Volume 5, Number 5 May 1987

IBM Operating System/2

IBM Personal System/2™ Seminar Proceedings

The Publication for Independent Developers of Products for IBM Personal System/2

Published by International Business Machines Corporation Entry Systems Division

Contents

Foreword

IBM Personal System/2™ Seminars and Proceedings provide information about new product announcements and enhancements to existing products, and are intended to assist independent developers in their hardware and software development efforts.

Over the past several years, the success of the IBM Personal Computer family was due in part to the efforts of independent developers, whose hardware and software products have become widely used. For its part, IBM helped these vendors by holding relevant technical seminars and publishing the proceedings of those seminars. The result was a mutually beneficial partnership and transfer of technical knowledge.

With the advent of the Personal System/2 family, IBM's seminar program will continue. Through these seminars and the corresponding proceedings, IBM will address the independent developers' need for technical information about the latest IBM products. In these and future proceedings, you will find technical information about subjects such as:

- IBM computer design and architecture
- IBM computer components and their interaction
- Memory capacities, speeds, transfer rates
- Input/output device capacities, speeds, access methods and rates
- Graphics and display technologies, programming considerations
- Printing technologies, programming considerations
- Operating system high level interfaces
- Development tools: capabilities, languages, program verification aids
- Compatibility considerations
- Communications: capabilities, offerings, statistics
- Enhancements to existing IBM hardware and software products
- Hints, tips and techniques to enhance your productivity

Through these seminars and proceedings, IBM intends to maintain its partnership with independent developers and assist them in successfully producing hardware and software products for the IBM Personal System/2 family.

Personal System/2 is a trademark of the IBM Corporation.

IBM Operating System/2™ Overview

Offerings

IBM Operating System/2™ is a new generation of IBM operating system providing new and expanded function for both the end user and the application developer. IBM Operating System/2 consists of two offerings:

- The IBM Operating System/2 Standard Edition is a multitasking operating system that breaks the 640 KB memory barrier, provides greater flexibility to run multiple applications concurrently and has a Presentation Manager for graphics and windowing functions.
- The Operating System/2 Extended Edition provides a Communications Manager that supports a broad range of communications capabilities and a Database Manager that supports the IBM relational data base model, in addition to the operating system function and Presentation Manager provided by the Standard Edition.

Highlights of Operating System/2

- 16 MB Addressable Random Access Memory Support
- Concurrent Processing of Multiple Applications
- High-Level Programming Interface
- Presentation Manager
- Enhanced Ease of Use Facilities
- Compatibility with DOS 3.30
- Communications Manager (Extended Edition only)
- Database Manager (Extended Edition only)
- Systems Application Architecture
- New Service and Warranty
- National Language Support

16 MB Addressable Random Access Memory Support

Operating System/2 supports up to 16 MB of addressable random access memory, which enables application developers to take full advantage of memory beyond 640 KB for applications and data. End users will have the benefit of larger and functionally richer applications that can process larger amounts of data such as spreadsheets and large documents.

Operating System/2 implements virtual memory through segment swapping. An Operating System/2 application program can be larger than available real memory. Actual size is dependent on program characteristics and the physical capacity of the system. Operating System/2 manages physical memory by swapping memory segments to a disk file as required.

Concurrent Processing of Multiple Applications

New applications written to take advantage of the function provided in Operating System/2 may be run and displayed concurrently for the convenience and increased productivity of the end user. Switching between applications is fast and simple. Productivity can increase because time previously spent starting, stopping, entering and exiting applications in a single application environment can now be spent more productively actually processing data. Applications also can be written without knowledge of what other applications will coexist.

High-Level Programming Interface

IBM is addressing application development productivity and future compatibility by providing application developers with a high-level CALL interface to Operating System/2. Applications written to this interface will be compatible with successive versions of Operating System/2. Applications also can take advantage of a high level of device independence. This assists in upgrading to new IBM Personal System/2 systems or to new versions of Operating System/2.

Application developers may choose to use a subset of the Operating System/2 CALL interface. This subset allows the same program to run under DOS 3.30 and under Operating System/2.

Operating System/2 is a trademark of the IBM Corporation

Presentation Manager

The Presentation Manager provides windowing and graphics functions along with an application enabling interface which makes it easy to write applications that support the common user access of the Systems Application Architecture.

Windowing allows multiple applications to be viewed by the end user at the same time. Each application can support multiple windows. The user can control window size and position, and an application can create or delete windows. A clipboard function is provided to enable the user or application to extract data from one window and move it to another window or from one application to another.

Graphics support enables the development of a broad range of applications which take advantage of the supported all-points- addressable (APA) devices. There is a wide range of vector graphics, raster operations and extensive font support, including multiple font styles and sizes. Graphics orders can be stored, enabling the application to manage the picture data and facilitates fast redrawing of pictures.

Enhanced Ease-of-Use Facilities

Enhanced ease-of-use facilities in Operating System/2 make it easier for the novice to learn and simpler for both the novice and the experienced user to operate. Operating System/2 provides comprehensive Help information and descriptive written system messages. Much of this Help information is contextual in nature, giving the end user the information needed to complete the task at hand. For beginners, a tutorial assists in getting started by showing how to perform basic operating system tasks.

Operating System/2 also provides an interface to the system commands that support the IBM Systems Application Architecture conventions for Common User Access. Through this interface, applications can be started and stopped or added and deleted from the system as required. The size and position of the application windows also can be controlled. The user can control local print out on a printer or a plotter. Also, information and data contained in the Operating System/2 file system can be accessed.

The Extended Edition can be tailored by installing an individually selected mixture of Communications Manager and Database Manager.

Compatibility with DOS

Operating System/2 provides a DOS environment that allows many existing DOS applications to run unchanged. This assists in the transition from DOS to Operating System/2.

Applications which may not run in the DOS compatibility environment include time-dependent programs such as communications and real-time applications, hardware specific routines, such as device drivers, and network-dependent applications.

Operating System/2 uses many commands from the basic set of DOS commands. This also assists in the transition from DOS to Operating System/2. The user familiar with DOS commands need only learn the Operating System/2 commands and concepts that support Operating System/2 extended capabilities.

With the new Programmer Toolkit it will be possible to create applications that run under both DOS and Operating System/2 using a subset of the full capability of Operating System/2.

Files created by the user under either DOS 3.30 or Operating System/2 are interchangeable. This also assists in the transition from DOS to Operating System/2.

Communications Manager (Extended Edition only)

The Communications Manager provides a wide range of concurrent connectivities and protocols, concurrent emulation of multiple terminal types, file transfer under terminal emulation and Communications and Systems Management (C&SM) support.

Included in the Communications Manager are the following:

- Wide range of connectivities which can be used concurrently
  - SDLC
  - DFT
  - IBM Token-Ring Network
  - IBM PC Network
  - Asynchronous
- Multiple protocols which can be used concurrently
  - LU6.2
  - IBM 3270 Data Stream
  - Asynchronous
- Concurrent emulation of multiple terminal types

  - IBM 3270
  - IBM 3101 or DEC[1] VT100
- File transfer under terminal emulation
- Communications and Systems Management support (C&SM)
  - Communications and system management alerts for SDLC, asynchronous, IBM Token-Ring and PC Network data links
  - Problem determination data
- Programming interfaces
  - Advanced Program to Program Communications (APPC)
  - Server-Requester Programming Interface (SRPI) for Enhanced Connectivity Facilities (ECF)
  - Asynchronous Communications Device Interface (ACDI)
  - IBM NETBIOS
  - IEEE 802.2

Database Manager (Extended Edition only)

The Database Manager supports the IBM relational data base model, which provides a data structure in simple, tabular form. Data definition, retrieval, update and control operations are supported by the Structured Query Language (SQL). The SQL is a high-level data language available to end users interactively and through application programs written in IBM C/2. This SQL is consistent with the IBM family of relational data base products, DB2, SQL/DS, QMF and the IBM Systems Application Architecture, thus enabling the importing and exporting of data from various IBM sources.

Application creation tools, such as customizing display forms, menus and procedures enable the end user to develop a complete data base application without the need for programming. Facilities are available to the end user for data entry, data edit, query and report writing. Interchange with specific PC file formats provides the capability of importing and exporting data to the Database Manager. The Database Manager provides record level locking and data recovery in the event of application, system or media failure.

Systems Application Architecture

Operating System/2 is a participant in IBM Systems Application Architecture, a collection of selected software interfaces, conventions and protocols whose initial set of specifications is planned to be published in 1987. IBM Systems Application Architecture is to be the framework for development of consistent applications across the future offerings of the major IBM computing environments - System/370, System/3x and the IBM Personal Computers.

IBM Systems Application Architecture consists of four related elements, two of which are new (Common User Access and Common Programming Interface), plus extensions to today's existing communications architectures (Common Communications Support), and finally, IBM developed applications consistent across IBM systems (Common Applications).

Operating System/2 participates in the following elements of IBM Systems Application Architecture:

- Common User Access
- Common Programming Interface
  - Presentation Interface
  - Dialog Interface
  - Query Interface
  - Database Interface
  - COBOL
  - FORTRAN
  - C
- Common Communications Support

The following components of Operating System/2 support these elements of IBM Systems Application Architecture:

- Presentation Manager
- Query Manager
- Database Manager
- Communications Manager
- IBM COBOL/2
- IBM FORTRAN/2
- IBM C/2

1 Trademark of Digital Equipment Corporation.

New Service and Warranty

For the first time on an IBM personal computing operating system, IBM is introducing service and limited warranty for Operating System/2. In addition to the media warranty, IBM is providing a three month program warranty that includes replacement, correction, or refund. Central Service will be available for IBM Operating System/2 until the service expiration date, which will be published at availability.

National Language Support

Operating System/2 facilitates the translation of machine readable information, such as panels and messages, into various national languages, and it also contains support for many national keyboards and country conventions such as date and time.

Operating System/2 Phased Release

Operating System/2 will be released in several phases to enable users to begin taking advantage of the new enhanced operating system function.

Operating System/2 Standard Edition Version 1.0

Operating System/2 Standard Edition Version 1.0 is an early release of Operating System/2 Standard Edition Version 1.1 and will be replaced by version 1.1 when it becomes available.

Operating System/2 Standard Edition Version 1.0 provides all the functions of the new enhanced operating system with the exception of the graphics and windowing functions of the Presentation Manager and the Database and Communications Managers of the Extended Edition.

Operating System/2 Standard Edition Version 1.1

Operating System/2 Standard Edition Version 1.1 provides the new enhanced operating system functions, including the graphics and windowing functions of the Presentation Manager. It does not contain the Data Base or Communications Managers. Operating System/2 Standard Edition Version 1.1 will continue as a separate product from Operating System/2 Extended Edition and will satisfy the requirements of users who do not need the advanced functions of Operating System/2 Extended Edition.

Operating System/2 Extended Edition

Operating System/2 Extended Edition is the comprehensive operating system consisting of Operating System/2 Standard Edition Version 1.1 and the additional functions of the Communications Manager and the Database Manager.

Application Enablers

Operating System/2 Programmer Toolkit

The IBM Operating System/2 Standard Edition Programmer Toolkit contains significant programmer productivity functions for the creation of Operating System/2 and Family applications. The Operating System/2 Standard Edition Programmer Toolkit Version 1.0 contains the tools described below, except for the Presentation Manager tools. The Operating System/2 Standard Edition Programmer Toolkit Version 1.1 contains all the tools. Although the Linker is packaged with Operating System/2, Linker information is contained in the Operating System/2 Technical Reference and Operating System/2 Standard Edition Programmer Toolkit publications.

The Operating System/2 Standard Edition Programmer Toolkit contains the following major components:

- Presentation Manager tools. Used to create graphics and screen windows for interaction between application and end user.
- Link Related Functions. Includes the Operating System/2 Linker description, Import Librarian (IMPLIB) for creating import libraries, and the appropriate tools (BIND utility and Family Application Program Interface library) to create a Family application.
- Sample Programs. Used to illustrate Operating System/2 programming features and Operating System/2 Standard Edition Programmer Toolkit utilities.
- Programming Aids for IBM Macro Assembler/2 and IBM C/2 Languages. Contains files of macro libraries, declarations and error equates which enhance programmer productivity.
- Message Preparation Utilities. Used to:
  - Convert a source message file to an indexed file accessible by an application.

  - Bind messages to an executable module for fast access.
- Publications.

The following Presentation Manager components support windowing and graphics:

- Dialog Editor. Used to design a dialog box on the display screen. The dialog box is a pop-up or child window that contains one or more window controls. An application can use a dialog box to prompt a user for additional information about a current command selection.
- Icon Editor. Used to create icons, cursors and bitmaps that are not predefined in the Operating System/2 Standard Edition or the Operating System/2 Standard Edition Programmer Toolkit. The size of the icon or cursor can be changed.
- Font Editor. Used to create application font files. An Operating System/2 application can use a maximum of four fonts.
- Include Files. Used when compiling programs that need windowing capability.
- Resource Compiler. Used to compile resources (such as icons, cursors, menus and dialog box templates) and place them in the applications resource file.

Operating System/2 Technical Reference

The Operating System/2 Standard Edition Technical Reference contains technical information which supplements the Operating System/2 and Operating System/2 Standard Edition Programmer Toolkit publications. The Technical Reference is available separately. It is not included in the Operating System/2 Standard Edition Programmer Toolkit, but it is a prerequisite for understanding some topics discussed in the Toolkit publications.

The Technical Reference provides the technical information required for programming an application for Operating System/2 Standard Edition, including the Operating System/2 Standard Edition CALL interface. It is arranged in typical reference format for easy retrieval of information.

Languages for Operating System/2 and DOS 3.30

IBM provides the following language versions to support both DOS 3.30 and Operating System/2:

- IBM COBOL/2
- IBM FORTRAN/2
- IBM C/2
- IBM Macro Assembler/2
- IBM Pascal Compiler/2
- IBM BASIC Compiler/2

Each language is upwardly compatible with its previous version and provides extensive facilities for application developers to take advantage of the capabilities of all members of the IBM Personal Computer family. COBOL/2, FORTRAN/2 and C/2 are participants in the IBM Systems Application Architecture. The following additional information is provided for COBOL/2 and FORTRAN/2.

COBOL/2

- An intermediate level implementation of ANSI X3.23-1985 COBOL.
- A high level implementation of ANSI X3.23-1974 COBOL.
- A subset of VS COBOL II, Release 2.0
- A subset of OS/VS COBOL, Release 2.4
- Highly upward compatible from IBM Personal Computer COBOL Compiler, Version 1.00.

FORTRAN/2

- FORTRAN/2 supports the full standard defined in ANSI X.9-1978. An option warns the user of statements that are allowed by FORTRAN/2 but which are extensions to the ANSI X.9-1978 standard.

Hardware Supported

The recommended minimum hardware configuration for Operating System/2 Standard Edition is:

- IBM Personal Computer system unit with 1.5 MB of memory when configured to run only Operating System/2 applications, and 2 MB of memory when configured to run both Operating System/2 and DOS applications:
  - Personal System/2 (Models 50, 60 or 80)

- IBM Personal Computer AT®[2] (5170): Models 099, 239, 319 or 339
- IBM Personal Computer AT (5170): Model 068 with fixed disk to make the system unit equivalent to a Model 099.
- IBM Personal Computer XT (TM)[3] (5162): Model 286

- One diskette drive (5.25-inch or 3.5-inch) as described below
- One fixed disk drive as described below
- Keyboard
- Display adapter and associated display as described below

The recommended minimum hardware configuration for Operating System/2 Extended Edition is that described above for Operating System/2 Standard Edition plus:

- Recommended minimum memory of 3 MB when configured to run only Operating System/2 applications
- For use of the Communications Manager add one or more modems and/or communications adapters supported by the Communications Manager
- Users needing large data bases, large numbers of programs and files, or execution of several concurrent applications (requiring segment swap areas) should assure they have the capability to expand their fixed disk above 20 MB.

Operating System/2 supports the following devices:

- Diskette Drives
  - IBM 3.5-inch high-capacity diskette drive (1.44 MB)
  - IBM 3.5-inch diskette drive (720 KB)
  - IBM 4865, Model 2. IBM Personal Computer 3.5-inch External Diskette Drive (720 KB)
  - IBM 5.25-inch high-capacity diskette drive (1.2 MB)
  - IBM 5.25-inch diskette drive (360 KB)
  - Personal Computer 5.25-inch External Diskette Drive (360 KB)
- Fixed Disks
  - IBM 20 MB Fixed Disk Drive
  - IBM 30 MB Fixed Disk Drive
  - IBM 44 MB Fixed Disk Drive
  - IBM 70 MB Fixed Disk Drive
  - IBM 115 MB Fixed Disk Drive

    Note: The 44 MB, 70 MB and 115 MB fixed disk drives are supported as multiple logical drives, each having a maximum size of 32 MB.
- One of the adapter and display combinations listed below:
  - IBM Color/Graphics Monitor Adapter with IBM Color Display (5153).
  - IBM Enhanced Graphics Adapter with one of the following displays: IBM Enhanced Color Display (5154) or IBM Color Display (5153).
  - Personal System/2 Display Adapter (FC #4050 for IBM Personal Computer AT or XT 286) with one of the following displays: Personal System/2 Color Display (8513), Personal System/2 Monochrome Display (8503), Personal System/2 Color Display (8512) or Personal System/2 Color Display (8514).
  - Personal System/2 system unit (Models 50, 60 or 80) with one of the following displays: Personal System/2 Color Display (8513), Personal System/2 Monochrome Display (8503), Personal System/2 Color Display (8512), Personal System/2 Color Display (8514).
  - Personal System/2 Display Adapter 8514/A (FC #4054) with one of the following displays: Personal System/2 Color Display (8513), Personal System/2 Monochrome Display (8503), Personal System/2 Color Display (8512) or Personal System/2 Color Display (8514).

Operating System/2 Standard Edition Version 1.0 support for specific adapters is as follows:

- IBM Color/Graphics Monitor Adapter. Operating System/2 only supports text mode (25 lines) in the Operating System/2

[2] Personal Computer AT is a registered trademark of the IBM Corp.

[3] Personal Computer XT is a trademark of the IBM Corp.

environment and CGA compatibility modes in the DOS environment.

- IBM Enhanced Graphics Adapter. Operating System/2 only supports text mode (25 or 43 lines) in the Operating System/2 environment and CGA compatibility modes in the DOS environment.
- Personal System/2 Display Adapter (FC #4050 for IBM Personal Computer AT or XT Model 286). Operating System/2 only supports text mode (25 or 50 lines) in the Operating System/2 environment and CGA compatibility modes in the DOS environment.
- Personal System/2 system unit (Models 50, 60 or 80). Same support as for Personal System/2 Display Adapter (FC #4050).
- Personal System/2 Display Adapter 8514/A (FC #4054). Same support as for Personal System/2 Display Adapter (FC #4050).

The Operating System/2 Standard Edition Version 1.1 support for each adapter includes all Version 1.0 support for that adapter, plus all-points-addressable support in the Operating System/2 environment.

