Un modo in cui le applicazioni di Win32 possono eseguire la comunicazione tra processi è di condividere gli oggetti del kernel; un altro modo è tramite il passaggio di messaggi: metodo che è particolarmente popolare per le applicazioni 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* (*posting*) e *inviare* (*sending*) un messaggio è che le procedure di spedizione sono asincrone: restituiscono immediatamente il risultato; il thread chiamante non sa quando il messaggio verrà realmente consegnato. Le procedure di invio sono sincrone: bloccano il chiamante finché il messaggio non è stato consegnato ed elaborato.

In aggiunta all'invio di un messaggio, un thread può anche inviare dei dati con il messaggio. Siccome i processi hanno spazi di 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 viene inviato, Windows 2000 copia i dati in un nuovo blocco di memoria e assegna l'indirizzo virtuale del nuovo blocco al processo ricevente.

A differenza dell'ambiente a 16 bit di Windows, 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 nelle finestre a 16 bit, poiché, con code separate, un'applicazione bloccata non può bloccare l'input ad altre applicazioni. Se un'applicazione Win32 non chiama GetMessage per gestire gli eventi nella propria coda di input, la coda si riempirà completamente e dopo circa 5 secondi il sistema contrassegnerà l'applicazione come "non rispondente."

## 7.4 Gestione della memoria

L'API di Win32 fornisce ad un'applicazione parecchi modi per usare la memoria: memoria virtuale, file a memoria mappata, heap e memoria locale del thread.

Un'applicazione chiama VirtualAlloc per riservare o coinvolgere la memoria virtuale e VirtualFree per eliminare il coinvolgimento o rilasciare la memoria. Queste funzioni permettono all'applicazione di specificare l'indirizzo virtuale in cui la memoria è allocata; esse operano su multipli della dimensione della pagina di memoria e l'indirizzo di partenza di una regione allocata deve essere maggiore di 0x10000. Esempi di queste funzioni appaiono in Figura 14. Un processo può bloccare alcune delle proprie pagine coinvolte nella memoria fisica chiamando VirtualLock; il numero massimo di pagine che un processo può bloccare è 30, a meno che il processo non chiami prima SetProcessWorkingSetSize per aumentare la massima dimensione del working-set.

```
...
//alloca 16 MB in cima al nostro spazio di indirizzo
void * buf =
    VirtualAlloc (0,0x1000000, MEM RESERVE/ MEM TOPDOWN,
        PAGE READWRITE);
//coinvolge 8 MB superiori dello spazio assegnato
VirtualAlloc (buf + 0x800000,0x800000,
    MEMCOMMIT, PAGE,READWRITE);
//esegue qualcosa
//elimina il coinvolgimento
VirtualFree (buf + 0x800000,0x800000, MEM DECOMMIT);
//rilascia tutto lo spazio di indirizzo allocato
VirtualFree (buf, 0, MEM RELEASE);
...
```

**Figura 14.** Frammenti di codice per allocare memoria virtuale.

Un altro modo di usare la memoria da parte di un'applicazione, consiste nella mappatura in memoria di un file nel proprio spazio di indirizzi. La mappatura della memoria è anche un modo conveniente per due processi di condividere la memoria: 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 proprio per condividere una regione di memoria con un altro processo, non c'è bisogno di alcun file. Il processo può chiamare CreateFileMapping con un handle 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 il file mappando una dimensione di 8 MB
HANDLE hmap = CreateFileMapping (hfile, PAGE_READWRITE,
    SEC_COMMIT, 0,0x800000, "SHM 1");
//dare uno sguardo allo spazio mappato
void *buf = MapViewOfFile (hmap, FILE_MAP_ALL_ACCESS,
    0,0,0x800000);
//fare qualcosa con esso
//demappare il file
UnmapViewOfFile (buf);
CloseHandle (hmap);
CloseHandle (hfile);
...

```

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

Il terzo modo di usare la memoria da parte delle applicazioni è lo heap che, nell'ambiente Win32, è proprio una regione di spazio di indirizzi riservati. Quando un processo Win32 è inizializzato, viene creato un **heap di default** (default heap) di 1 MB, e poiché molte funzioni di Win32 usano lo heap di default, l'accesso allo heap è sincronizzato per proteggere la propria allocazione dello spazio dai danni di aggiornamenti concorrenti da parte di thread multipli. Win32 fornisce parecchie funzioni di gestione dello heap in modo da potere allocare e gestire da parte di un processo uno heap privato; queste funzioni sono: HeapCreate, HeapAlloc, HeapRealloc, HeapSize, HeapFree e HeapDestroy. L'API di Win32 fornisce pure funzioni di HeapLock e HeapUnlock per permettere ad un thread di ottenere accesso esclusivo ad un heap. A differenza di VirtualLock, queste funzioni permettono solo la sincronizzazione e non bloccano le pagine in memoria fisica.

Il quarto modo di usare la memoria da parte delle applicazioni è un meccanismo di memorizzazione locale dei thread. Le funzioni che fanno affidamento su dati globali o statici non lavorano correttamente in un ambiente multithread. Per esempio, la funzione run-time del C: strtok usa una variabile statica per tenere traccia della propria posizione mentre analizza una stringa. Affinchè due thread concorrenti eseguano strtok correttamente, hanno bisogno di variabili separate della *posizione corrente*. Il meccanismo di memorizzazione locale dei thread assegna memoria globale sulla base del singolo thread. Fornisce sia metodi dinamici che statici di creazione della memoria del thread locale. Il metodo dinamico è illustrato in Figura 16. Per usare una variabile statica di un thread locale, l'applicazione dovrà dichiarare la variabile nel modo seguente per assicurarsi che ogni thread abbia la propria copia privata: declspec (thread) DWORD cur pos = 0.

