Operating Systems
A Modern Perspective
Second Edition
Gary Nutt

FEDEX-1008

Acquisitions Editor: *Maite Suarez-Rivas*
Assistant Editor: *Jason Miranda*
Project Manager: *Trillium Project Management*
Production Assistant: *Tracy Treeful*
Composition: *Michael and Sigrid Wile*
Text Design: *Delgado Design, Inc.*
Cover Illustration: *Susan Cyr*
Cover Design: *Lynne Reed*
Copyeditor: *Stephanie Argeros-Magean*
Proofreader: *Trillium Project Management*

Access the latest information about Addison-Wesley titles from our World Wide Web site:
www.awlonline.com

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and Addison-Wesley was aware of a trademark claim, the designations have been printed in initial caps or all caps.

The programs and applications presented in this book have been included for their instructional value. They have been tested with care but are not guaranteed for any purpose. The publisher and author do not offer any warranties or representations, nor do they accept any liabilities with respect to the programs or applications.

"CHANGES" Words and Music by David Bowie
© 1971 EMI MUSIC PUBLISHING LTD., London WC2H0EA, TINTORETTO MUSIC and MOTH MUSIC. All Rights for EMI MUSIC PUBLISHING LTD. Controlled and Administered by SCREEN GEMS-EMI MUSIC INC. Reproduced by Permission of IMP LTD. All Rights to TINTORETTO MUSIC Administered by RZO MUSIC. All Rights for MOTH MUSIC Administered by CHRYSALIS SONGS. All Rights Reserved. International Copyright Secured. Used by Permission.

"OVERS" Words and Music by Paul Simon
© 1967 Paul Simon. Used by Permission of the Publisher, Paul Simon Music.

"TERRENCE, THIS IS STUPID STUFF" by A. E. Housman
Used by Permission of The Society of Authors as the Literary Representative of the Estate of A. E. Housman.

This book was typeset in Quark 3.3 on a Macintosh G3. The font families used were Times, ITC Officina, and Prestige Elite. It was printed on New Era Matte.

LIBRARY OF CONGRESS CATALOG CARD NUMBER 99-073717
ISBN 0-201-61251-8

Copyright © 2000 by Addison Wesley Longman, Inc.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the publisher. Printed in the United States of America.

2 3 4 5 6 7 8 9 10-MA-03020100

I/O Manager to read byte i and the next K bytes into the buffer. Then when the thread requests byte $i + 1, i + 2, \ldots, i + K$, those bytes will have already been read, so the thread need not wait for a device operation to complete. The write-behind strategy works similarly.

18.6.4.1 ■ Virtual Memory Manager

Windows NT is a *paging virtual memory system.* When a process is created, it is given 4 GB of virtual addresses, though none of the addresses are actually allocated at that time. When the process needs space, it first *reserves* as much of the address space as it needs at that moment; reserved addresses do not cause any actual space to be allocated; rather virtual addresses are reserved for later use. When the process needs to use the virtual addresses to store information, it *commits* the address space, meaning that some system storage space is then allocated to the process to hold information. Ordinarily, a commit operation causes space on the disk (in the process's *page file*) to be allocated to the process; the information is stored on the disk until it is actually referenced by a thread.

Like all paging virtual memory mechanisms, when an executing thread references a virtual address, the Virtual Memory Manager ensures that the page containing that virtual address is read from the page file and placed at some system-defined location in the physical executable memory. The Virtual Memory Manager *maps* the virtual address referenced by the thread into the physical memory address where the information is loaded.

The Virtual Memory Manager has been designed so that a large portion of each process's address space (usually half of it, though different configurations of Windows NT use different fractions) is mapped to the information used by the system when it is in supervisor mode (see Figure 18.13). There are a few important implications of this decision, as follows:

- A process can directly reference (but not necessarily access) every location in the system space.
- Every process shares the same view of the system's space.
- Such a large, shared virtual address space makes memory-mapped files feasible.