- Printers[4]
  - IBM 4201 Proprinter (TM)[5], Model 1
  - IBM 4201 Proprinter II
  - IBM 4202 Proprinter XL, Model 1
  - IBM 4207 Proprinter X 24
  - IBM 4208 Proprinter XL 24
  - IBM 5152 Graphics Printer, Model 2

    Note: The Graphics Printer is no longer marketed
  - IBM 5182 Color Printer, Model 1

    Note: The Color Printer is no longer marketed
  - IBM 5201 Quietwriter®[6], Models 1 and 2
  - IBM 5202 Quietwriter III
  - IBM 5216 Wheelprinter (parallel)
  - IBM 5223 Wheelprinter E (parallel)

    Note: Operating System/2 Standard Edition Version 1.0 supports each of the above printers as an IBM Graphics Printer (5152, Model 2). Operating System/2 Standard Edition Version 1.1 provides all-points-addressable support to the printers, where appropriate.
- Keyboard
- Pointing Devices
  - IBM Personal System/2 Mouse attached to the system pointing device port
  - Serial pointing device
    - Microsoft®[7] Mouse for IBM Personal Computers, Part Number 039-099, 100ppi
    - Microsoft® Mouse for IBM Personal Computers, Part Number 039-199, 200ppi
    - PC Mouse(TM)[8], Part Number 900120-214, 100 ppi
    - Visi-On Mouse(TM)[9], Part Number 69910-1011, 100 ppi
  - Parallel pointing device for IBM Personal Computer AT and XT Model 286
    - Microsoft® Mouse for IBM Personal Computers, Part Number 037-099, 100ppi
    - Microsoft® Mouse for IBM Personal Computers, Part Number 037-199, 200ppi
  - InPort Microsoft® Mouse for IBM Personal Computers AT and XT Model 286, Part Number 037-299, 200ppi

---

4 Parallel attached printers are supported in both the Operating System/2 and DOS environments. Serially (asynchronous) attached printers are supported only in the Operating System/2 environment.

5 Proprinter is a trademark of the IBM Corp.

6 Quietwriter is a registered trademark of the IBM Corp.

7 Microsoft is a registered trademark of the Microsoft Corp.

8 PC Mouse is a trademark of Metagraphics/Mouse Systems.

9 Visi-On Mouse is a trademark of the Visi-On Corp.

- Plotters[10]
  - IBM 6180 Plotter
  - IBM 6184 Plotter
  - IBM 6186 Plotter
  - IBM 7371 Plotter
  - IBM 7372 Plotter
  - IBM 7374 Plotter
  - IBM 7375 Plotter

    Note: The IBM 7371, 7374 and 7375 Plotters are no longer marketed.
- Other
  - IBM Personal Computer AT Serial/Parallel Adapter Card (FC #0215, #3395, or #3400)
  - Personal System/2 Dual Asynchronous Adapter/A (FC #3033)
  - Personal System/2 Multiprotocol Adapter/A (FC #3042) in asynchronous mode[11]
  - Math Coprocessor (Intel®[12] 80287)
  - Math Coprocessor (Intel® 80387)[13]

For additional memory for the IBM Personal Computer AT or XT Model 286, the customer can order one or two of the following combinations:

- FC #0209 (128 KB Memory Expansion, AT only)
- FC #3343/3339 (512 KB to 2 MB Memory Expansion)
- FC #3395/3397 (512 KB to 3 MB Memory Expansion)
- FC #3395/3397/3402 (512 KB to 5.0 MB Memory Expansion)
- FC #3400/3402 (1 MB to 6 MB Memory Expansion)

For additional memory for the Personal System/2 (Models 50 or 60), the customer can order:

- FC #3006/3012 (512 KB to 2 MB Memory Expansion)[14]
- FC #3920 (2 MB Memory Expansion)[14]

For additional memory for the Personal System/2 (Model 80), the customer can order:

- FC #3009 (1 MB System Board Memory Expansion)
- FC #3019/3064 (2 MB to 6 MB Memory Expansion)

Supported Hardware for Family Applications: The application developer can write a Family application which is portable from Operating System/2 to DOS. A Family application is an executable module that can run in all three environments: Operating System/2 Application Environment that runs Operating System/2 applications, an Operating System/2 DOS Application Environment that runs a DOS application, or the DOS Version 3.3 environment. A Family application has the same or similar capabilities as a DOS Version 3.3 application; a Family application cannot use the new Operating System/2 capabilities, such as larger memory addressability, multitasking Application Program Interface or the graphics and windowing capabilities of the Presentation Interface. Assuming there is sufficient memory, display and other appropriate hardware, a Family application can run on all the system units supported by the Operating System/2 and DOS 3.30.

---

[10] Plotters are asynchronously attached, and can be supported only in the Operating System/2 environment. Operating System/2 Standard Edition Version 1.1 provides plotter support. Plotter support for Operating System/2 Standard Edition Version 1.0 must be provided by the application. For example, the Graphics Development Toolkit, Version 2.0, supports the IBM 6180, IBM 7371, and IBM 7372 plotters.

[11] A maximum of three serial ports is supported on the Personal System/2 (Models 50, 60, or 80) system unit. One port is already on the system board.

[12] Intel is a registered trademark of the Intel Corp.

[13] Supported as an 80287.

[14] Operating System/2 supports this feature as extended memory.

80286 Protected Virtual Address Mode

Preface

The purpose of this chapter is to provide the reader with an introduction to the protected virtual address mode of the Intel 80286 microprocessor. The emphasis is on the benefits of this new mode and its programming considerations. Some of the information concerns only operating system developers but is provided as additional background for application developers. For additional information, refer to the *Intel iAPX 286 Programmer's Reference Manual* or the *Intel iAPX Operating Systems Writer's Guide*.

Introduction

Since the introduction of the IBM Personal Computer, microcomputer applications have become increasingly complex, and their requirements for computer resources have grown dramatically. Systems extensions for networking and host connectivity, graphics and sophisticated user operating environments require additional resources when a personal computer is widely used as a workstation.

Fortunately, new technology has become available to meet these needs. Physical memory sizes, processor speeds, and secondary storage device capacities all have increased significantly, providing microcomputers with far more resources at the same or lower cost than were available five years ago. However, the basic architecture of IBM DOS, which was designed for "real address mode" of the Intel 8088 and 80286 microprocessors, has remained largely unchanged during this period and is subject to these hardware limitations:

- The "640K Barrier"

  The addressing scheme restricts the amount of physical memory the machine can actually address

- No hardware memory management support

  The processor does not provide features that facilitate system software techniques to improve physical memory utilization

- No hardware multitasking support

  There are no hardware mechanisms for providing tasks with separate address spaces or for maintaining and switching task state

- No hardware protection

  There are no mechanisms to prevent a program from inadvertently disrupting the execution of system software or other programs

The Intel 80286 microprocessor provides the IBM Personal Computer AT, the IBM Personal Computer XT Model 286, and the IBM Personal System/2 Models 50 and 60 with features to utilize and effectively manage resources, enhancing the environment for developing system software, system extensions, and application programs. These features are available when the 80286 microprocessor operates in "protected virtual address mode," and they include larger physical memory addressing, virtual memory support, hardware protection features and multitasking support.

The IBM Personal System/2 Model 80 is based upon the Intel 80386 microprocessor which is utilized by Operating System/2 as if it were an 80286. Most of the description in this chapter applies equally well to the Model 80.

Terminology

For the sake of brevity, the following terms will be used for the meanings indicated:

8088

The Intel 8088 (iAPX 88) microprocessor. In most contexts, this term also refers to the 80286 microprocessor operating in real address mode.

286

The Intel 80286 (iAPX 286) microprocessor. In most contexts, this term refers to the 80286 microprocessor operating in protected virtual address mode.

Protect Mode

The protected virtual address mode of the 80286 microprocessor.

8088 Review

In order to better understand protect mode, it is helpful to review certain aspects of the 8088 architecture. In particular, those areas that differ in protect mode are mentioned here.

The 8088 operates in "real address mode," where programs are loaded in their entirety into physical memory and address memory directly. The registers in the 8088 each contain 16 bits, and a segmentation scheme is employed to combine the contents of segment registers with offset values to produce 20-bit addresses. This effective memory address calculation method results in a maximum physical address space of 1 megabyte.

When a program is loaded into memory, the loader program selects the memory locations for the program segments and relocates references. Since the program has access to segment register values, it can identify where in memory its segments have been loaded and use alternative methods for referencing data. Therefore, once a program has been loaded into memory, its segments cannot be moved.

Programs executing on the 8088 processor are free to reference any memory location. In addition, applications are free to execute IN and OUT instructions, allowing them to control I/O devices and other hardware components directly. While this lack of protection mechanisms offers flexibility, it does not assist in debugging programs and is not conducive to having multiple programs execute concurrently.

With appropriate system software, it is possible to effect multitasking on the 8088 processor. The processor architecture does not provide any special features for multitasking, so maintaining and switching task states is the responsibility of the system software. Furthermore, since there are no hardware protection mechanisms, there is no method for determining which programs may co-exist in such a multitasking system. Exhaustive compatibility testing is the only means to verify that combinations of programs will operate correctly in a multitasking system on the 8088.

286 Overview

The 286 offers two distinct operational modes. The first of these is the "real address mode," the mode in effect under DOS. In real address mode, the 286 processor is similar to the 8088. The execution speed of the 286 processor in this mode is 3-5 times faster than the 8088, and memory addressability is still limited to 1 megabyte physical memory. There are some minor differences between the 286 in real address mode and the 8088, such as new instructions and differences in the way some other instructions work.

The second mode of the 286 is the "protected virtual address mode," or "protect mode." In protect mode, the 286 processor addresses up to 16 megabytes of physical memory. Four additional address lines are activated, to provide 24 bit physical memory addresses. Programs no longer address physical memory directly, and the segment register values have no relation to the physical memory addresses at which segments are loaded. This memory addressing scheme provides each program with a "logical address space" of up to 2^{30} bytes, or 1 gigabyte.

Protect mode also provides the operating system with special hardware features for multitasking. Hardware primitives are available for maintaining task state and switching tasks. The memory addressing mechanism provides separate address spaces for each task, and prevents one task from corrupting the address space of another task.

The protection mechanisms of the 286 processor provide several advantages for implementing a multitasking system. System software and application programs are protected from inadvertent disruption by other application programs which may contain errors. The protection mechanisms can help to identify errors in system software and application programs which might otherwise go undetected. These mechanisms also facilitate system software management of resources. Protection is accomplished through the use of "privilege levels." These levels allow system software to control access to data and code segments and enforce a software hierarchy of resource management and hardware control that is beneficial to all programs in the system.

Since application programs are protected from each other, the need arises to allow programs to share data. The 286 operating in protect mode provides the ability to share data in different ways, depending on the situation. A data segment can be shared among all programs, or sharing can be limited to a specific set of programs. The 286 allows system software to select the appropriate type of data sharing based on application programs' requirements.

The 286 processor, operating in protect mode, provides a virtual memory capability which permits programs to be larger than physical memory. Systems software can support virtual memory in conjunction with secondary storage devices to provide better utilization of physical memory. This capability also eliminates the need for specific program memory requirements since there is no longer a direct relationship between physical memory size and logical address space.

Memory Management

The memory addressing method employed by the 286 is quite different from that of the 8088. From the view of the application program, the principal difference is in the usage of the segment register. Recall that in the 8088, the segment register contains the upper sixteen bits of the physical memory address of the beginning of the segment.

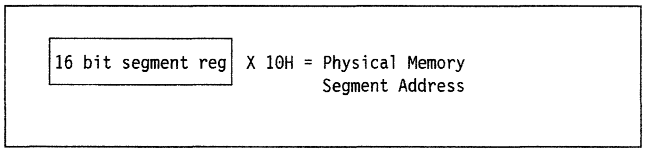


Figure 1. 8088 Effective Memory Address Calculation

In the 286, the segment register still contains a value which identifies the segment, but this value has no relationship to the starting address of the segment. Instead, the value in the segment register contains an offset into a table which the operating system maintains, and the processor obtains the physical memory address of the segment from this table.

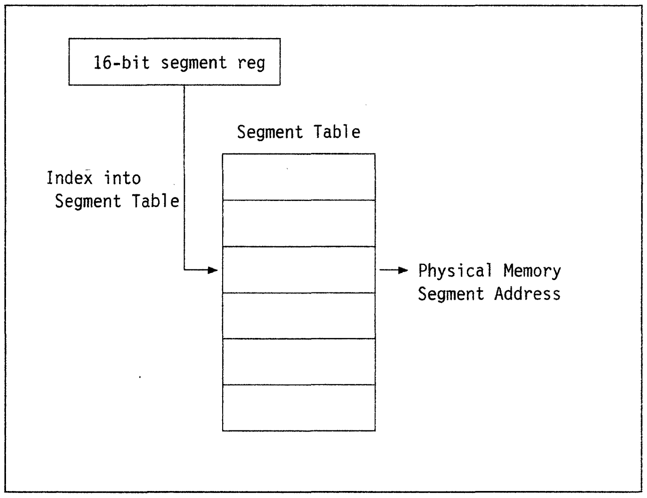


Figure 2. Protect Mode Effective Memory Address Calculation

The value placed in a segment register to identify a segment is called a "selector." The selector contains the index into the segment table as well as protection information and an indication of which segment table is to be used.

The entries that identify segments in physical memory in the segment tables are known as "descriptors." A descriptor contains information about the segment it identifies, including the starting address of the segment, the size of the segment and access information for use in memory management. These segment tables, therefore, are referred to as "descriptor tables." At any instant, there are two active descriptor tables in the system, the global descriptor table (GDT) and the local descriptor table (LDT). A bit in the selector indicates which table is to be referenced.

The effective address calculation of the 286 can be summarized as follows: The program specifies a segment (selector value) and offset. The processor uses the selector to identify a descriptor which, in turn, indicates the physical memory address of the segment. The effective address is generated at the specified offset in that segment.

Typically, there will be a single GDT in use at all times and a separate LDT for each task. The GDT defines code and data segments which are available to all tasks, whereas each LDT defines the code and data segments specific to its task. When the processor switches from executing one task to another, the new task's LDT automatically becomes the current LDT in the system. Therefore, as each task executes, it sees its own address space but does not have access to address spaces belonging to other tasks.

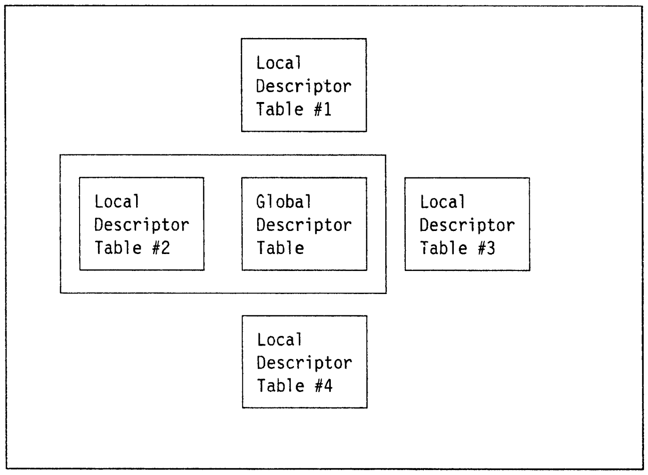

Local Descriptor Table #1
Local Descriptor Table #2
Global Descriptor Table
Local Descriptor Table #3
Local Descriptor Table #4

Figure 3. Address Space for Task 2

Hardware Protection

With multiple programs executing concurrently in the system, hardware mechanisms must be available that enable system software to insure that programs do not conflict with each other and that their requests for resources are handled in an orderly manner. These mechanisms provide what is referred to as a "protected system" and are intended to assure the correct overall operation of the system. This is accomplished by preventing application programs from performing operations which could interfere with system software or other application programs, either through an inadvertent action such as a programming error or through an intentional action.

Providing separate address spaces for programs and confining each program to its own address space is an essential first step in providing a protected system. However, a complete protection scheme involves other considerations as well.

One problem which must be addressed is the restriction of access to I/O devices to trusted software in order to provide proper resource management. For example, if two programs attempted to print at the same time and were allowed to do so by simply performing I/O operations to the printer I/O port, the result would be unpredictable. By requiring programs to request printing through a call to system software, these requests can be managed appropriately by the system software to assure the printing is performed serially.

In addition, certain instructions are intended for use by the operating system to control memory management and other architectural features. If application programs execute these instructions, the operation of the system could be disrupted. Therefore, there must be some means to restrict these operations to system software only.

The 286 provides this protection capability through the use of "privilege levels." These privilege levels denote different levels of trust and can be used to control access to both code and data, as well as to certain sensitive processor instructions and operations. There are four different levels, numbered 0 through 3, with 0 being the most privileged or "most trusted." This allows different levels to be associated with various components in a software hierarchy. For example, one approach that system software designers could choose would be to reserve level 0 for the system software, level 1 for I/O device drivers, level 2 for system extensions and level 3 for application programs.

Protection for data segments can be provided in different ways. Only those segments whose descriptors reside in a program's LDT or in the GDT can be accessed by that program. In addition, a privilege level is associated with each descriptor which specifies the required privilege level at which a program must be executing in order to access the segment. The type of access permitted for each segment is also checked, such as whether or not the segment can be written, and a check is performed to make sure the segment is not referenced beyond its limit.

Code segments are protected in much the same way as data segments. The 286 also provides checking to prevent code segments from being modified and checking which allows them to be read selectively. A code segment cannot be called from a code segment at a different privilege level, except through the use of a special mechanism known as a "call gate." Call gates are entries in either the LDT or the GDT which permit control transfers to code at higher privilege levels. This mechanism allows a program to call system software or system extensions to perform functions on its behalf. The call gate also has a privilege level associated with it, and the privilege level of the gate must be the same or lower (numerically higher) than the caller to allow the transfer. Call gates cannot be used to transfer control to lower privilege levels.

Instructions which are required to perform I/O operations are restricted to code executing at or above a specified privilege level. These instructions include IN, OUT, INS, OUTS, CLI and STI. The privilege level required to execute these instructions is set by level 0 code. This approach permits flexibility in determining the necessary privilege level for executing I/O instructions, and this level would normally be set to the intended level for device drivers.

Certain instructions affect the correctness of overall system operation and must be guarded to maintain a secure system. An example is the LLDT instruction which loads the local descriptor table register. If an application program executed this instruction, it would be possible for that program to access data in other applications. To prevent this problem, instructions of this type are restricted to execution by code at privilege level 0.

Multitasking and Task Management

In addition to maintaining a separate address space for each task, the 286 provides other features for efficiently handling the concurrent execution of programs. The 286 has special hardware support for storing the state of a task in execution and for switching the processor from executing one task to another. Without this hardware, these operations would have to be performed entirely by system software.

The 286 hardware stores the task state in a special data structure known as the task state segment (TSS). The TSS identifies the LDT for that task, its segment register values, general purpose register values, instruction pointer, flags and stack register values. The TSS for the currently executing task is pointed to by the task register (TR). When the processor is directed to switch from executing one task to another by the operating system, the state of the processor is stored in the TSS for the currently executing task. The processor state is then reloaded from the TSS for the next task to execute, and that task is then brought into execution.

A task switch can be caused by the execution of a CALL, JMP, INT or IRET instruction. The operand specified for the instruction determines whether or not a task switch occurs. The operand may be a descriptor to the TSS of the task to switch to, or it may refer to a "task gate." While TSS descriptors must reside in the GDT, task gates may reside in the LDT, giving the task associated with that LDT the ability to cause task switches to the task referenced by the gate.

Interrupt Processing

The 286 processes interrupts differently than the 8088. Rather than having interrupt vectors at fixed locations in memory, a data structure maintained by the operating system is used by the hardware to identify the action taken for each interrupt. This data structure is known as the "interrupt descriptor table" (IDT), and may contain interrupt gates, trap gates or task gates.

The processor takes different actions when an interrupt occurs, depending on the type of gate specified for the interrupt. Interrupt or trap gates cause an interrupt service routine to be invoked under control of the task which was running when the interrupt occurred. Task gates direct the processor to perform a task switch to an interrupt service task when the interrupt occurs, so that interrupt processing is handled under control of a separate task. When interrupt servicing is complete, the processor returns or performs a task switch back to the process which was interrupted, depending on the how the interrupt is being processed.

Data Sharing

Since application programs cannot access the logical address spaces of other programs, mechanisms must be provided in order for data to be shared among programs. Some types of data are appropriate for sharing among all programs, and some should only be shared among specific programs. Moreover, a program may share part of its data with one program, another part with a different program, and keep other data private so that other programs do not have access to it. A program also may control how shared data is accessed by other programs, for example, allowing the data to be read but not written. The 286 provides mechanisms for implementing all of these data sharing capabilities.