```

//riserva una posizione per una variabile
DWORD var index = TlsAlloc ();
//la pone ad un certo valore
TlsSetValue (var index, 10);
//recupera il valore
int var = TlsGetValue (var index);
//rilascia l'indice
TlsFree (var index);

```

**Figura 16.** Codice per la memorizzazione dinamica locale del thread.

## 8 Sommario

Microsoft ha progettato Windows 2000 per essere un sistema operativo estensibile e portabile e in grado di trarre vantaggio dalle nuove tecniche e dall'hardware. Windows 2000 supporta ambienti operativi multipli ed elaborazione a multiprocessore simmetrica. L'uso degli oggetti del kernel per fornire servizi di base e di supporto all'elaborazione client-server, permette a Windows 2000 di supportare un'ampia varietà di ambienti applicativi. Per esempio, Windows 2000 può far funzionare programmi compilati per MS-DOS, Win16, Windows 95, Windows 2000 e/o POSIX. Fornisce memoria virtuale, cache integrata e schedulazione preemptive. Windows 2000 supporta un modello di sicurezza più elevato dei precedenti sistemi operativi Microsoft ed include caratteristiche di internazionalizzazione. Windows 2000 funziona su un'ampia varietà di computer, in modo che gli utenti possano scegliere ed aggiornare l'hardware per venire incontro alle loro possibilità economiche e alle richieste di prestazioni, senza avere bisogno di alterare le applicazioni che utilizzano.

## Esercizi

- 1 Quali sono alcune delle motivazioni che, a causa dello spostamento del codice grafico di Windows NT dalla modalità utente alla modalità kernel, possono far diminuire l'affidabilità del sistema? Quale degli obiettivi originali di progetto di Windows NT vengono violati da questa degradazione?
- 2 Il gestore di Windows 2000 VM usa un processo a due stadi per allocare la memoria. Individuare parecchi vantaggi di questo metodo.
- 3 Discutere alcuni vantaggi e svantaggi della particolare struttura della tabella di pagina usata da Windows 2000.
- 4 Quale è il numero massimo di fault di pagina che potrebbero presentarsi nell'accesso ad: (a) un indirizzo virtuale, (b) un indirizzo virtuale condiviso? Quale meccanismo hardware viene utilizzato dalla maggior parte dei processori per fare diminuire tali numeri?
- 5 Quale è lo scopo di un'ingresso nella tabella di pagina di Windows 2000?
- 6 Quali sono i passi che il gestore di cache deve compiere per copiare i dati dentro e fuori la cache?
- 7 Quali sono i principali problemi che si generano facendo funzionare applicazioni Windows a 16 bit in una VDM? Identificare le soluzioni scelte da Windows 2000 per ognuno di questi problemi. Per ogni soluzione, indicare almeno una limitazione.
- 8 Quali cambiamenti sarebbero necessari a Windows 2000 per far funzionare un processo che usa uno spazio di indirizzo a 64 bit?
- 9 Windows 2000 ha un gestore centralizzato di cache. Quali sono i vantaggi e gli svantaggi di questo gestore della cache?
- 10 Windows 2000 usa un sistema I/O guidato dai pacchetti. Discutere i pro e i contro di questo metodo.
- 11 Si consideri un database della memoria principale di 1 terabyte. Quale meccanismo di Windows 2000 si potrebbe usare per accedere al database?

## **Note bibliografiche**

Solomon e Russinovich [2000] danno una descrizione di Windows 2000 e notevoli dettagli tecnici riguardanti la struttura interna e i componenti del sistema. Tate [2000] è un buon riferimento sull'uso di Windows 2000. Microsoft Windows 2000 Server Resource Kit (Microsoft [2000b]) è costituito da sei volumi rivolti all'uso ed alle spiegazioni di Windows 2000. Microsoft Developer Network Library (Microsoft [2000a]) viene pubblicato trimestralmente e fornisce molte informazioni su Windows 2000 e su altri prodotti Microsoft. Iseminger [2000] è un buon riferimento sul direttorio attivo di Windows 2000. Richter [1997] fornisce una discussione dettagliata sulla scrittura di programmi che usano l'API di Win32. Silberschatz et al. [2001] contiene una buona discussione sugli alberi B+.