In Figure 18.13, when a thread references an address in the user space, the virtual memory system loads the target location into the physical memory prior to its use so that the thread can read or write the virtual memory address by referencing a physical memory address. The same mapping takes place for OS memory references, though these references are protected, and every process's OS addresses map to the OS memory rather than to the application-specific part of the address space.

18.6.4.2 ■ I/O Manager

The I/O Manager is responsible for handling all the input/output operations to every device in the system. Its operation can be quite complex (Solomon, 1998).

- The I/O Manager creates an abstraction of all device I/O operations on the system so that the system's clients can perform operations on a common data structure.

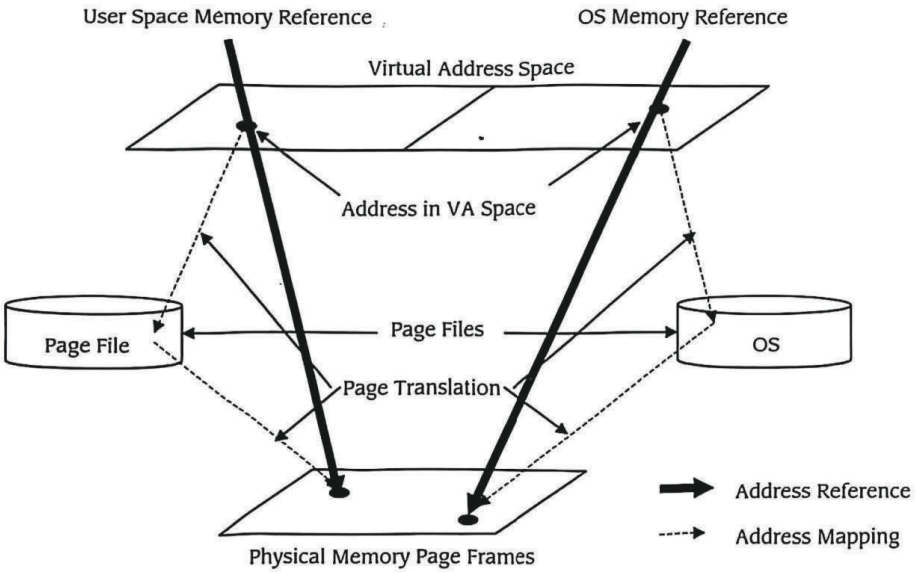

Figure 18.13
Virtual Memory

- The client can perform synchronous and asynchronous I/O.
- The client can invoke the Security Reference Monitor whenever security is an issue.
- The I/O Manager must accommodate device drivers written in a high-level language by third parties. Those drivers must be able to execute in supervisor mode. Installation and removal of a device driver must be dynamic.
- The I/O Manager can accommodate alternative file systems on the system's disks. This means that some file systems might use the MS-DOS format, others might use an industry standard CD-ROM format, and yet others might use Windows NT's own file system (NTFS).
- I/O Manager extensions—device drivers and/or file systems—must be consistent with the memory-mapped file mechanism implemented in the Virtual Memory Manager, so extension designs are constrained by the facilities provided by the manager.

The I/O Manager is made up of the following components, as shown in Figure 18.14 (Nagar, 1997).

- *Device drivers* are at the lowest level. They manipulate the physical I/O devices. These drivers are described generically in Chapter 5.
- *Intermediate drivers* are software modules that work with a low-level device driver to provide enhanced service. For example, a low-level device driver might simply pass an error condition "upward" when it detects it, while an intermediate driver might receive the error and decide to issue a retry operation to the lower-level driver.