The 286 has three different mechanisms for data sharing, each of which is appropriate under different circumstances. The first of these is the GDT, where a descriptor is visible to all tasks operating in the system. The privilege level specified in the descriptor is the only means to restrict access to the segment when this type of sharing is used. Any program with the same or higher privilege level than the descriptor may access the segment. Since all programs access the data through the same descriptor when this type of sharing is used, the access rights are the same for all programs that use the segment, and all programs view the segment as having the same type.

Another way programs can share data is to use the same local descriptor table. With this approach, sharing is restricted to those programs which share all common LDT. The limitation of this mechanism is that programs which share an LDT have access to all of each other's segments. As with sharing through the GDT, accesses are through the same descriptor, so the access rights and segment type are uniform. Because the address spaces of the tasks involved are the same, this method is applicable only if programs are closely cooperating.

The third mechanism for sharing data involves duplicating descriptors for segments. These duplicate descriptors are referred to as "aliases," and provide greater flexibility in controlling how segments are shared. Through the use of aliases, it is possible to specify exactly which segments are to be shared and which tasks are to have access to them. Aliases also can be used to provide different access rights to segments or to define segments with varying types for different programs. For example, a single segment can be defined as a code segment for one program and a readable data segment for another.

Virtual Memory

Virtual memory is a technique often used in larger computer systems to provide the characteristics of very large physical memory. This is accomplished by executing programs even though all of their code or data may not be resident in memory at any one time. The 286 processor provides the necessary hardware support for implementing a virtual memory system.

Virtual memory offers several important benefits. In particular, any single program or collection of concurrently executing programs can be larger than physical memory. Programs need not specify a minimum memory size in order to execute since the virtual memory system automatically simulates the required amount of memory for the programs executing. This is accomplished without requiring programs to manage manual code or data overlays. The result is improved system-wide utilization of physical memory.

The concept of virtual memory is based on certain assumptions about program behavior. Provided that a large program is made up of many code and data segments, studies often show that, over a period of time, only a relatively small subset of the program's segments will be referenced. As the program continues to execute, membership in this set of segments will gradually change. This is referred to as "locality of reference," and it is the key to an effective virtual memory system. If programs exhibit this behavior, there will be a minimum set of segments which must be in memory at any time for programs to execute efficiently. This minimum set of segments is known as the "working set," and physical memory need only be large enough to accommodate the working set for programs to execute efficiently and to prevent segments from being replaced too frequently.

The hardware provided by the 286 for virtual memory includes a "present bit" in the segment descriptor and a "segment-not-present" trap. Whenever a program references a segment which is not currently in memory, an exception interrupt is generated. The system software allocates memory for the segment to be brought into memory. If necessary, a segment to be replaced is identified and written to disk. The segment which generated the fault is then read from the disk, the descriptor is updated and the instruction restarted.

Programming Guidelines

The virtual protected address mode of the 286 is not completely software-compatible with the 8088 or the real address mode of the 286. While it is possible to design programs which will run in both modes, many programs previously written to take advantage of certain characteristics of the 8088 will not work in protect mode. Certain programming techniques used with the 8088 should be avoided to facilitate a program running either in protect mode or in both real mode and protect mode.

Segment addressing differences have the greatest effect on 8088 applications being considered for protect mode. In particular:

- Do not depend on any relationship between segment numbers and physical addresses. The selector values in segment registers do not indicate the physical addresses of the segments.
- Do not depend on segments overlapping. Segments must be referenced through valid selector values only.
- Do not use "wraparound" segment offsets. Wraparound offsets can cause unpredictable results.
- Do not access beyond the allocated size of a segment. The segment size is checked during effective address calculation.

- Place only valid segment numbers in segment registers. The descriptor privilege level is checked when a segment register is loaded, and a fault will occur if a valid segment is not specified.

Protect mode also places restrictions on code segment usage to assure that code segments can be used by multiple tasks. The following points must be remembered about code segments in applications running in protect mode:

- Do not put data in a code segment. A fault will occur if an attempt is made to modify a code segment.
- Do not modify the contents of a code segment. A fault will occur if an attempt is made to modify a code segment.

System services for I/O and other functions should be used rather than performing these functions directly from an application. In protect mode, application programs must not attempt to perform certain instructions or operations which are reserved for use by system software:

- Do not attempt to issue privileged instructions. Privileged instructions can only be executed at privilege level 0.
- I/O instructions can be executed only with the proper privilege level. A fault will occur if a program attempts to execute these instructions without the required privilege level.

There are several other differences between the 286 and the 8088:

- PUSH SP results in different SP values between the two processors.
- Shift counts greater then 31 do not produce the same results.
- Redundant or repeated prefix bytes can cause the instruction to exceed the maximum length of 10.
- The use of undefined OP codes results in a fault.
- There is an error in the POPF instruction in the 286 processor.

In addition to the areas mentioned, programmers should be aware of several other points about protect mode. Programs should not depend on the execution speed of the processor for any purpose since multitasking may affect the behavior of an application under varying circumstances. The flag register should be examined only through the use of flag specific instructions rather than depending on specific flag register values. This is because the 286 defines additional flag register bits. Since interrupt vectors are maintained in the IDT, which is not visible to application programs, interrupt vectors cannot be "hooked." Functions which would have been performed through hooking interrupt vectors should instead be performed through operating system or system extension programming interfaces.

Note: "Hooking" is a process in which a program replaces an existing interrupt vector address in the system with an address of its own interrupt service routine.

General Architecture of Operating System/2

Operating System/2 is a new operating system that provides support for the Intel 80286 processor's unique features -- protected virtual mode operation, multiprogramming and improved memory management. Operating System/2 particularly emphasizes utilization of real memory above 640 KB. In addition, the level of function in DOS 3.30 is available as a migration aid for existing DOS applications.

This chapter presents a conceptual overview of the system structure, functions, features and major components of Operating System/2. The approach is to depict various features of Operating System/2 and to describe the most important concepts. For a full understanding of some of the finer points of the Operating System/2 architecture, certain portions assume the reader is familiar with the architecture of the 80286 processor, covered in the previous chapter.

Enhancements Over DOS 3.30

The significant enhancements of Operating System/2 over DOS 3.30 are:

- Memory management expanded to support large real memory above 640 KB
- Full multitasking support
- System services invoked by a CALL rather than INT 21H with runtime resolution to the target program
- An extensive set of console device interfaces for the video, keyboard and mouse
- An extensive set of session management features.

A major compatibility goal of Operating System/2 is that applications written under previous versions of DOS must be executable in Operating System/2. This goal is accomplished by means of a DOS 3.30 execution environment, hereinafter referred to as the DOS Application Environment. Programs executing in this DOS Application Environment cannot take advantage of the new functions of Operating System/2.

Figure 4 depicts the overall Operating System/2 software system structure.

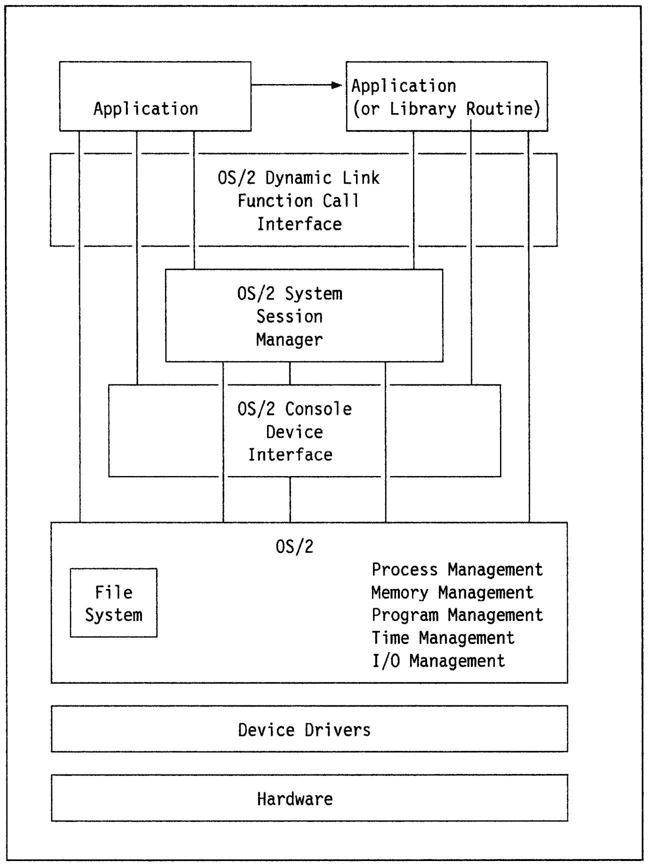


Figure 4. Overall Operating System/2 System Structure

Memory Management

One of the major features of Operating System/2 is its segmented memory management, which exploits the virtual segmentation hardware in the protected mode of the 80286.

This memory management feature permits applications to use more memory than the 640 KB limit of previous DOS versions. A comprehensive set of memory management functions makes it possible to use memory up to the processor's storage limit of 16 MB.

For applications running in this new Operating System/2 Application Environment, Operating System/2 lets the user concurrently run more programs than will fit in storage. In addition, any single program and its data can be larger than real memory. Operating System/2 maintains the most active segments in real memory. It writes unneeded segments out to disk storage, and reloads those segments from disk when they are needed next.

The new memory management functions allow a program to:

- Allocate a large number of data segments, each up to 64 KB in size
- Keep each segment private or share it with other programs
- Package an application so that the linkage to library routines or to infrequently used routines is not made until runtime (Dynamic Linking)
- Implement an application as a number of distinct CALLable segments, with Operating System/2 loading the segments when they are CALLed
- Load programs explicitly if desired.

Protection Hierarchy

The segmented architecture of the 80286 enables automatic protection between applications. In fact, special steps must be taken to *allow* separate applications to access the same memory.

The protection features used by Operating System/2 for program execution are:

- Applications run at privilege level 3
- Special-purpose routines (other than general device drivers) requiring I/O privilege (IOPL) execute at privilege level 2
- Operating System/2 and device drivers run at privilege level 0.

For a module to have I/O privilege (IOPL), the CONFIG.SYS IOPL= command must indicate that this is permitted. The IOPL modules cannot use dynamic linking and, therefore, cannot make Operating System/2 function calls.

The Operating System/2 protection hierarchy takes advantage of the 80286's ring protection architecture to ensure that data at any level in the hierarchy can be accessed only by programs at that level or a more privileged level:

- An application at level 3 can access only its own data, which is also at level 3
- A level 2 I/O module can access both its own data at level 2 and its clients' application data at level 3
- The kernel and device drivers can access data at all levels.

Figure 5 illustrates the protection model provided by Operating System/2.

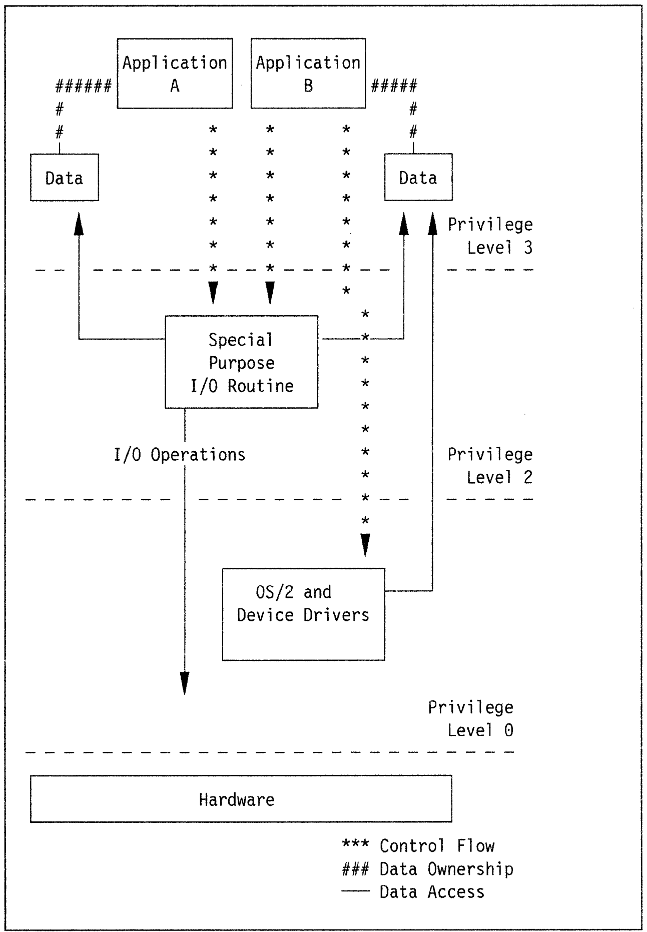


Figure 5. Protection Hierarchy

Operating System/2 Application Environment Memory Management

Figure 6 on page 19 depicts the virtual, segmented, protected memory management of Operating System/2. Note that each application has its own distinct address space defined in its Local Descriptor Table (LDT) and Global Descriptor Table (GDT). The LDT defines a private address space for an application, while the GDT provides addressability for system-wide data and programs that are shared among all applications. Together, these tables provide a virtual address space of 1 gigabyte (2^{30}). The LDT and GDT map this 1-gigabyte virtual address space to the 16-megabyte (maximum) physical address space of the 80286.

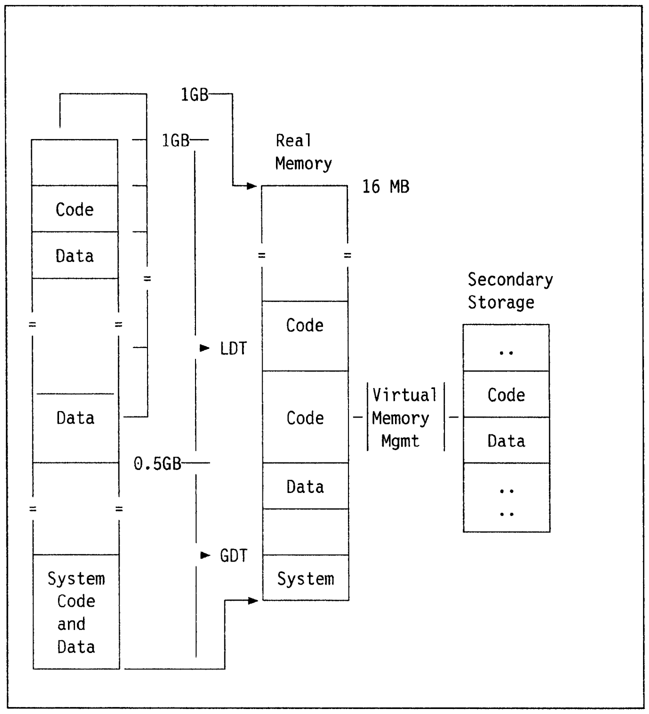


Figure 6. Operating System/2 Application Environment Memory Management

Memory management features include:

- **Storage overcommitment**. The amount of virtual memory allocated at any instant for data and code segments can be and typically will be, greater than the amount of real memory available. Not only can the memory for the system as a whole be overcommitted, but the memory for a single application can be overcommitted as well.
- **Segment swapping**. Storage overcommitment is supported by a least-recently-used (LRU) algorithm that makes room in real memory by writing some segments out to a temporary file on disk (secondary storage).
- **Segment discard**. Real memory occupied by pure segments that are still in the address space of an application but not currently in use can be reclaimed by "discarding" the segment. When the discarded segment is later referenced, a fresh copy is read from the original location on the disk.
- **Segment motion**. Because segments have variable length, real memory is subject to fragmentation due to holes of deallocated real memory. Each hole is not large enough to satisfy a request for memory; however, the space occupied by all holes may be large enough. Segment motion means that all holes are grouped together in order to satisfy the request for memory.
- **Protection**. Applications can address only specific memory segments authorized by the system. Operating System/2 and each individual application are protected against access by other applications.

Operating System/2 Application Environment-Only Real Memory Map

Figure 7 illustrates how real memory is used in a Operating System/2 Application Environment-Only system. Note the "y KB" boundary. This is a movable boundary that separates the fixed and movable regions of memory. The actual location of this boundary varies, depending on the system load and the amount of real memory installed.

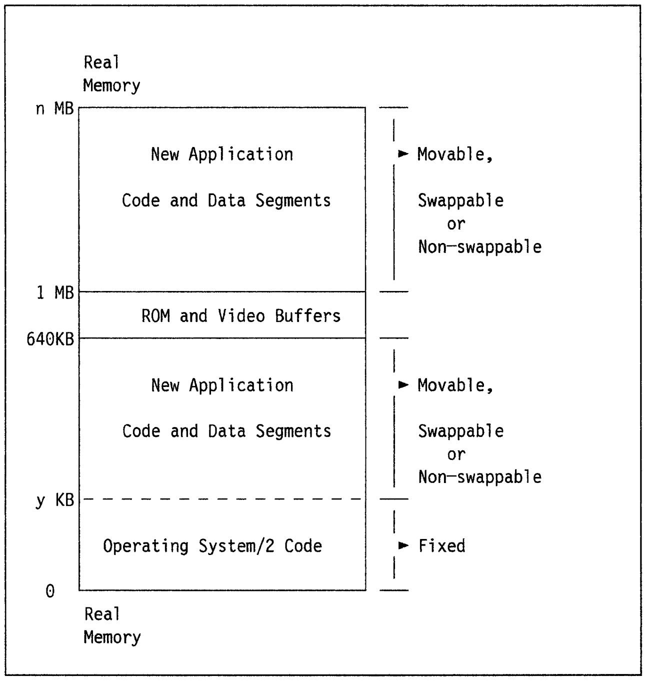


Figure 7. Operating System/2 Application Environment-Only Memory Map

DOS + OS/2 Application Environment Real Memory Map

Figure 8 on page 20 depicts the use of real memory in a DOS Application Environment system. The DOS Application Environment allows execution of a single DOS application (one which was originally intended for DOS 2.00 or higher) with newer Operating System/2 applications.

A DOS Application Environment Program can run only below 640 KB, while an Operating System/2

Application Environment Program can run at any address.

In Figure 8, the "y KB" boundary behaves as described in Figure 7 on page 19. The "n kb" boundary defines the logical "end of memory" for the DOS 3.30 application and may vary up to the 640 KB limit. Above the "n kb" boundary is Operating System/2 Application Environment memory.

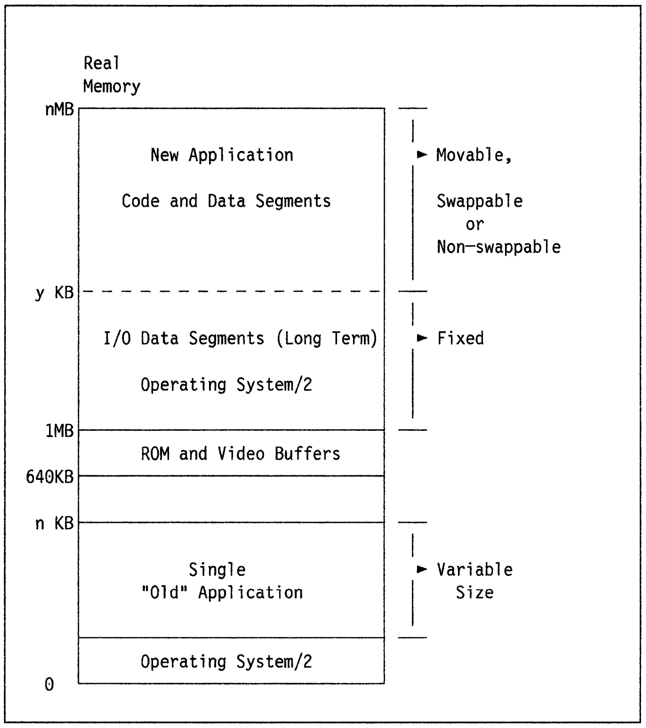
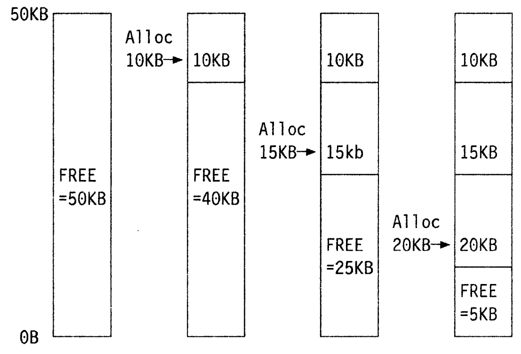


Figure 8. DOS + OS/2 Application Environment Memory Map

Memory Suballocation

In addition to the extensive virtual memory management functions described above, Operating System/2 has a mechanism for suballocating memory within a segment. This mechanism is patterned after the classical linked list of storage descriptors.