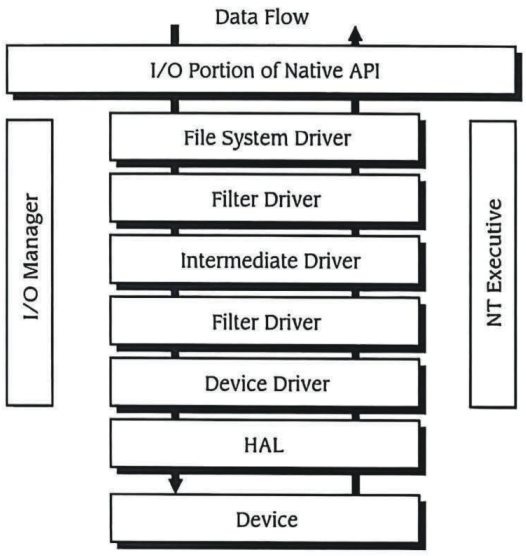

Figure 18.14
The I/O Manager

- *File system drivers* extend the functionality of the lower-level drivers (such as intermediate and device drivers) to implement the target file system.
- A *filter driver* can be inserted between a device driver and an intermediate driver, between an intermediate driver and a file system driver, or between the file system driver and the I/O Manager API to perform any kind of function that might be desired. For example, a network redirector filter driver can intercept file commands intended for remote files and redirect them to remote file servers.

Drivers are the single component that can be added to the NT Executive to run in supervisor mode. The OS has not been designed to support third-party software, other than drivers, that want to add supervisor mode functionality.

The Windows NT I/O Manager defines the framework in which device drivers, intermediate drivers, file system drivers, and filter drivers are dynamically added to and removed from the system and are made to work together. The basic idea of a stream of modules between the API and device was first used in the AT&T System V UNIX I/O streams (Ritchie, 1984). The dynamic stream design allows one to easily configure complex I/O systems; the System V streams were the basis of the network protocol implementations.

Also similar to System V streams, the I/O Manager directs modules by issuing *I/O request packets* (IRPs) into a stream. If the IRP is intended for a particular module, that module responds to the IRP; otherwise, it passes the IRP to the next module in the stream. Each driver in the stream has the responsibility of accepting IRPs, either reacting to the IRP if it is directed at the driver or passing it on to the next module if it is not.

All information read from or written to the device is managed as a stream of bytes called a *virtual file*. Every driver is written to read and/or write a virtual file. Low-level

device drivers transform information read from the device into a stream and transform stream information into a device-dependent format before writing it.

18.6.4.3 ■ The Native API

While the NT Executive and Kernel are designed and programmed as separate modules, they are combined with the `kernel` executable when Windows NT is built. The combined NT Executive and Kernel module (with the underlying HAL) implements the full NT OS. In Version 5.0, `the kernel` exports about 240 functions (Russinovich, 1998), most of which are *undocumented,* meaning that only subsystem developers should base their software on the functions.

18.6.5 ■ NT Subsystems

The Windows NT system software is constructed as a layered architecture. Layer i is constructed using the services provided by layer $i - 1$ ("at the layer $i - 1$ interface"), creating its own services and exporting them through its own (layer i) interface. In the Windows NT architecture, *subsystems* provide a layer of service above the Native API. There can be many different subsystems, some related, but others independent of one another, as functionality is added to the computer system. For example, a typical Windows NT system includes the following:

- The Win32 subsystem
- The WinLogon service (to authenticate users, using the LSA server and Secure Reference Monitor described in the NT Executive section, when they begin to use the system)
- A remote procedure call service

It may also include a Win16 subsystem. If the Windows NT machine were required to support POSIX application programs, a POSIX subsystem could be added as a component in the subsystem layer.

Each subsystem uses the Native API to provide the services it implements. The *environment subsystems* behave as a traditional interior layer. In the layered architecture approach, they use the Native API, add functionality and services, and then export their own API. In the Microsoft strategy, subsystem APIs are documented APIs, meaning that a programmer can write new software at the next higher-level layer and be assured that the API will be unchanged when implementations at a lower-level layer in the architecture are changed.

Figure 18.15 shows how this layering works in Windows NT. The *Win32 subsystem* exports a documented interface, the *Win32 API,* as a set of about 2000 functions. The Win32 API is a documented interface. An application programmer can write software above the Win32 subsystem that calls the API functions to accomplish an application-specific task.

The Win32 subsystem also provides one other type of service: It implements a user interface management system, since the NT Executive/Kernel does not have one of these. This is primarily a matter of practicality—when the system begins to run, some part of the system software needs to read the keyboard and mouse and manage the display. Rather than have each environment subsystem provide its own user interface, the