Figure 9 gives an example of a series of memory suballocation requests. A succession of allocation requests depletes the free memory within the segment.



The application may now begin freeing the storage as it is no longer needed. Adjacent free blocks will be combined:

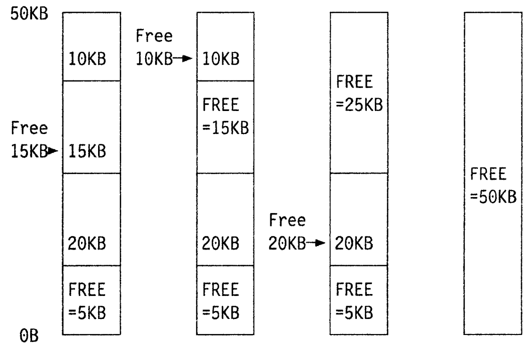


Figure 9. Memory Suballocation Example

System Extensions

Operating System/2 provides a base upon which programmers can construct more complex applications. These application solutions typically appear to be an extension of the operating system, and require many of the capabilities of the operating system, such as protection, data isolation and sharing, and an efficient means of invocation.

Operating System/2 provides functions that allow the system extension developer to obtain these capabilities easily.

The system extension has a callable routine which is its application interface. The actual connection from the application program to the system extension is made at runtime (see the Dynamic Linking section for details about this linkage.) When it is called, the extension routine either performs the request directly or passes the request to a separate process that performs the request asynchronously. The determination of whether to process the request directly or under a separate process must be made based on the data isolation and performance characteristics of the solution being offered.

For processing of the request under a separate process, classical interprocess communications techniques can be implemented using semaphores, queues, flags, or shared memory.

Figure 10 shows the invocation of a system extension via the runtime dynamic linking.

Figure 10 also demonstrates the program execution flow and the data handling aspects of providing a system extension as a callable routine. Note that the extension may allocate private data for each of its clients.

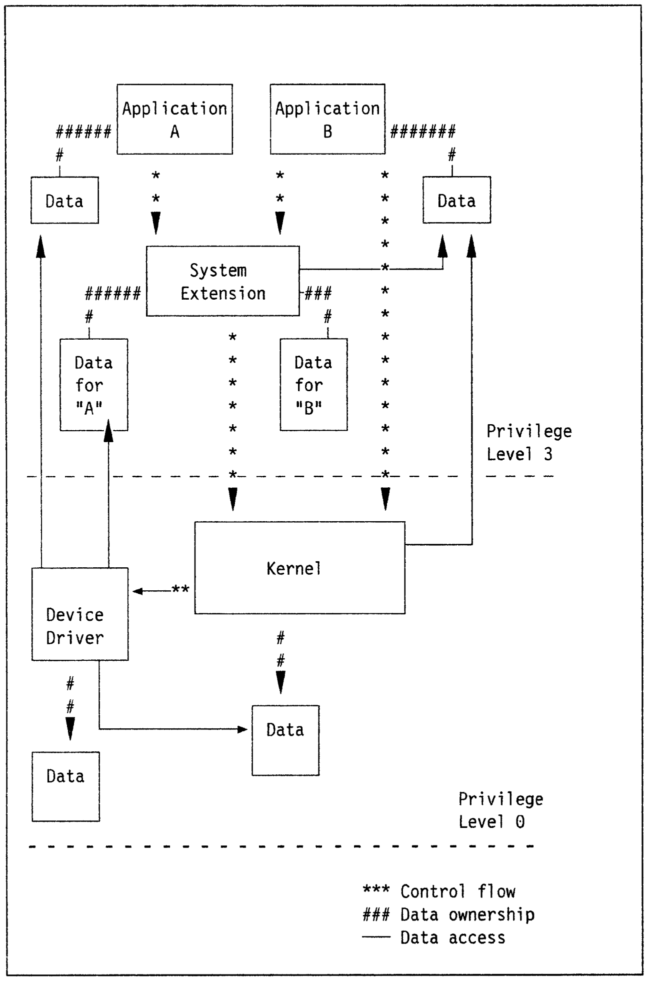


Figure 10. System Extension

Operating System/2 Application Program Interface

Application programs' requests for Operating System/2 services are invoked by a CALL-RETURN interface, with the stack being used to pass the request parameters. The most obvious benefits of this are:

- Less need for a high-level language system services library. The respective system call may be made from a high-level language.
- Optimum performance. The target routine may be invoked directly rather than first giving control to an intermediary router.
- The same interface mechanism is available for invoking an Operating System/2 routine as well as a library routine.
- Function replacement is an architected and well-defined activity rather than an ad-hoc mechanism.

In the multiple protection ring environment of the 80286, significant performance gains can be achieved by using a CALL programming interface when the parameters are PUSHed onto the stack before issuing the CALL. The hardware itself actually copies the parameters from the requestor's stack to the receiving program's stack, thus giving optimum addressability and protection at minimal execution cost. All Operating System/2 functions are invoked via this CALL interface. Also provided are the means to similarly call system extensions and I/O privilege routines executing at privilege level 2. As shown in Figure 4 on page 17, the application interface to Operating System/2 Application Environment and device drivers is strictly hierarchical, and the 80286 hardware supports and enforces this hierarchy.

Dynamic Linking

The 80286's protected mode CALL architecture also offers some benefits of greater importance than the hierarchical structure and data copying mentioned above. In particular, there are definite advantages over the typical static module structure of earlier DOS versions:

- Application programs need only have the most commonly used segments loaded when they are started. Exception processing routines may be left unloaded and be called (and be automatically loaded by the system) only when necessary.
- Dynamic link packages can be updated transparently to their clients; i.e., existing applications need not change whenever functions

they use in a dynamic link package are reshipped.

The actual programming steps required to use the dynamic link feature are no different from those of a static environment. The steps are:

1. The programmer codes a call to a subroutine to be dynamically linked and declares it "EXTRN FAR."
2. The compiler generates a standard external reference.
3. When the object module is linked, the linker is provided with the names of libraries that contain dynamic link definition records. These records provide a correspondence between the called entry point and the module file that contains the routine being called.

Family Application Model

To aid application developers who want to develop a single product that can be used with either Operating System/2 or DOS 3.30, a subset of the full-function Operating System/2 Application Programming Interface (API) maps to functions available in DOS 3.30. An application that conforms to this subset can run under either operating system.

Library bindings provide the interface to the target operating system as a set of modules which are loaded when executing in DOS 3.30 but are ignored in the Operating System/2 Application Environment.

Compatibility Considerations

Programs written to execute in the DOS 3.30 environment should have the following characteristics:

- Run only in the DOS Application Environment of Operating System/2
- Cannot execute as a background process
- Must use the existing INT 21H DOS API
- Must not rely on undocumented DOS interfaces
- Have I/O Privilege.

Programs written to execute in the Operating System/2 Application Environment should have the following characteristics:

- Cannot use the existing INT 21H DOS API
- Must use the new Operating System/2 API CALL interface
- Can overcommit physical memory (data and code) via Operating System/2 memory management functions
- Cannot issue software interrupts
- Cannot process hardware interrupts
- Always reside above the Operating System/2 Application Environment memory line in dual-mode systems
- Must obey Intel 80286 segment manipulation rules (see "80286 Protected Virtual Address Mode" on page 10)
- Do not have I/O privilege without special requests to Operating System/2.

Timer Management

In Operating System/2, all time-related functions are based on a periodically interrupting time source. Because it would require an inordinate amount of system overhead to service a timer at a high interrupt frequency, the timer operates with a frequency of approximately 32 hertz. While this rate is sufficient for most applications, it precludes specifying a time interval with a resolution of less than one timer tick. Time-related discussions in this section are subject to this limitation.

Timer Services

In addition to the Date and Time functions found in previous versions of DOS, Operating System/2 provides the following functions:

- Asynchronous intervals. The system notifies a task that a period of time has elapsed.
- Regularly occurring intervals. The system continually notifies a task that a designated period of time has elapsed.
- Sleep. A task may delay its execution for a certain period of time.

The interval functions permit specification of the time interval in milliseconds; however, to reduce system overhead, the actual resolution is about 32 milliseconds.

Multitasking

Multitasking is an integral part of Operating System/2. Operating System/2 has a priority-based, time-slicing scheduler that gives special consideration to applications with critical response time requirements.

The multitasking features of Operating System/2 enable a user to operate several applications concurrently. For most purposes, each application appears to run independently in a separate computer and can be designed and coded in much the same way as under DOS.

This case of multiple concurrent applications is the simplest instance of multitasking; i.e., each application's execution is managed under a *process* and one *thread* of execution.

A *process* represents the execution of an application and the ownership of any resources associated with that execution, while a *thread* is the dispatchable entity used by Operating System/2 to track processor execution cycles.

An application may be designed as several distinct processes or as multiple threads within a single process. When deciding which is appropriate, the designer should consider these points:

- The creation and termination of a thread is fast, whereas the creation of a process is relatively slow.
- Sharing data and resources between threads is natural, whereas sharing between processes takes special consideration.

For more complex application requirements, an application can be designed to have its functions divided among a collection of cooperating processes (or threads).

Figure 11 shows the process structure when the user starts three applications: an editor, a data base manager, and a communication handler. Because they are independent applications, they are unaware of each other. The fact that their files may share a physical disk is known and managed only by Operating System/2.

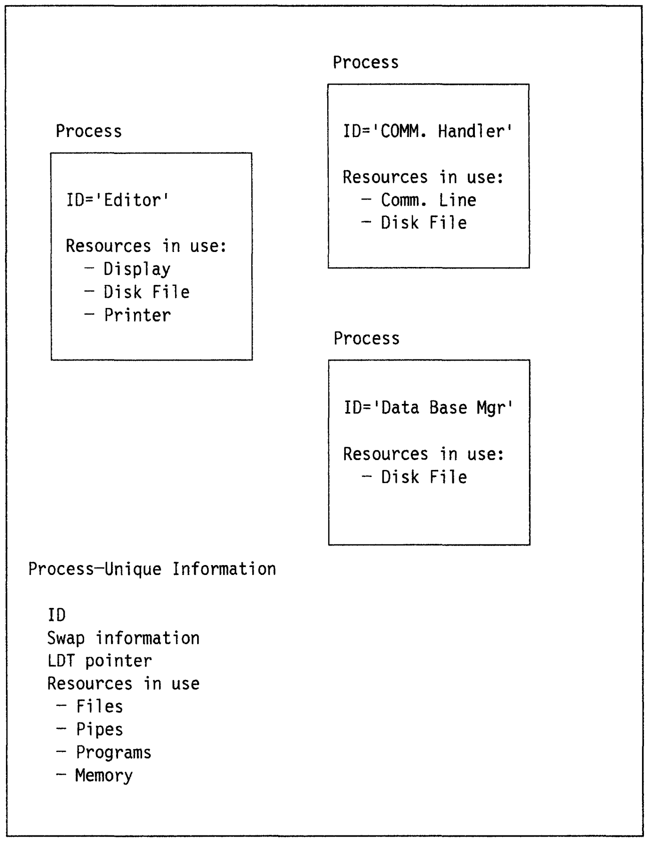


Figure 11. Multiple Independent Processes

Figure 12 on page 24 illustrates an application in which multiple asynchronous execution threads are needed. Each thread uses different devices to accomplish a portion of the overall problem solution.

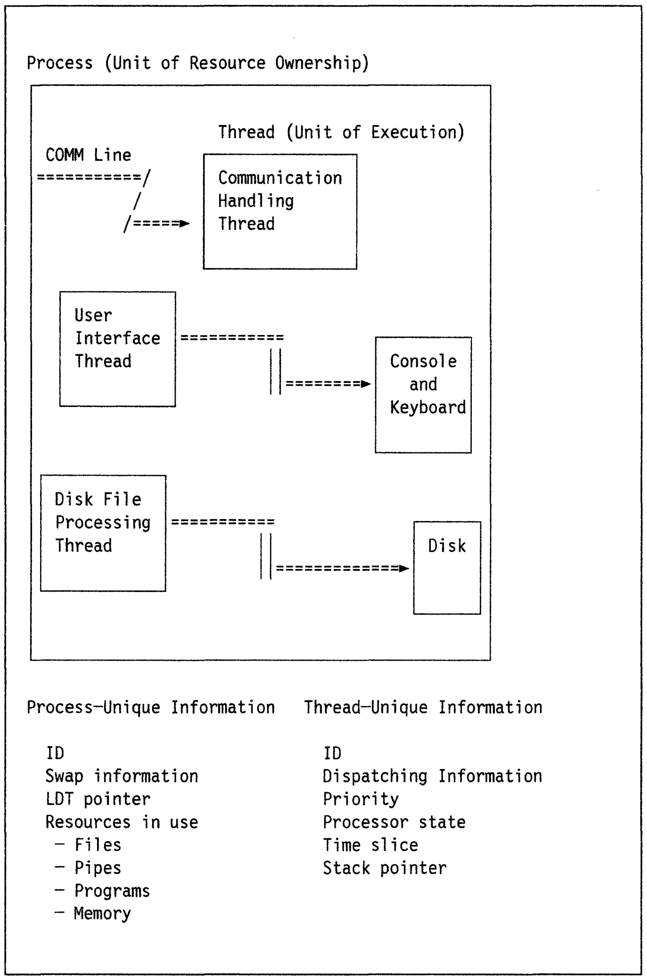


Figure 12. Multiple Threads Within a Process

Interprocess Communication (IPC)

Operating System/2 provides several methods of interprocess communication: pipes, queues, semaphores and shared memory.

Pipes, queues, and semaphores are created by the using applications, and their handles or addresses are passed among the interested processes as necessary.

Semaphores provide serialization and signalling. There are two types of semaphores:

- RAM semaphores are a high-performance mechanism best used between the threads *within* a process.
- System semaphores are a high-function mechanism particularly suited for use *between* processes.

Operating System/2 eases the task of managing the resources shared between programs, by extending the standardized naming conventions of the file system to queues, system semaphores and shared memory. This ensures that references to the same name are resolved to the same resource.

Pipes

Pipes are used to establish file I/O communication between two programs. Typically the two programs are not aware that they are using a pipe rather than a file.

For example, with the OS/2 command interface, a user can produce a sorted directory listing by first issuing a DIR command and then sending the output of DIR, via a pipe, as the input to a SORT command. Figure 13 depicts this use of pipes.

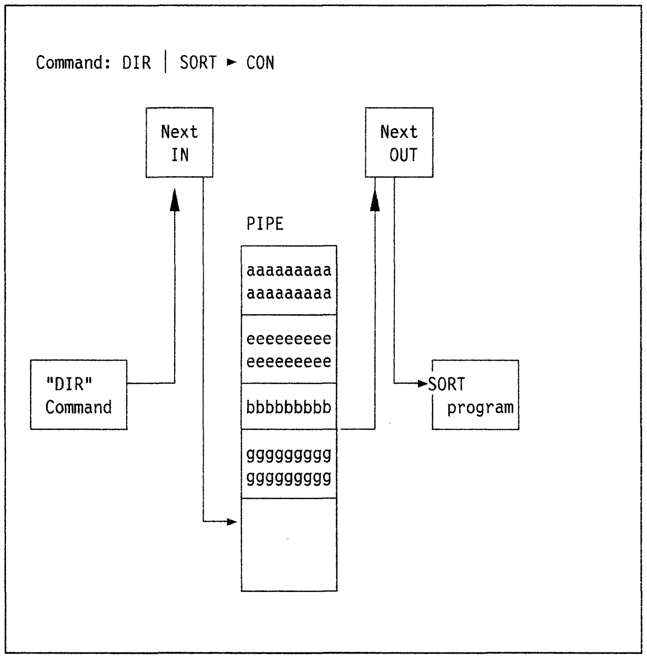


Figure 13. Piping Output of One Program to Another

Alternatively, pipes can be used as a form of fixed-length, First-In-First-Out (FIFO), circular queue that enables communication between processes.

Figure 14 on page 25 depicts the use of a pipe to pass data to a server process X from three child processes A, B, and C. The sending processes send data independently of one another; process X removes data from the pipe at will.

Operating System/2 uses the Next IN and Next OUT pointers to keep track of the data and free space in the pipe. When the pipe is full, the process that writes to the pipe next will wait until process X has removed enough data to make room for the new message.

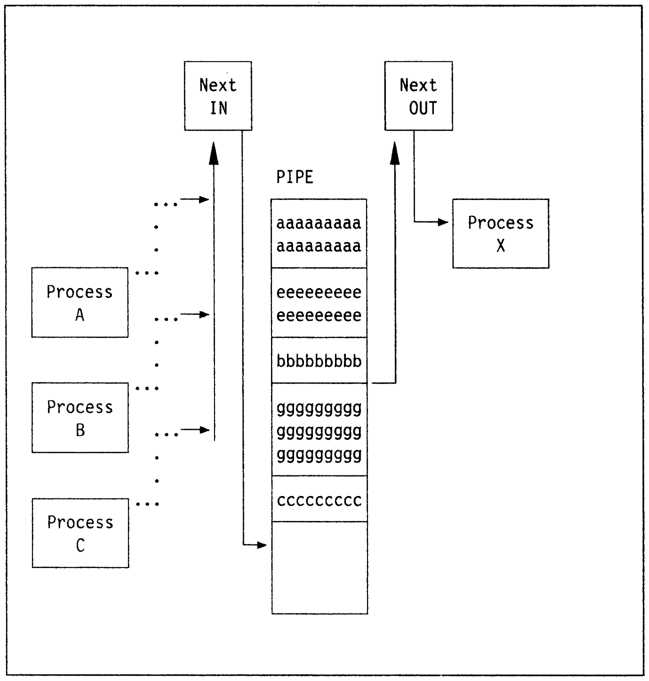


Figure 14. Communication Using a Pipe

Queues

For more sophisticated applications, queues provide a more powerful mechanism for communicating data between processes.

Figure 15 depicts the use of a queue to pass data to a server process Y from three child processes D, E and F. As with pipes, the sending processes may send data independently of one another. However, unique to queues, outstanding elements may be ordered by priority or by arrival order, First-In-First-Out (FIFO) or Last-In-First-Out (LIFO). Also, process Y may examine each element in the queue and remove elements whenever, and in any order, desired.

Queues have a performance advantage over pipes because data is not copied but is passed in a shared segment. Also, there is virtually no size limitation for the messages themselves. While the total message text that a pipe can contain is 64 KB, queues can contain very large amounts of data -- each message is a unique block of storage, and the aggregate of all messages may be dispersed across the entire machine.

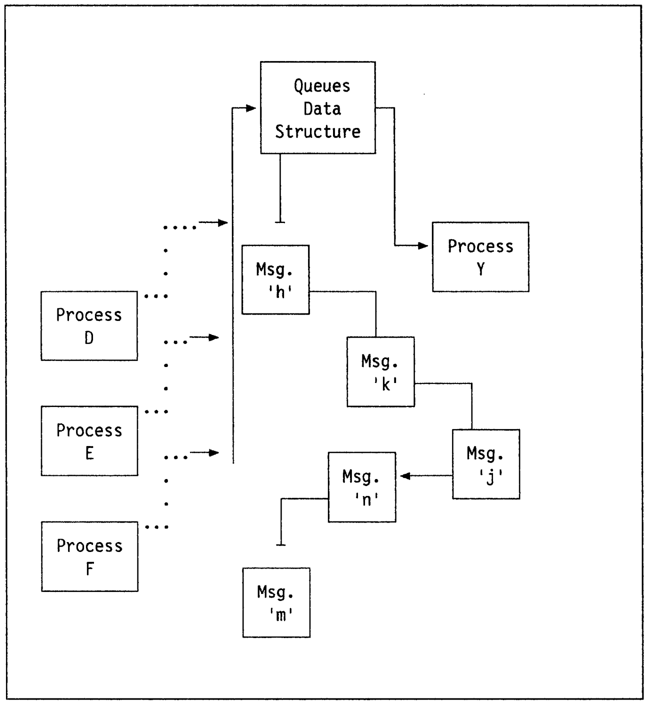


Figure 15. Communication Using a Queue

RAM Semaphores

Figure 16 depicts the use of semaphores for serializing access to a serially reusable resource. Only a single thread can enter the semaphore at a time. The effect is that all other threads are locked out from using the serially reusable resource until the entering thread clears the semaphore.

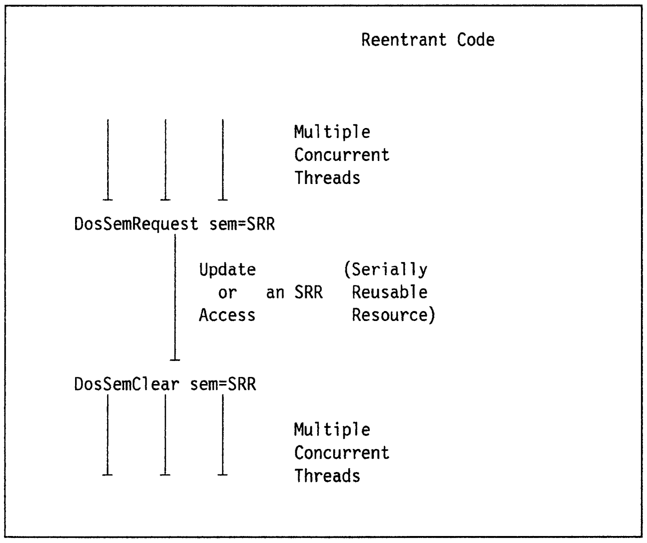

Reentrant Code
Multiple Concurrent Threads
DosSemRequest sem=SRR
Update or Access an SRR (Serially Reusable Resource)
DosSemClear sem=SRR
Multiple Concurrent Threads

Figure 16. RAM Semaphores

Session Management

In Operating System/2, the user's application executes under the concept of multiple screen groups. A screen group is best seen as a routing mechanism for user interaction via the screen and keyboard.

The mode and contents of the screen and keyboard are separate for each screen group. The screen group concept does not extend beyond the user interface. Machine resources, the file system, system memory, interprocess communication, etc., take place throughout the machine environment as a whole and are independent of the screen group concept.

An operator selects an application from a menu presented by the shell user interface program. The shell is the top layer of the session manager. The shell passes the operator's request to start a program to a lower session manager layer which creates a screen group for the application to run in. This layer also controls switching between screen groups when the operator requests it.

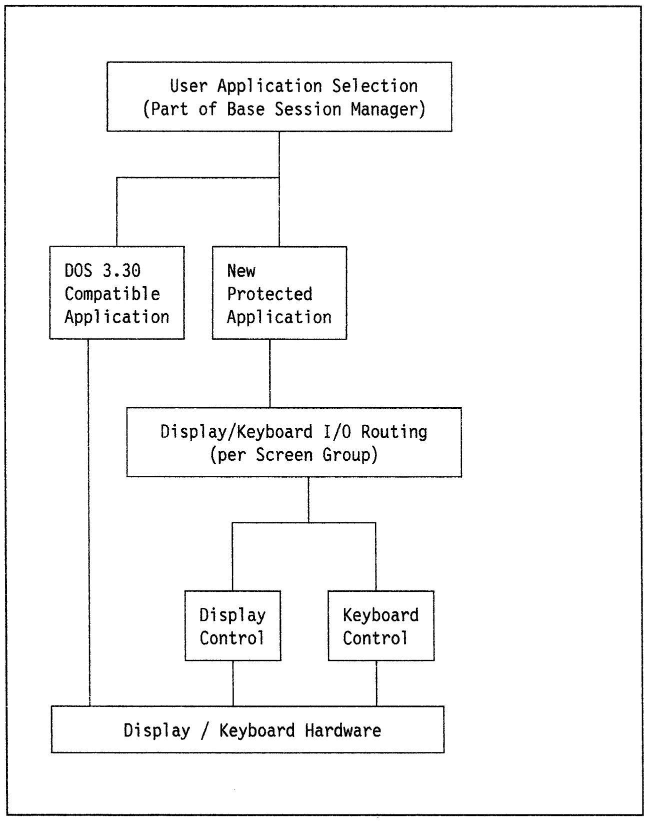

User Application Selection (Part of Base Session Manager)
DOS 3.30 Compatible Application
New Protected Application
Display/Keyboard I/O Routing (per Screen Group)
Display Control
Keyboard Control
Display / Keyboard Hardware

Figure 17. Base Session Manager Controls

The Operating System/2 video I/O architecture makes it possible to replace the standard video I/O system calls within a screen group with the calls provided by an independent video package, such as a graphics package. This replacement is effective only for processes that execute within that screen group.

Figure 18 is an example of replacing the base video I/O functions with a package offering graphics extensions in the block labeled Advanced VIO. The Advanced Graphics Subsystem defined these advanced functions during its initialization so that any applications would have these features available.

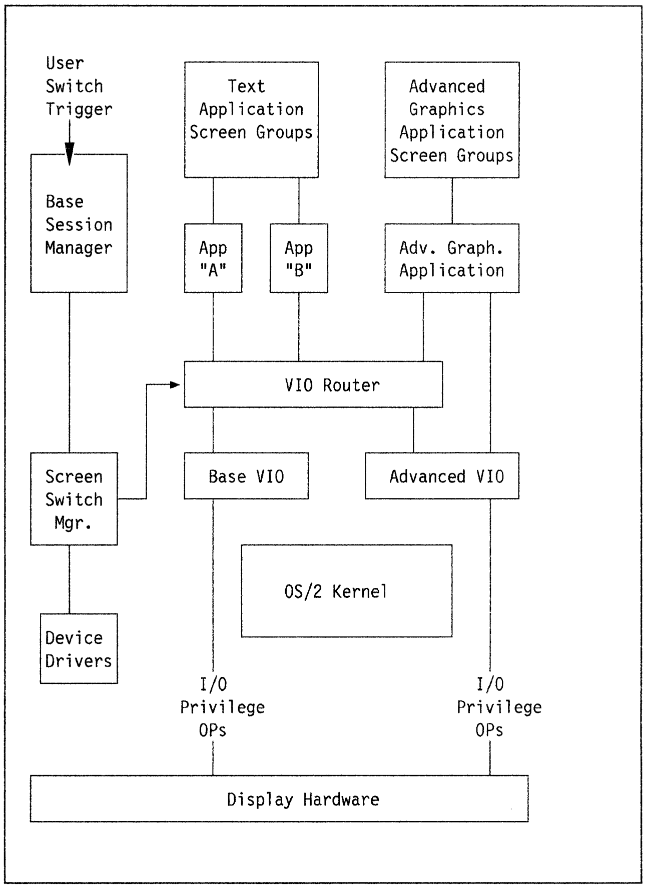


Figure 18. Session Management System Structure

Device Drivers

Operating System/2 device drivers can support multiple synchronous and asynchronous I/O requests. For example, a disk device driver can queue several read and write requests from multiple requesters and service those requests in a sequence that minimizes access mechanism movement across the disk.

Device drivers are divided into two primary parts: a strategy routine and a hardware interrupt handler.

- The strategy routine is called with an I/O packet that describes the request. If the device driver supports multiple outstanding requests and the device is busy, then it can queue the request. If the device is not busy, it starts the device.

- The hardware interrupt handler services the I/O completion. If there is new work in the queue, it redrives the device. Then it indicates that the previous operation is complete and unblocks any threads which are waiting for this request to complete.

OS/2 services are provided for device drivers to manage the request queue, block and unblock, lock and unlock memory, etc.

Figure 19 depicts the flow of control for synchronous I/O:

1. If the device is busy, the strategy routine marks the request incomplete and queues the request.
2. If the device is not busy, it starts the device.
3. Execution of the thread is blocked until the interrupt routine indicates the request is done.

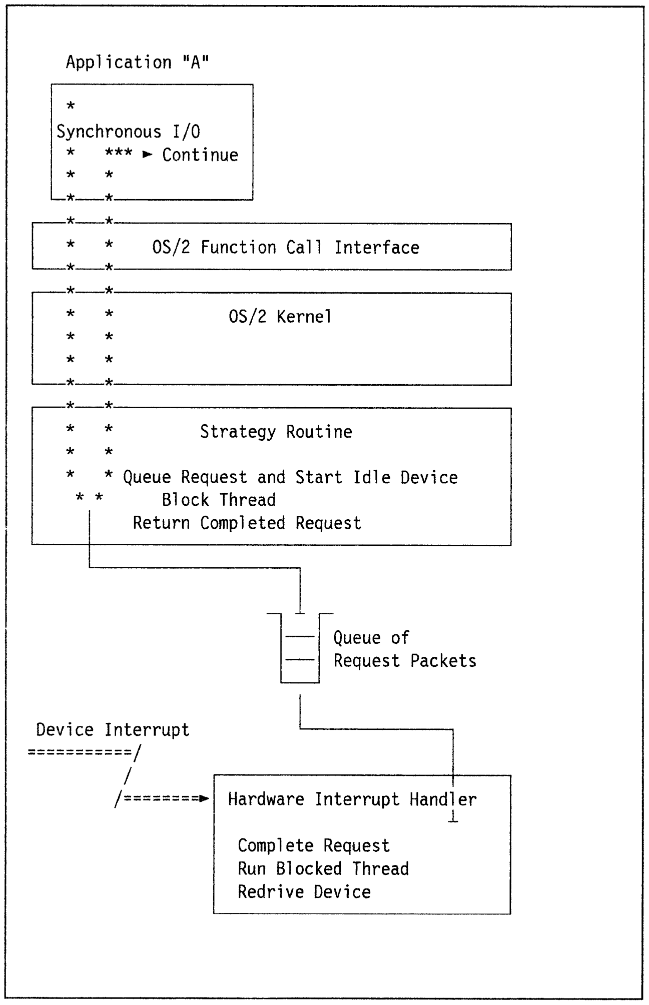


Figure 19. Device Driver for Synchronous I/O

In Operating System/2, asynchronous I/O is performed by using a separate thread. When a read or write request asks for asynchronous processing, the OS/2 device support creates an I/O thread. This thread then performs the I/O request in a synchronous (to itself) fashion while the requesting thread continues executing. When the operation is complete, the I/O thread posts that completion to a RAM semaphore and terminates.

Figure 20 depicts asynchronous I/O. The flow is similar to that in Figure 19, except that the requesting thread continues to execute while a separate I/O thread executes the I/O.

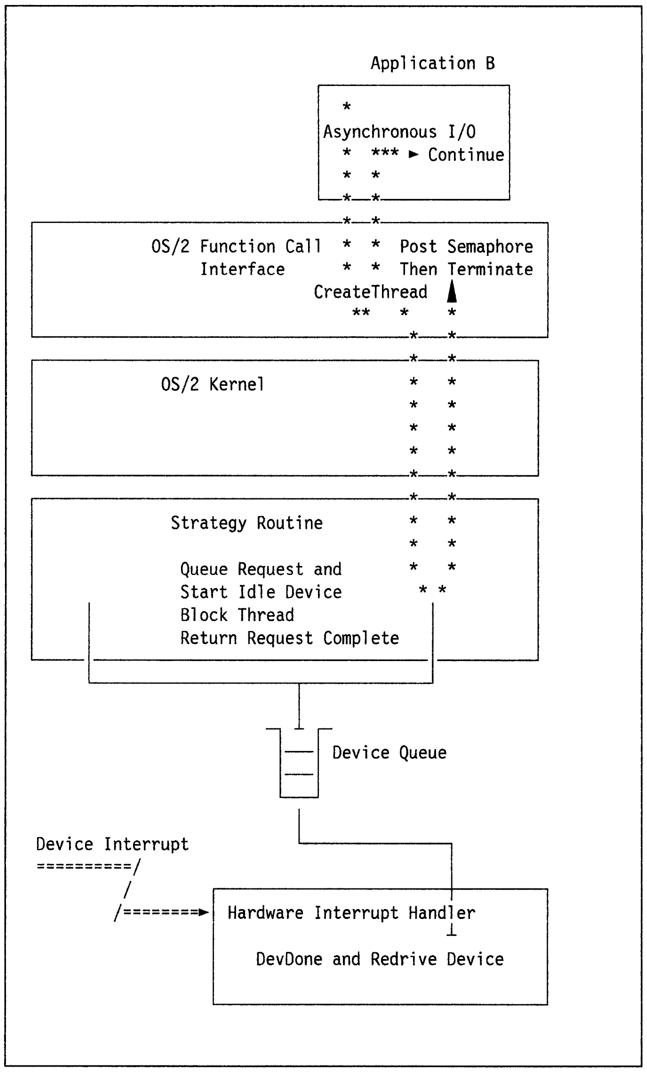


Figure 20. Device Driver for Asynchronous I/O

Video, Keyboard, Mouse Device Interface Management

I/O to the console can be done via handle-based I/O, which is redirectable, or via video (VIO), keyboard (KBD) or mouse (MOU) device interfaces. VIO, KBD and MOU device drivers provide standard read/write entry points.

Figure 21 depicts Video I/O that is not redirectable (such as direct-cursor-positioned I/O). The VIO device interface and its device driver are a pair. A private protocol is used to provide flexibility in placement of function and an opportunity to exploit hardware features.

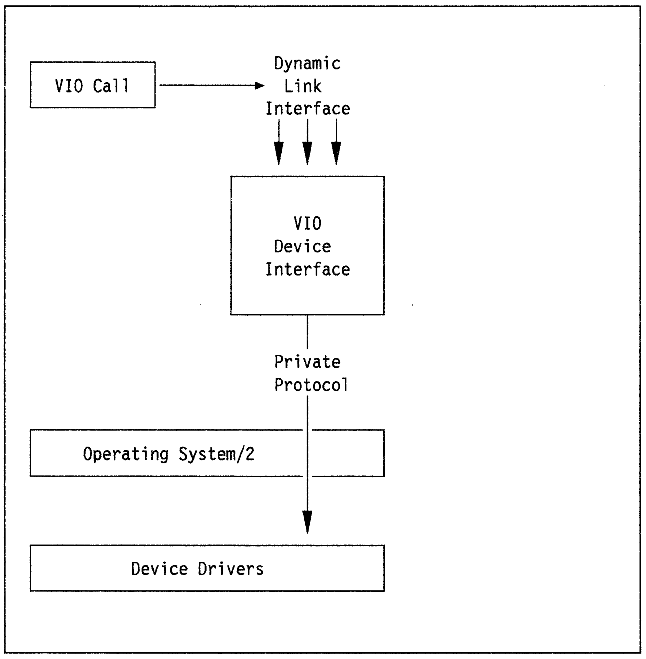


Figure 21. VIO Device Interface

Figure 22 shows function split between the VIO device interface and its device driver.

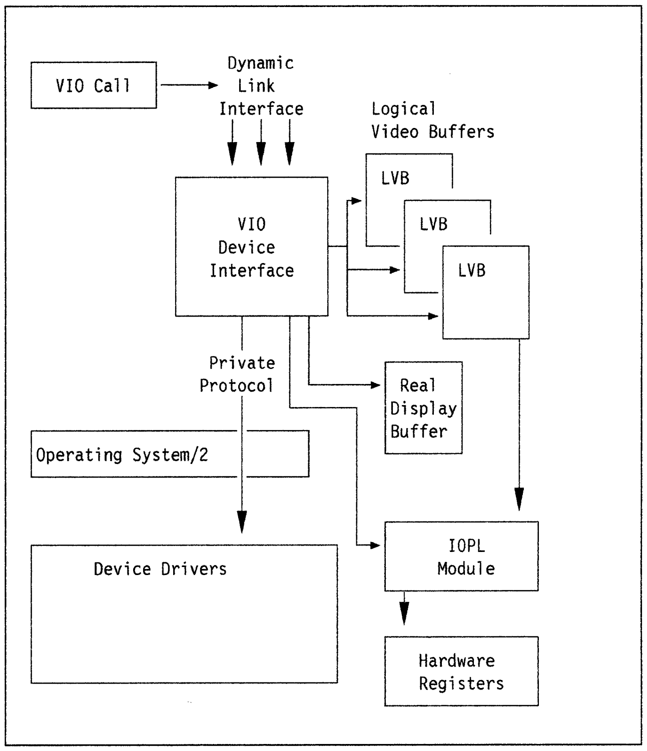


Figure 22. Display Hardware Management

Handle-based, console I/O is redirectable. Figure 23 on page 29 shows both handle-based and VIO. Operating System/2 returns handle-based I/O to a dynamic link layer, where the I/O is converted into a VIO function call. The VIO router (not shown) routes the VIO function call to the VIO device interface so the I/O can be managed in both the real and logical video buffers.

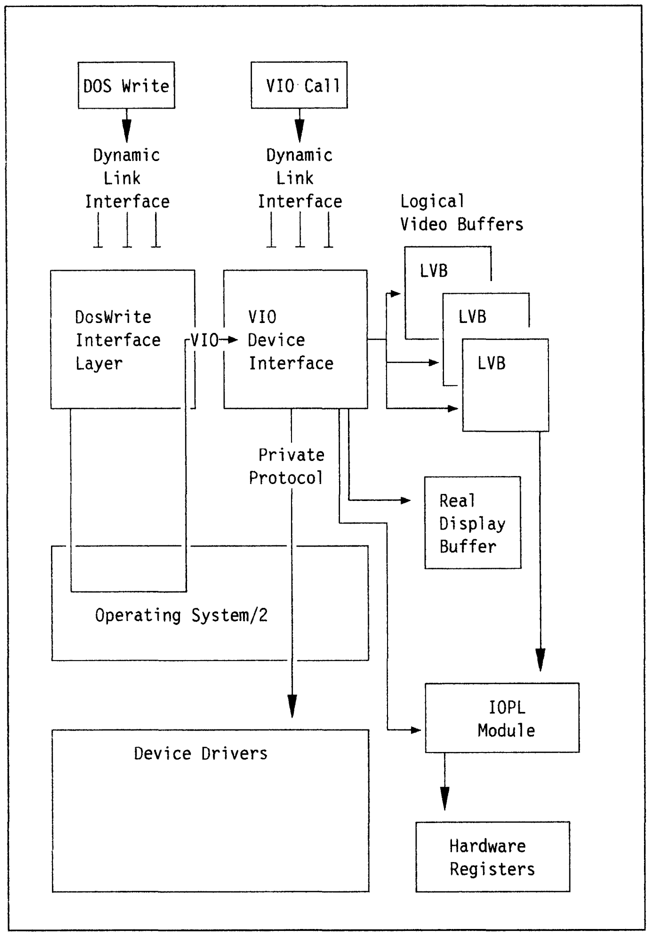


Figure 23. Both Handle-Based and VIO Device-Interface I/O

Keystroke Monitor Interface

Some applications monitor all key strokes and provide global system function before more conventional applications receive the keystrokes. Examples might include national language support for switching the keyboard layout and for Asian language input conversion. Other examples are applications that provide a desk calculator or keystroke macro expansion. Hardware-enforced protection requires the system to provide interfaces for such applications, which run as processes.

Figure 24 depicts the keystroke monitor interface. The keystroke monitors have been previously enrolled as monitors. A monitor may pass the keystroke through, consume the keystroke or replace the keystroke with one or many keystrokes.

A keyboard monitor executes at process time and can use Operating System/2 function calls. For example, a desk calculator needs to write to the display, and a keyboard macro expander may have the macros stored in the file system.

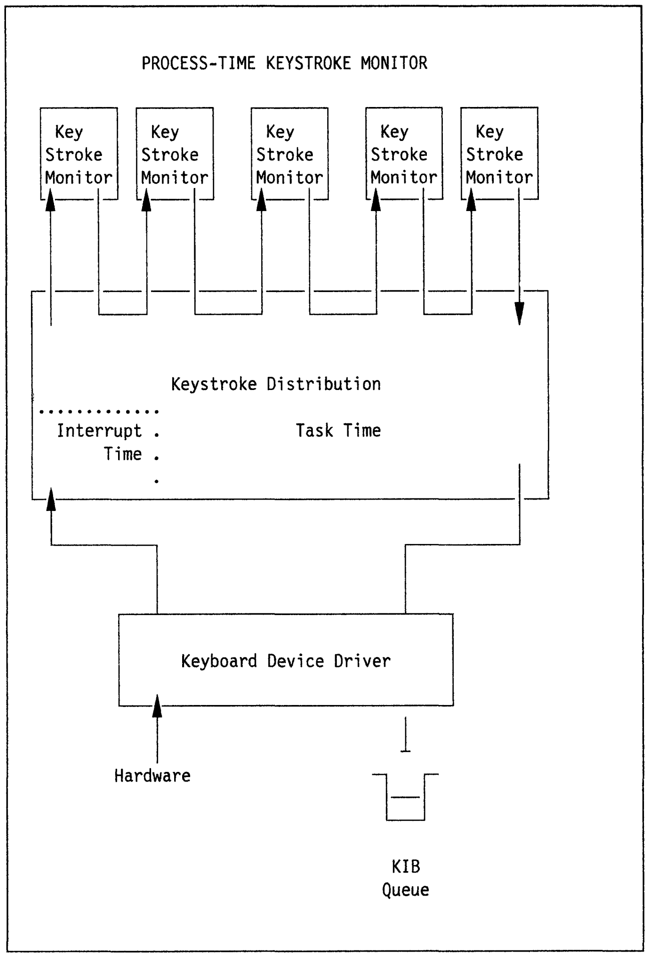


Figure 24. Keystroke Monitor Interface

Figure 25 briefly depicts the keystroke monitor interface.

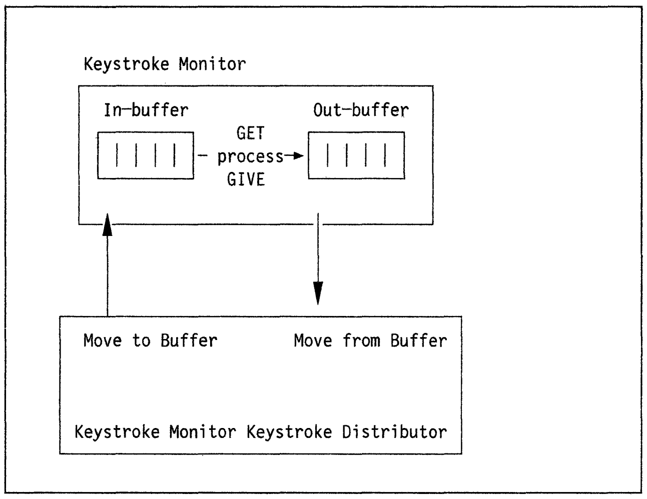


Figure 25. Process-Time Keyboard Monitor

Country Considerations

For countries outside the United States, Operating System/2 Standard Edition includes:

- National language support for system message files
- Country-dependent information
- Support for national keyboard layouts
- Double-Byte Character Set (DBCS) enabling
- Programming interfaces for country support
- Message retriever for message files
- Utility commands

National language support is provided for left-to-right languages for system messages displayed to the user.

Country-dependent information is available as follows:

- Country format information includes time, date and currency
- Lower- to upper-case character conversion tables
- Collating sequence for character sorting
- DBCS environmental vector for DBCS character determination
- Valid single-byte characters used in file names

Different keyboard layouts are provided and are selectable.

Double-Byte Character Set enabling exists for DBCS-based national languages. There are specific kernel functions for DBCS enabling for mixed single- and double-byte character strings, but full DBCS implementation and DBCS-based hardware support is not provided.

A set of programming interfaces allows applications to use the country-dependent information described above. Applications do not need to change the current country code of the system in order to use this capability. The current country code for the system is the same for all screen groups and between DOS Application Environment and Operating System/2 Application Environment.

The message retriever gives user applications a programming interface for retrieving and displaying application messages from customized message files.

Utility commands let the user select the keyboard layout and specify the system country code. The Operating System/2 Installation Aid also supports these capabilities. A message utility provides the capability to create customized message files.

Presentation Manager

Overview

The Presentation Manager provides additional function in the Operating System/2 Standard Edition Version 1.1. This function enables applications to run in a windowing environment and provides wide ranging programming interfaces for application developers, a rich set of graphics functions, support for all-points-addressable (APA) hardware, and utilities for obtaining hardcopy and for converting data within files for interchange with other systems.

The following summary is a brief overview of the main features.

- This is a major enhancement to the first release of the Operating System/2 Standard Edition, particularly in the way in which it helps users to interact with the system and with applications. Operating System/2 Version 1.1 Presentation Manager makes it possible for multiple concurrent applications to be viewed on the screen simultaneously via overlapping "screen windows."
- Enhanced ease-of-use facilities which enable users to access the services of Operating System/2 Standard Edition Version 1.1. These include windows, utilities and the Operating System/2 file system. Personal Computer applications can be easily installed, started, or stopped, with these facilities.
- The Presentation Manager provides a high-level CALL interface, known as the Presentation Manager API, which enables programmers to write applications which benefit from:
  - Standardized menus and dialogs
  - Screen windows
  - Input devices
  - Alphanumerics
  - Graphics
  - Text with typographic fonts
  - Bitmaps
- The all-points-addressable (APA) capabilities of the supported display adapters and various graphics printers and plotters are utilized.
- Users can spool and subsequently obtain hardcopy of graphics and/or alphanumeric data.
- There are utilities for use with picture files. One utility displays picture files; another can be used for conversion of data where this is necessary for files interchanged with other systems.

Additionally, the Operating System/2 Standard Edition Version 1.1 Programmer Toolkit, is enhanced to provide programmer aids and tools which include:

- A dialog editor
- Bitmap/cursor editors and generators
- A font editor
- Sample programs.

Additional Advantages

Apart from its rich function set, Presentation Manager offers three other major advantages:

- Participation with other Personal System/2 computers and host systems in IBM Systems Application Architecture
- Expanded use of the Operating System/2 supported hardware
- A base for future growth.

Consistency

In March 1987, IBM announced a collection of selected software interfaces, conventions and protocols to be published during 1987. The IBM Systems Application Architecture is the framework for the development of consistent applications across future offerings of the major IBM computing environments - System/370, System/3X, and the Personal System/2. The Presentation Manager is part of this Systems Application Architecture evolution.

Common Programming Interface

A subset of the Presentation Manager API is the Presentation Interface, which is one of the elements of the evolving Common Programming Interface of Systems Application Architecture. The initial set of interfaces provides a base on which applications can be developed and ported, with minimal change, among other IBM Systems Application Architecture operating systems.

Common User Access

The user interface of the Presentation Manager is based on the Common User Access portion of Systems Application Architecture.

The primary goal of the Common User Access is to achieve, through consistency, transfer of learning, ease of learning and ease of use across the range of IBM Systems Application Architecture applications and environments.

The Presentation Manager uses the Common User Access definition and makes it available to application developers.

Expanded Use of the IBM Personal System/2

Not all writers of applications for Personal System/2 are interested in developing applications that can be ported to other operating systems. Those whose interest lies largely in the Personal System/2 marketplace also can take advantage of the benefits offered by the Presentation Manager to utilize the Personal System/2 hardware.

In particular, the Presentation Manager exploits the display hardware through the use of bitmaps. This can be particularly useful for applications written specifically for the Personal System/2 where key considerations include fast screen update or full use of the all-points-addressable hardware. Examples of the uses for bitmaps include:

- Writing or removing menus very rapidly.
- Creating and using symbols required by the application.
- Rapidly restoring a picture when an overlaying window has been removed.
- Animation

A Base For Future Growth

Systems Application Architecture defines a foundation upon which to build portable, consistent applications systems for the future, capitalizing on IBM hardware, control programs and the Systems Application Architecture products.

The Presentation Manager is a Systems Application Architecture participant. Applications written to the programming interface of the Presentation Manager can look forward to a flourishing Personal System/2 and host systems base on which to run.

Screen Appearance

Presentation Manager enables multiple applications to be visible on the screen at the same time. It does this by displaying applications in a series of overlapping windows.

A window is a rectangular region on the screen that may be surrounded by a border. Each application displays its data in one or more of these windows. If two windows overlap, only one is actually visible on the screen at the point of overlap. The windows are arranged like papers on a desktop, where one paper can lie on top of another.

Use of multiple windows within one application allows the application to structure its displayed data in a logical way, making its user access easier to understand.

In addition to applications started by the user, the screen also can contain windows belonging to the *user shell*, a Presentation Manager application which gives access to a range of system functions. Figure 26 shows a typical screen.

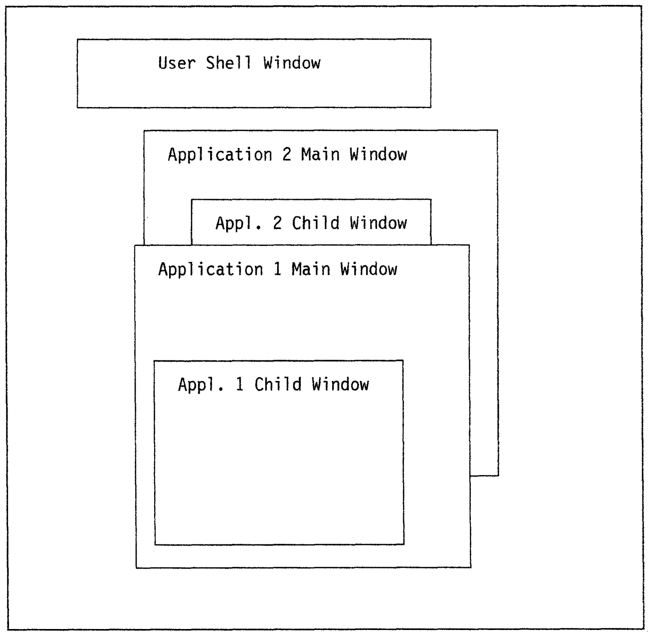


Figure 26. Appearance of Typical Presentation Manager Screen.

For systems which have a mouse attached, a *pointer* is displayed on top of the windows and the data displayed in them. The pointer is a small shape like an arrow. It indicates the point of interest and reflects mouse movements onto the screen. It can change shape as it goes from window to window, according to application requirements.

Within menus and dialogs, the user sees a kind of cursor called the *selection cursor*. This applies to systems with or without a mouse. This cursor applies to items that are selectable, such as the items in a menu, and it can be moved from item to item either by mouse movements or by keyboard keystrokes. It indicates the items in a window to which the user's next action may apply.

Use of Screen and Display Adapter

Presentation Manager only supports display adapters with APA capabilities. Presentation Manager uses the display adapters in an APA mode, typically the highest resolution mode available on the adapter, to give the best possible screen appearance.

Presentation Manager supports the following display adapters and modes, where resolution is specified in pixels horizontally and vertically:

IBM Color/Graphics Display Adapter
640 x 200, 2 color.

IBM Enhanced Graphics Adapter
640 x 350, 16 color.

IBM Personal System/2 system unit
640 x 480, 16 grayscale or 16 color.

IBM Personal System/2 Display Adapter 8514/A

A range of modes is supported, according to the options present on the adapter and the display monitor attached:

- 640 x 480, 16 grayscale or 16 color.
- 640 x 480, 256 color.
- 1024 x 768, 16 color.
- 1024 x 768, 256 color.

User Shell

The Presentation Manager User Shell provides user access to the facilities of the system. These facilities include:

Starting Programs
where the user can view a list of the programs installed on the system and start one or more of them by selecting the name in the list.

Switching Tasks
where the user can view a list of the programs or tasks that are already running and decide which one to work with next.

Window Layout
where the user can control the size and position of the windows that are visible on the screen in order to produce a suitable layout for whatever work is being done.

File System Access
where the user can view and operate on the contents of the file system. The view includes disks, directories and files, and a powerful set of commands is available for each of these. The commands also can include user-supplied functions for flexibility.

Print/Plot Control
where the user can control the printers and plotters attached to the system. The user can decide which options and devices to use and can manage the queue of print jobs for each device.

Clipboard Viewer
where the user can view data that is being copied from one place to another using the Cut and Paste functions.

Control Panel
where the user can control various system features, such as the colors used for the screen background or the mouse double click rate.

Command Line
gives equivalent command entry capability to that provided in DOS 3.30 and to provides the experienced user an alternative to a menu-driven interface. It is expected that most users will use other parts of the User Shell for the common functions and only use the Command Line for specialized functions.

The User Shell is designed to give the user easier access to the functions of the system. It makes the data and the capabilities of the system visible on the screen and provides direct and fast ways of getting work done.

Presentation Manager API Overview

Presentation Manager provides a very rich set of functions to application program writers through its Presentation Manager Application Programming Interface (API). These functions make the tasks of providing a good user access to presenting data and controlling devices as easy as possible.

The API is divided into groups of related functions:

User Dialog
used to create and process dialog boxes and menus as an important part of the application's user interface.

Screen Windows
used to create and manipulate screen windows as places to structure and display data on the screen.

Input and Messages
used to receive user input from mouse and keyboard, to receive system messages and for inter-process communication.

Alphanumerics
used to display simple text data.

Graphics used for APA display data, including lines, curves, symbols, filled areas and for typographic quality text.

Bitmaps which are in-memory representations of APA display data generally used for rapid screen updates.

Device Independence

An important feature of the Presentation Manager is the high degree of device independence it offers to the application programmer.

The systems supported by Presentation Manager offer a wide variety of devices that can be attached to the system, including:

- Mouse
- Keyboard
- Display and display adapter
- Printers
- Plotters.

An application which uses the Presentation Manager interacts with these devices in a high-level manner that does not depend on the peculiarities of the device. The Presentation Manager performs the job of mapping the high-level constructs of the API to the devices. The system is designed to take advantage of device features without making applications dependent on those features.

An example of such a feature is a printer that has a set of built-in or cartridge-loaded fonts tuned to the printer's characteristics. The Presentation Manager permits use of these fonts, but an application that uses them requests the fonts in a logical way, which does not depend on that particular printer being attached to the system. If another type of printer were attached to the system, the Presentation Manager would map the application's font request to the most closely matching font available for this other printer.

Device independence relieves the burden of writing new versions of an application for every new piece of hardware that appears.

If an application wants to use features of particular hardware, this also can be achieved. The Presentation Manager has a number of query functions that allow an application to learn about the display devices attached to the system. The application can then make full use of this data. There also are certain Escape function calls that permit an application to communicate directly to a Device Driver to obtain specialized functions not available through the Presentation Manager.

Screen Windows

Screen Windows are used to display panels of data on the screen and are the places to which the end-user directs input.

Window Types

Although a Window is essentially a rectangular area on the screen, there are several types of windows which differ in appearance and usage:

Main Windows

- Are generally long lived and really "represent" the application on the screen. An application typically has one main window, but it can have more than one if the application has more than a single main function. For example, a spreadsheet application also may have a chart function to display the data in a graphical form.

The spreadsheet can be put in one main window and the chart in another. The spreadsheet and chart can then be manipulated independently by the user.

- Are not directly related to other windows on the screen. They are positioned relative to the screen and their size is not constrained by other windows.
- Typically have a standardized appearance. This includes a number of features which occur in a standard place in the window and operate in a standard way:
  - An Action Bar and associated Pull-Down Menus
  - Wide borders for sizing operations
  - A title bar with the program and/or data file name
  - Icons for maximizing, minimizing and restoring the window's size
  - An icon for accessing a menu containing system operations
  - Optional scroll bars, if the data in the window can be scrolled

Child Windows

- Are windows that "belong" to another window, termed the *parent window*
- Typically contain data that relates to the parent window contents
- Always appear on top of the parent window and are contained it; if too large, the child windows are clipped at the edge of the parent window
- Can themselves contain child windows

Dialog Boxes

- Are transient windows containing a user dialog required to achieve a specific action
- Appear on top of a window to which the action relates (either a parent or 'owner' window)
- Normally have their contents defined using the dialog editor utility program
- Can be updated dynamically by the application program

Window Properties and Appearance

Windows have a number of properties and an appearance which are designed to aid conformance to the Common User Access definition of Systems Application Architecture.

Windows have the following intrinsic properties:

Position the position of the top left corner of the window relative to the screen or to the parent window.

Size the horizontal and vertical extent of the window. A window has three sizes:

1. its normal size. The user can change the normal size by performing a Size operation.
2. its maximum size. The user can get the window to this size rapidly by maximizing the window using the maximize icon.
3. its minimum size. The user can get the window to this size rapidly by minimizing the window using the minimize icon.

Visual Ordering
decides which window appears when two or more windows overlap on the screen.

Visibility indicates whether or not the window should be shown on the screen.

Border surrounds the window and is used to delineate the window from other objects on the screen. The border can have several forms:

- Thin border for minimal delineation
- Normal border for simple delineation of the window
- Wide border used for window sizing operations as well as delineation

Pointer Shape
defines what the pointer looks like when the pointer is in the window. By default, the pointer shape is a system-defined arrow. However, an application can choose to use a pointer shape of its own on a window-by-window basis.

Client Area
is the main area of the window where the application draws the data it wishes to display in the window.

Windows also can have a number of optional features:

Title Bar which is a narrow bar across the top of the window which contains:

- A name for the window
- Maximize/minimize/restore icons
- A system menu icon

The name is used to identify the window and its contents. For example, the main window of an editor application called FRED operating on a file called JOE.DOC would best have a name FRED-JOE.DOC.

The maximize/minimize/restore icons are used to rapidly switch the window size between maximum, minimum and normal. The system menu icon is used to access a system menu which contains a list of system functions such as move and size. Mouse users can do the same things more directly.

Scroll Bars
appear on the right side and/or bottom of the window and are used when the displayed data is too large to fit in the window and needs to be scrolled through the window to be viewed. The scroll bars contain arrows at their ends; a long rectangle in the middle holds a small rectangle called the *thumb.*

The thumb position within the rectangle indicates the current position of the visible data relative to the total amount of data. The thumb can be moved with the mouse to cause scrolling by an arbitrary amount. Selecting the arrows causes scrolling by fixed amounts in the direction of the arrows.

Action Bar
appears at the top of the window below the title bar, if there is one. The action bar is a horizontally-aligned menu containing a number of items. Each item has a pull-down menu associated with it which provides detailed choices relating to the major actions held in the action bar. The action bar thus provides a visible, easy-to-use means of doing the actions that relate to the window.

Dynamic Operations
are operations the user can perform on a window while it is displayed on the screen, including changing its size and position. The application can control whether the user can perform these operations on any window it creates. If the operations are allowed, the various user access elements relating to the operations are enabled. Otherwise, they are disabled.

Window Procedure
is a section of program code associated with a window. It has no direct relevance to the window's appearance, but it influences the way in which a window operates internally. If a window has a window procedure, the window can be treated as an *object* in programming terms.

Object-oriented programming implies that messages sent to the window are processed in a given way. This may change the appearance of the window or generate responses or messages in some pre-defined way.

An object-oriented program can be a very useful way of structuring an application. The program can be partitioned into modular sections of code, each of which is responsible for a particular aspect of the user interface or data display. It is particularly useful when the application needs to have many windows containing the same kind of information, because all similar windows can share the same window procedure. This is the way in which most of the user access controls (for example, the scroll bars or the push buttons) are implemented within the system.

A window with all the visual features described in this section has the following appearance:

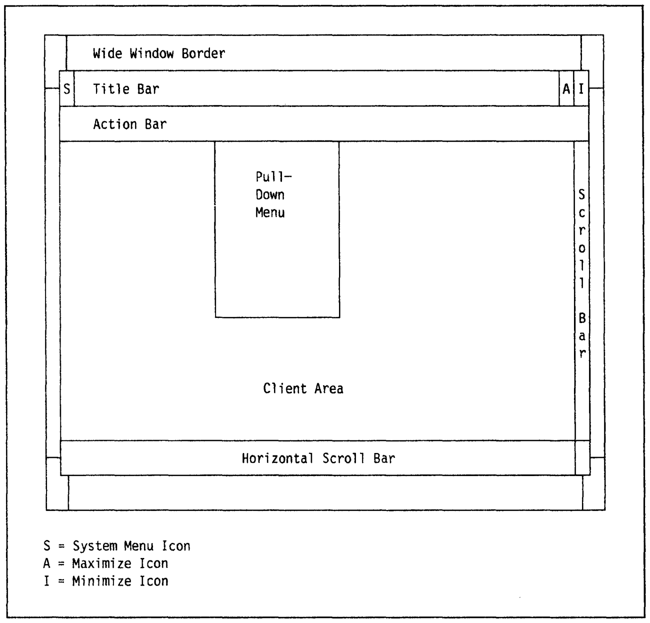


Figure 27. Default Window Appearance.

The window appearance shown in Figure 27 is a default appearance which is produced very simply using Presentation Manager functions and conforms to the Common User Access of Systems Application Architecture. However, an application which needs a specialized window appearance to meet some specific requirements, for example, a split-client area showing two pieces of data each with its own pair of scroll bars, can do this using the Presentation Manager.

User Access Functions

Presentation Manager provides a series of functions that enable the application user interface to be designed for ease-of-use and consistency. This is a very important area of function for almost all applications, and Presentation Manager provides a rich set of functions which should meet most needs. The Presentation Manager User Shell itself uses these functions.

Common User Access Definition for Applications

Systems Application Architecture and Presentation Manager encourages all application developers to use the Common User Access. This offers advantages to both programmers and end users.

The benefits to the programmer are:

- A toolkit which makes producing user interface elements straightforward.
- Not worrying about which keystrokes or mouse actions to use to achieve a given result - - these are defined by the Common User Access.
- Not having to make all parts of an application have the same look and feel.

The end user benefits from:

- An interface that is easy to learn and easy to use, but gives expert users rapid access to functions
- Reduced learning time. Once the user knows one application that conforms to the definition, it is easy to move to another application that uses it because all applications using the definition present and operate their functions in a similar way. The user can concentrate on the function offered by the application rather than on the way it is presented.
- No sudden breaks or jumps in the way things work when using multiple applications simultaneously with Presentation Manager. Applications and the Presentation Manager system all have the same look and feel.

Menus, Dialogs and User Interface Controls

Presentation Manager's user interface functions are divided into two broad areas:

Action Bars
which are simple selection menus.

Dialog Boxes
which are windows containing a series of *user interface controls* offering dialog that is free-format and more complex than a simple selection menu.

Action Bars: An action bar is a menu containing simple selection items. The user sees a list of text items and selects one from the list to cause some action.

The menu consists of two parts. The first is the action bar itself, which is a horizontally-aligned list of items, with the first item at the left. The action bar is positioned at the top of a window, just beneath the window title bar. Most of the main, commonly used functions relating to that window should be placed in the action bar.

The action bar is often a single line of items, but if there are too many items to fit on one line within the window, the action bar is split into two or more lines placed one above the other.

The second part of the action bar menu consists of the pull-down menu associated with each action bar choice. Selecting an item in the action bar causes its associated pull-down menu to appear. Selection of the action bar item does not cause any other action to occur.

A pull-down menu is a vertically aligned list of items. Each pull-down appears directly below the item in the action bar with which it is associated. The items in the pull-down represent either immediate actions which take place when the item is selected, or the start of a longer dialog as the appearance of a dialog box.

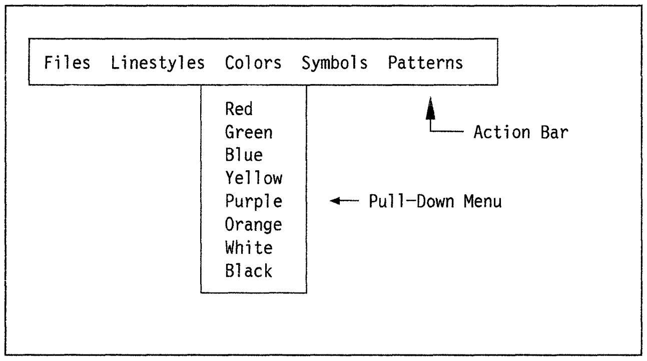


Figure 28. A Typical Action Bar and Pull-Down Menu

Dialog Boxes: A dialog box is a window that contains a series of user interface controls arranged in a free-format layout. A dialog box supports a user dialog functionally richer than that offered by an action bar; in particular, it can support more than simple selection fields. The dialog box is normally defined using the dialog editor program, which is a WYSIWYG (What You See Is What You Get) editor that allows the application writer to see the dialog as it is created.

The user access controls that can be placed within a dialog box are:

Push Buttons
which consist of some text surrounded by a round-cornered rectangle. The button is normally two-state: it is selected or not-selected. This is indicated by highlighting when the button is selected. Buttons often indicate actions that are performed immediately when the button is pressed.

Check Boxes
are used to indicate selection of text items. Selection is indicated by a check mark in a small box that appears to the left of the text. Check boxes are used when multiple selections are allowed.

Radio Buttons
are used to indicate mutually exclusive selections of text items. Only one of a set of radio buttons can be selected at once. Selecting another item results in the currently selected item becoming unselected.

List Box
is a vertical list of text items held within a rectangular area. The items are selectable. If the list of items is longer than the rectangular area can display at one time, a scroll bar appears at the right of the area. This allows the user to scroll through the list as required.

Input Field
is a single-line horizontal area where the user can type in some alphanumeric data from the keyboard. It can have some initial text displayed, which can be overtyped or modified by the user.

Scroll Bars
are similar to the scroll bars used for scrolling data through windows. However, they can be used for purposes different from this, in particular the thumb position can be used to indicate the value of a linearly-varying property of some kind, such as the brightness of a color.

User-Interface Controls: The various user-interface controls are available to the application for use in any window, not just for a dialog box. This can be very useful when an application needs to have simple user interaction with application data. For example, an application displaying a window with some graphical data may need a couple of push button controls at the bottom of the window to allow the user to view the next piece of data or to exit the viewing process.

Input

Input functions concern:

- reception of input events from the mouse and keyboard generated by the user
- reception of messages concerning system events and conditions
- the sending and receiving of messages between tasks

The User's View of Input

The end user communicates with the system using the mouse and/or the keyboard. A mouse, however, allows the user to interact more easily with objects on the screen.

When communicating with the system, the user sends input to one window at a time. The window is called the input focus, while the application to which the input focus belongs is called the active application.

The user can generally decide to change the input focus and the active application at any time. This is done either by a special keystroke or by moving the mouse to put the pointer in another window and then clicking the mouse selection button. There are also functions in the user shell allowing the user to change the active application.

Messages and Message Sources

The basic element of input is the message. Each input event is delivered to the application as a message. Each message contains data that identifies the type of message involved and conveys the information related to the message. Message originate from a variety of sources:

Mouse which involves user input when the mouse is moved and when mouse buttons are pressed and released.

Keyboard which generates messages when the user presses and releases keys.

System Messages
these relate to events occurring in the system generally as the side effect of some direct user actions. They include the following messages:

- Size, if the user changed the size of a window
- Move, if the user moved a window across the screen
- Redraw, if some user action caused part of a window to be revealed that was not previously visible. The data in this part of the window must be redrawn.
- Clipboard, if the user used the cut and paste functions of the system to copy data from one window to another.

Timer An application can set one or more timers running. When one of these expires, a message is sent to notify the application.

InterTask Messages
Tasks in the system can send one another messages of a general nature. Their meaning is defined by the applications using them.

Semaphore Messages
An application can wait for an Operating System/2 semaphore message for some event to occur. Semaphore messages permit the application to tie such events to other messages in the system.

Note that mouse, keyboard and system messages all relate to a window. All identify the window (by means of its handle) in the data passed with the message. The other messages are not related to a window.

How Applications Receive Messages - Message Queues

Whatever messages are destined for an application, they are first placed in an *input queue*. The application has to allocate a queue when initializing the Presentation Manager. It is possible for an application to allocate more than one queue, but to do this the application must be multitasking since it is only possible to read input from one queue on a single task thread.

The Presentation Manager input queue structure has two levels. There is a *system queue* for all events directly received from the user - - mouse and keyboard messages. Each application also has its own application queue that holds messages directly relating to that application alone, such as timer messages. Application queues do not hold user input. The system queue is shared between all applications in the system and, in principle, an event could go from the system queue to any application.

Within both queues, the messages are ordered in time sequence.

The application receives input by making a *Get Message* or *Peek Message* function call. When calling Get Message, the application is suspended until a message arrives. When a message arrives, the application resumes and the message is passed back on the completion of the Get Message call.

When an application asks for input using the GetMsg function, it receives the following:

- the first message in the application queue; or,
- if the *application queue* is empty and the application has the input focus, the first message in the *system queue*; or,
- if neither of the first two apply, a redrawing is required, a redraw message; OR,
- if no message is available, the application waits within the GetMsg function for a message.

Receiving Input - Recommended Application Style

Any application that creates an input queue should expect to receive input messages from the user. To do this, the application should issue calls to Get Message or Peek Message regularly, which means responding to user input within a short time, from 0.1 to 0.5 seconds. If the response is longer than this, the user will notice the delay and the smooth flow of work will be interrupted.

Not all application functions will be able to complete within this time period. Some operations, particularly those involving heavy computation or disk access, take much longer. In these circumstances, the application should have separate tasks for responding to user input and for doing work that takes considerable time. The user response task can then continue to service user input messages while the other task finishes its work.

At the very least, the user response task should indicate to the user that the application is busy and allow the user to switch tasks away from that application to another one that is not busy.

Alphanumerics Output

Alphanumerics output functions are provided for the fast output of simple textual data. The text is displayed in a presentation space which is a fixed grid of equal-sized characters with limited flexibility and quality. This function of the Presentation Manager is called Advanced VIO, or AVIO.

Migration of Non-Windowing Text Applications

The first set of functions provided by the AVIO interface is the support of the Operating System/2 VIOxxx, KBDxxx, and MOUxxx functions within the windowing environment. Applications that access the display and input devices only through these functions in the non-windowing environment of Operating System/2 will, with some minor exceptions, be able to run in a single default window within the windowing environment without the need for any change.

Exceptions concern the use of functions that directly access the screen in some some way, such as VioGetPhysBuf. These cannot be supported in the windowing environment.

Text in Multiple Windows

The AVIO interface extends the VIOxxx functions to enable an application to display text output in multiple windows. Note that MOUxxx and KBDxxx functions are not supported in this situation. Their functions are provided by the input functions of Presentation Manager, which should be used instead.

The AVIO interface also extends the VIO interface to provide IBM 3270-style alphanumerics function. The AVIO interface supports the CGA format of presentation space, with each character having a single attribute byte to control features such as color. It also supports an alternative format which has two attribute bytes per character.

The alternative format allows for:

- Underscoring
- Strike-through
- Use of up to four application-loaded fonts,

in addition to the attributes supported by the single attribute byte format.

The AVIO functions allow the presentation space to be updated by direct updating of the rectangular character buffer or by text string functions.

Graphics Output

The graphics output functions provided by the Presentation Manager are a rich set for drawing APA graphics. The functions themselves follow the Systems Application Architecture making the interchange of graphics data with other IBM systems straightforward.

An important part of the graphics output function is devoted to supporting typographic quality text, which can be mixed with graphics as required.

Graphics Primitives

The following graphics primitives are supported:

Lines of various kinds:

Line a straight line between two points

Polyline successive straight lines drawn between an array of points

Box a rectangle with optional rounded corners

Arc/Full Arc all or part of a circle or ellipse. Ellipse major axis can be at an arbitrary angle.

Pie a pie section - an arc whose end points are joined to the arc center point by straight lines

Fillet a succession of curves based on conic arcs

Spline a succession of curves similar to a fillet, but with more elaborate variations possible.

Filled areas are defined as a boundary made up of connected line sections filled with a pattern.

Character strings are arrays of characters

Images are rectangular arrays of pixels. Compressed images are not supported.

Markers are symbols used to label or mark points, for example on a graph.

Graphics Attributes: Each type of primitive has a set of attributes that can modify its appearance. Some of the attributes are general and apply to all primitives:

Color is the color in which the object is drawn.

Background color is the color in which the background parts of some objects are drawn. For example, a character has a foreground that represents the shape of the character. It also has a background that is the rest of the rectangle which forms the character cell.

Mix is the way in which a primitive interacts with previously-drawn primitives when it is drawn. For example, it can overpaint a previous drawing so that only the new primitive shows through.

Background mix is the way in which the background part of a primitive interacts with previously drawn objects.

Transformations are coordinate transformations that can be used to scale, rotate or translate primitives before they are drawn. A number of transformations can be used. Some operate on the whole picture, others only on subsections of the picture.

Clipping is used to restrict the region of the output device in which drawing occurs. Parts of primitives that fall outside the clipping region are not drawn.

The following attributes are specific to types of primitives:

Lines have:

- Linestyle - dot/dash patterns
- Line width - varying thicknesses of line
- Line ends and joins - the form of the ends and joins of thick lines

Filled areas have:

- Variable fill patterns

Character strings have:

- Character Set/Font - a variety of character styles can be used
- Cell Size - the size of characters
- String Angle - the angle of the string to the horizontal
- Precision - the accuracy with which the characters are placed.

Images have no specific attributes.

Markers have:

- A choice of marker shape
- Marker size
- Precision - the accuracy of placing of the markers

Typographic Text Capabilities

The Graphics functions include capabilities for generating text output with typographic quality using the character string functions. A wide variety of font typefaces and styles can be supported, and a set of fonts is supplied with the Presentation Manager for the devices it supports.

In addition to simple character strings, which simply place one character after another according to the character metrics in the font, there is a function that produces a line of text in which each character can be individually positioned. This enables an application to produce text output on the screen that closely matches the output from a printer, for example.

A variety of services enable an application to take full advantage of typographic quality fonts:

- The API enables use of fonts supplied with the Presentation Manager as well as fonts available on specific hardware devices such as printers.
- The application can specify the font(s) it requires in a logical way. This allows the system to perform a match to the best font(s) available on whichever output device the text is drawn. This also makes it easier to transmit the document to a remote system since the it may not have the same precise set of fonts available.
- Query functions enable the application to find out the metrics and other attributes of every font. These can be very important when considering certain types of applications, such as page layout.
- The system is extendable. More fonts can be added to the system to give a wider range of typeface styles, for example:
  - The font file format is published
  - A font editor utility is provided for the production of image fonts
  - Applications can request that any available font be loaded into the system and used.

Presentation Manager supplies a set of fonts for the devices it supports. These include both image-defined fonts and also vector or outline fonts. The latter enable a much wider range of point sizes to be supported across all devices. The former offer better performance and better appearance at small point sizes and on lower resolution devices such as most displays.

Graphics Processing

The graphics processing performed by Presentation Manager is not just confined to drawing primitives on the screen. A much broader range of function is supported.

Drawings can be sent to a range of output devices, such as printers and plotters, in addition to the display screen. Output to a file for storage or for interchange with other applications or systems is also supported.

The graphics API supports two other very useful functions:

Correlation
otherwise known as *picking*. This enables an application to determine which primitives cause drawing within a defined rectangle. This is useful when the user selects some item on the screen by pointing with a cursor. The rectangle used is a small rectangle around the active point of the cursor, and the application uses the Correlation function to determine which item the user is pointing at.

Bounds enables an application to find out the smallest rectangle that will bound some set of primitives which it has drawn. This can be useful for determining the size of some complex graphical object, particularly when that object needs to fit into a picture with other objects such as text.

The application can draw various amounts of data with a single function call across the API:

- One primitive, such as a single line or a character string.
- A whole group of primitives and associated attributes.
- Part or all of the Graphics Store. (See the description of the Graphics Store Function.)

The final aspect of the drawing process is the set of resources that are used during the drawing of graphics. These include:

Default attributes
are values for graphics attributes that are not explicitly set by the application during the drawing process.

Character sets/fonts
are definitions of the symbols used when drawing character strings. Many of these can be available at the same time.

Linestyle sets
are application-defined linestyles, used for specialized linestyles not directly supported by Presentation Manager.

Color tables
are definitions of the colors required by an application, should the system defaults not be adequate. These are normally mapped to the closest available colors on the target output device.

All of the resources can be changed, added to or deleted from by application callable functions.

Graphics Store Function

With Presentation Manager, graphics pictures can be handled in two alternate ways: *non-retained* and *retained*. These terms relate to the way in which the system holds the picture for drawing:

Non-Retained Graphics
In this mode, primitives are passed from the application via the API and are simply drawn on the output device. The system has no memory of the primitive once it is drawn.

In this mode, much of the work of drawing a picture is done by the application. The application must have code to draw everything it wants to output on the device by issuing the appropriate API calls.

Retained Graphics
In this mode, the application builds up a picture in an area of system storage called the *graphics store*. It can do this by issuing appropriate API function calls, which are generally the same as those used to draw lines, etc. in non-retained mode.

Once the picture is built, the application can cause all or part of the picture to be drawn by issuing a single function call.

The advantages of using retained graphics mode are in the application are:

- The picture in the picture store can be built in a series of graphics segments, that allow the picture to be structured. For example, if a picture of a car is built up, the primitives representing a wheel need only be placed in the store once, in one graphics segment. The four wheels on the car can then be drawn by referencing the wheel segment four times from the overall car segment.
- Picture editing is made simpler, since the data is present in a convenient data structure and a series of editing functions are provided in the API.
- Because the picture is contained in the system rather than the application, the system can (if requested) automatically redraw the picture when necessary, for example, when a windowing operation generates a Redraw message for the window in which the picture is shown.
- Application code and data space is reduced, since the system can take over the burden of holding and drawing the picture. This is particularly useful if some or all of the picture data is obtained from a picture file on disk, since the API provides functions to load data directly from a disk file into the Graphics Store, requiring minimal application involvement.

Bitmaps

A bitmap is basically a memory representation of the data displayed on an APA device. It is normally used to perform fast updates to a display device, but a bitmap can be matched to any APA device.

The bitmap can be a color bitmap, with the same color capabilities as the matched device, or a mono bitmap. Mono bitmaps are used to save storage when the data drawn into the bitmap does not have a requirement for multiple colors.

Data can be drawn into a bitmap as if it were an output device like the screen. Both graphics and alphanumerics data can be drawn into the bitmap.

The basic operation that is performed with a bitmap is to *copy* a rectangular subsection of the bitmap to another bitmap or to a device like the screen. A third "modifier" bitmap may be used for masking purposes, and a full 256 logical mix combinations are supported for this function.

The application can also access the pixel data in the bitmap directly; i.e., it can get and put blocks of pixel data. However, when using these functions, the data format is device dependent and the application must be written with this in mind. Query functions can inform the application which format is involved for a particular bitmap, but no assumptions should be made which expect a specific format unless the application is designed to work only with a particular device.

Device Driver Interfaces

Presentation Manager supports an open architecture. It is structured so that support for new or different input or output devices can be provided easily.

This is achieved by providing the device support code in a package called a device driver. The device driver communicates with the main part of Presentation Manager via a device driver interface, which can support a range of device drivers.

Device drivers are supplied for:

- Input devices, including the mouse and keyboard.
- Output devices, including displays, printers and plotters.

The input device drivers are supplied with all versions of Operating System/2. The output device drivers only apply to Presentation Manager.

The output device driver interface of Presentation Manager is designed to support different devices with a wide range of capabilities. It is flexible so that it can support a relatively simple device such as the IBM Color/Graphics Display Adapter, but it can also support and utilize output devices with powerful built-in functions such as the IBM Personal System/2 Display Adapter 8514/A.

An appropriate device driver must be written and installed if Presentation Manager is to use a new input or output device. The device driver interfaces for input and output devices are documented and enable hardware developers to get support for new devices.

Toolkit

The Presentation Manager provides a toolkit to aid in the development of applications which use the Presentation Manager. It contains a number of components:

Dialog Editor
is a What You See Is What You Get (WYSIWYG) editor that enables the creation of dialog boxes. It allows the application writer to define the user interface controls to use in the dialog box and to position them as desired.

Once defined, the dialog box definition can be compiled and saved in a resource file for inclusion in the application program.

Bitmap/Cursor Editor
is an editor which enables the creation of bitmap, cursor/pointer and icon objects. It allows the application writer to define the pixel colors within a rectangular array.

Once created, the objects can be stored in a disk file which can be made into resource files using the resource compiler.

Font Editor
is an editor which enables the creation of an image font object. It allows the user to define each character of the font in turn.

Once created, the font can be saved in a disk file and made into a resource file using the Resource Compiler.

Include Files
is a set of files to provide definitions for the API functions and data structures needed when writing an application in the supported languages.

Resource Compiler
is used to produce resource files for linking with an application. Input is typically in a source format that is generally a form of readable English text.

Sample Programs
is a set of example programs which use the Presentation Manager. The set is designed to illustrate various aspects of the API which are relevant to the programmer who wants to develop an application using the functions provided by the Presentation Manager.

IBM Operating System/2 Extended Edition

Introduction

The IBM Operating System/2 Extended Edition incorporates all of the functions of the IBM Operating System/2 Standard Edition Version 1.1 with communications support and relational database management. The Operating System/2 Extended Edition is designed to provide a broad and stable base for the development of contemporary productivity applications. The operational environment allows applications to function in both standalone mode and as part of a comprehensive system solution across a network. In addition, the Operating System/2 Extended Edition strives for consistency and simplicity in the interfaces designed for both software developers and end users. At the same time, it will allow many existing IBM Personal Computer Disk Operating System (DOS) applications to run unchanged in a single application environment.

IBM Operating System/2 Communications Manager

The IBM Operating System/2 Communications Manager portion provides comprehensive communication capability in a single system. Thus, functions available in various communications programs for DOS now operate with the additional benefits of multitasking, expanded memory, common displays and enhanced usability. This innovative software package provides greater flexibility in aligning communications requirements to the growing needs of a business.

The Communications Manager provides access to information located in a variety of other local and/or remote systems. Other systems may be IBM Personal Computers, IBM Personal Systems/2, an IBM System/36, an IBM S/370, and even a non-IBM data servicer. This flexibility permits businesses to receive, analyze and make decisions on an expedited basis to respond to the needs of a fast-paced world.

Productivity is enhanced through multiple concurrent communications connections. No longer does a user have to wait for one type of communication session to complete before loading another. Thus, the user of Operating System/2 has the ability to do more than one thing at a time. Productivity may also be enhanced through network management. In a System/370 host network, Communications and Systems Management (C&SM) alerts are issued by the Communications Manager to enable efficient management of communications on the network.

The Communications Manager also allows flexibility in writing applications to multiple programming interfaces. This helps software developers to be more efficient and productive in effectively meeting users' needs.

The Communications Manager is written for the Operating System/2 Applications environment. Services include communication to other IBM Personal Computers and systems over a wide range of local and remote connectivities including Synchronous Data Link Control (SDLC), Distributed Function Terminals (DFT) mode to an IBM 3174 or 3274, IBM Token-Ring and PC Network Local Area Networks (LANs) and asynchronous (Async) links. Protocols used include LU 6.2, 3270 data stream (LU 2) and asynchronous communications.

Other functions include keyboard remapping, file transfer and concurrent emulation support for multiple terminal types. Programming interfaces are available to help migrate existing programs, facilitate programmer productivity in application development and allow programs to take advantage of the power of an IBM Personal Computer or IBM Personal System/2.

Communications Manager Highlights

The Communications Manager provides terminal emulation and file transfer utilizing the added intelligence of the IBM Personal Computer or IBM Personal System/2 and its other applications. Both synchronous and asynchronous terminals can be emulated.

IBM 3270 Terminal Emulation

The following IBM 3270 terminals may be emulated: IBM 3178 (Model 2); IBM 3278 (Models 2-5); IBM 3279 (Models S2A and S2B). All base data-stream functions are supported, as are the multiple interactive screen, extended attributes, extended data stream (including seven colors and extended highlights), file transfer and emulator keyboard remapping.

IBM 3101 and DEC[15] VT100 Emulation

An IBM Personal Computer or IBM Personal System/2 connected to a host supporting an asynchronous link may emulate the IBM 3101 (Model 20) or the DEC VT100 terminal. Lines may be switched, non-switched, or directly-connected, compatible with 1984 CCITT V24/V28 (RS232C) as implemented by IBM. Keyboard remapping gives each user the flexibility of personalizing the use of the keyboard. Emulation provides facilities to access data services such as Dow Jones News/Retrieval Service[16] ; Compuserve Information Service[17]; MCI Mail[18]; or The Source[19].

File Transfer

File transfer capability allows text or binary files to be moved to and from an IBM host, IBM Personal Computer or Personal System/2 or another supported non-IBM host. The 3270 PC File Transfer Program works with both 3270 and asynchronous emulation. XModem and Pacing protocols operate with asynchronous emulation. Pacing is not required when receiving an ASCII file from a host.

Multiple, Concurrent Communications Connectivities

The IBM Personal Computer or IBM Personal System/2 using the IBM Operating System/2 Extended Edition may be attached to another IBM Personal Computer, IBM Personal System/2, a departmental system, or to a System/370 family host. Connectivity may be local (DFT or Token-Ring) or remote (SDLC or asynchronous). Connectivities are under the control of the specific architectures and protocols shown in Figure 29 on page 51. Data streams and other protocols that allow the IBM Personal Computer or IBM Personal System/2 to communicate as an emulated terminal or directly through a program-to-program interface are illustrated in Figure 30 on page 52.

Concurrency is a key to productivity. From a communications standpoint, multiple protocols, terminal emulation, program-to-program support and programming interface support can operate concurrently. Thus, communications can be active while the user is doing another application. Concurrent communications are dependent upon the capabilities of the adapters in the IBM Personal Computer or IBM Personal System/2. (See note 1 of Figure 30 on page 52 for some limitations.) Supported SNA links can be shared by applications, which may use up to five 3270 SNA display and/or multiple LU6.2 sessions.

Network Management

Network management support (for the IBM System/370 host network) includes: Communications and Systems Management (C&SM) alerts for SDLC, Async, Token-Ring and PC Network data links, and problem determination data. Both Async and PC Network require an IBM SDLC or Token-Ring link to communicate alerts to the host. The Communications Manager sends alerts to the host when errors are detected for SDLC, Async, Token-Ring or PC Network connections. Applications written to the Communications Manager's programming interfaces can take advantage of the alert sending capability.

Problem Determination

The Communications Manager provides functions for gathering and processing problem determination data. These functions include tracing programming interfaces, data units, and/or system events; displaying and printing all or selected error logs from file; system dumping; and displaying all or selected messages.

Subsystem Management

The Communications Manager allows a system administrator to control and gain status information about the SNA communication resources maintained by the system. As a subsystem manager tool, it displays information about which programs are being used and which sessions are being used by the programs, and detailed information about the sessions and other active resources. It allows the activation or deactivation of the sessions, data link controls and specific links.

[15] Trademark of Digital Equipment Corporation

[16] Trademark of Dow Jones & Co. Inc.

[17] Trademark of Compuserve Inc.

[18] Trademark of MCI Communications Corp.

[19] Service mark of Source Telecomputing Corp.

IBM Operating System/2 Database Manager

The IBM Operating System/2 Database Manager supports the relational model of data, in which data is structured in the form of simple and easy-to-understand tables. Data definition and manipulation are supported by the Structured Query Language (SQL), used in IBM's relational database management systems DB2 and SQL/DS on host computers. The Database Manager provides a common interface for users and software developers on host systems and personal computers, facilitating application portability across systems.

The Database Manager is designed to provide a high degree of independence and ease-of-use characteristics for database design, creation and access. The simple data structure allows a user to specify what is needed, as opposed to how to obtain it.

Data integrity, security and concurrency controls in the Database Manager allow for multi-application access to a database.

End-user tools for user query and report generation, as well as application creation, are available. The Database Manager also includes an SQL Application Programming Interface for software developers.

Database Manager Highlights

The Database Manager incorporates a powerful relational database with end-user tools and application-creation tools. Neither the user nor the application program has to understand complex physical data structures and access methods.

Data Control and Protection

The Database Manager includes controls for transaction management, system and media recovery, database security and concurrency.

The COMMIT and ROLLBACK features help to ensure that a database will be properly updated by an application transaction. Recovery using COMMIT and ROLLBACK is automatically invoked when restarting after a system failure. Backup utilities protect against loss of data due to media failures.

The Database Manager provides password protection for database security. It also includes record-level or table-level locking to allow concurrent application access to a database. This feature ensures that updated transactions are written to the database before another update transaction can read the data it intends to update.

Data Types/Storage

The Database Manager supports data types that include integer, floating point, packed decimal, fixed- and variable-length character strings, date, time and timestamp. The maximum table size is limited only by the amount of fixed disk storage available. A database must reside completely on a single logical fixed disk (32MB maximum) or diskette.

Utilities

The Database Manager provides a number of utility functions in support of user database operations:

- Import - provides conversion from Operating System/2 files in other formats to an existing database manager table
- Export - provides conversion from a database manager table to an Operating System/2 file in another format
- Backup - backs up an entire database or the changes made since the last backup to fixed disk or diskette(s)
- Restore - restores a database that was backed up using the backup utility
- Unload - saves a single table onto a fixed disk or diskette(s)
- Load - restores a single table that was saved using the Unload utility
- Reorg - reorganizes a table in user-specified order to provide more efficient processing
- Runstats - updates statistics about the physical characteristics of a table or its indexes (used by the system to determine the most effective way to access data)

Query Manager

The Database Manager includes functions that allow the user to define, update and query data, as well as prepare reports. Data definition enables the user to create and delete tables, views and indexes. A data entry/edit facility allows data insertion, update and deletion of rows within a table.

A query facility enables the user to generate SQL queries to retrieve data from database tables. Either a prompted or a command interface may be used to generate a query.

A report generator allows the user to prepare a customized, formatted report using data generated from a database query. A customized report format can be displayed, printed or saved.

Application Creation Tools

Panels, menus and procedures may be used to create customized and standard applications. The panels facility allows the user to develop customized display screens that can be used for data entry, search and update. A complete, interactive database application can be developed around these panels, using menus and procedures.

The menu facility allows the user to define a database application selection menu. Menus may be used to allow the user to run predefined queries, procedures, panels or another menu.

A procedure allows a user to store a sequence of statements that can be invoked using a single command. A procedure can invoke a query, menu, panel, report or another procedure.

Programming Interfaces

Programming interfaces are incorporated into the Operating System/2 Extended Edition to help increase productivity and simplify writing applications. The Extended Edition interfaces serve one or more needs: migration from existing DOS applications, common user access, common communication support, and consistency of database applications across the family of products supported by IBM Systems Application Architecture.

Several programming interfaces are accessible through the Operating System/2 calls. Previously written programs must be recompiled or revised if they are to migrate to the supported DOS interfaces. The supported programming interfaces are described in the following text.

Structured Query Language (SQL) API

The Operating System/2 Extended Edition includes an SQL Application Programming Interface (API), in addition to application creation tools. This interface is an extensive subset of that announced as part of the common programming interface of the Systems Application Architecture. The SQL is a powerful, high-level data definition and manipulation language. Users can imbed SQL statements in IBM C/2 language source programs. The imbedded SQL statements are converted by a precompiler for subsequent application program compilation and execution.

Advanced Program-to-Program Communications Interface (APPC)

This programming interface implements the LU6.2 architecture and is designed to save the programmer from the detail and complexity of communications links. It controls conversation with the remote partner. It also controls communications services, and it supports both mapped (data stream independent) and basic (data stream dependent) verbs. Applications that use APPC may be written to IBM host computers with MVS/CICS and VSE/CICS; System/36; System/38; System/88; another Personal System/2; IBM Personal Computers; the IBM Personal Computer RT; and Series/1 systems. It provides the function of the Advanced Program-to-Program Communication Program that is available today with DOS but modified to take advantage of the Operating System/2 calls. Any existing programs written to this interface on DOS will need to be revised. This interface supports the Macro Assembler/2, Pascal Compiler/2, and C/2 compilers.

Server-Requester Programming Interface (SRPI)

This programming interface is part of the Enhanced Connectivity Facilities (ECF). It enables the writing of communications-independent, requester program calls to a server program. It is supported over links using the LU2 protocol. Host server support is available under MVS/TSO and VM/CMS. The same function is provided in the IBM PC 3270 Communications Family Programs. Any existing programs written to this interface on DOS will need to be recompiled or revised. This interface supports the Macro Assembler/2, Pascal Compiler/2, and C/2 compilers.

Asynchronous Communications Device Interface (ACDI)

This programming interface is designed to achieve a high degree of asynchronous hardware independence to allow applications to exchange data over asynchronous links. Device-specific programming modules are required for each supported device type and are included in the

Operating System/2 Extended Edition. Supported functions include the ability to manipulate the line characteristics and connection control (connect and disconnect) without having to deal with physical device-specific characteristics. This interface supports the Macro Assembler/2, Pascal Compiler/2, and C/2 compilers.

IBM LAN Interfaces

The IBM NETBIOS and the IEEE 802.2 data link control interfaces are provided for communicating across the IBM LANs. Applications already written to these interfaces for DOS will need to be revised. This interface supports the Macro Assembler/2, Pascal Compiler/2, or C/2 compilers.

IBM Operating System/2 Extended Edition Planned Enhancements

Planned enhancements to the Communications Manager will include SNA LAN gateway support, ECF enhancements, 5250 workstation feature, X.25 support and a high-level language 3270 program interface. Applications written to the Entry Emulator High-Level Language Application Program Interface (EEHLLAPI) of the IBM PC 3270 Emulation Program, Entry Level 1.1 can be recompiled to work on the Operating System/2 Extended Edition. The 3270 data stream will be enabled for double-byte character set language translation.

Planned enhancements to the Database Manager will include Remote Data Services to provide support for an IBM Personal Computer or Personal System/2 on Token-Ring or PC Network. This function will allow multiple workstations to access a common database and a single workstation to access a database located elsewhere on a LAN.

Other enhancements to the Database Manager include IBM Pascal/2 and IBM COBOL/2 precompiler support for SQL statements embedded in application programs and the Import Utility support of non-delimited (ASCII) flat files to assist in exchanging data with other applications. Enhanced support for fixed disks to support partitions, as well as application development facilities for the Dialog Manager interface, also are planned.

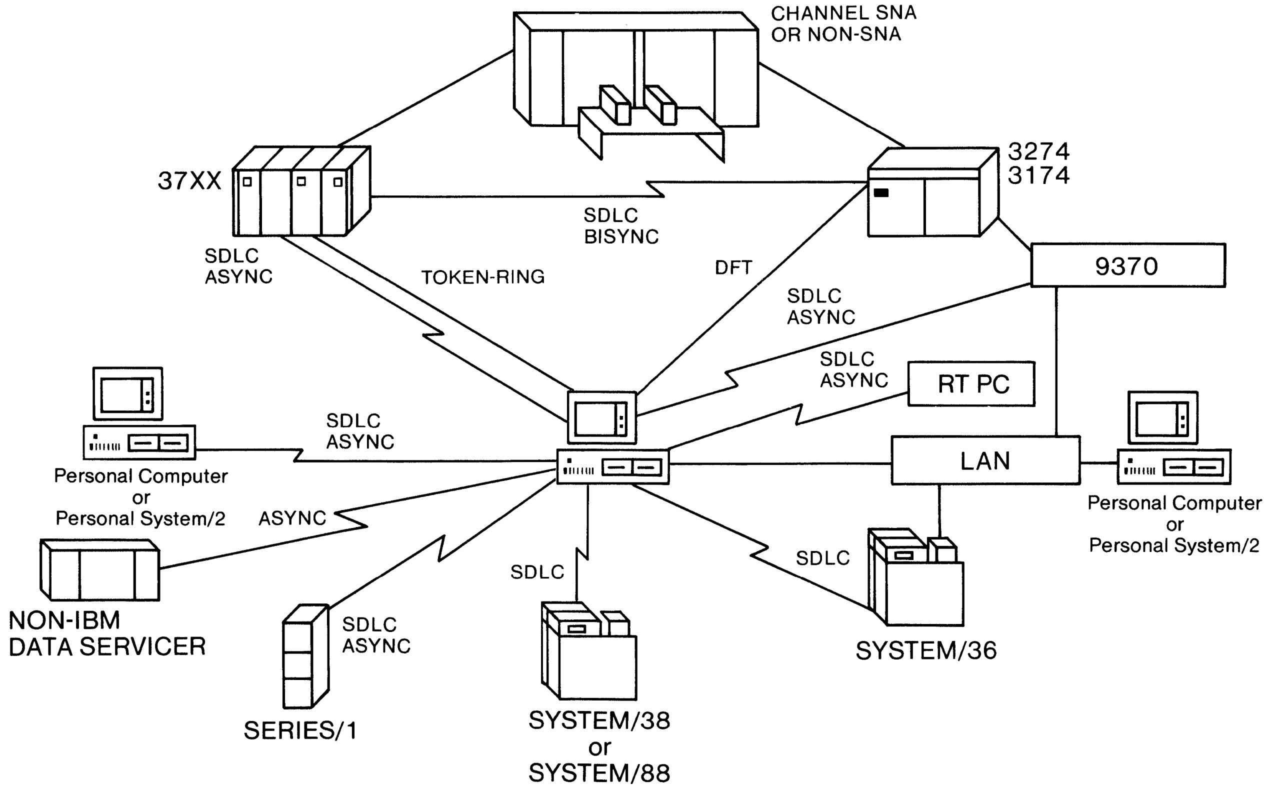


Figure 29. IBM Operating System/2 Connectivity

SUPPORTED SYSTEMS	INTERFACE OR EMULATION	PROTOCOL	FILE TRANSFER	LINK[1]
IBM System/370 Architecture including 9370	APPC	LU6.2	--	SDLC (3720, 3725, 3705, 3726 and 9370 Integrated Controller)
				Token-Ring (3720, 3725, 3726 and 9370 Integrated Controller)
				Token-Ring Using 3174's 3270 Gateway Feature (#3025) for PU2.0
	SRPI	LU2	--	DFT via 3174/3274 (TO SDLC, BSC, Or Channel and 9370 Workstation Controller)
	3270	LU2	3270-PC FILE TRANSFER PROGRAM	
				SDLC (3720, 3725, 3705, 3726, and 9370 Integrated Controller)
				Token-Ring (3720, 3725, 3726, and 9370 Integrated Controller)
				Token-Ring Using 3174's 3270 Gateway Feature (#3025) for PU2.0
	3101,VT100	--	3270-PC FILE TRANSFER PROGRAM	ASYNC[2]
IBM PC and IBM Personal System/2	APPC	LU6.2	--	SDLC, Token-Ring, PC Network
	--	--	XMODEM, PACING+	Async
IBM System/36	APPC	LU6.2	--	SDLC, Token-Ring
IBM System/38 and IBM System/88	APPC	LU6.2	--	SDLC
IBM Series/1	APPC	LU6.2	--	SDLC
	3101	--	--	Async
IBM RT PC	APPC	LU6.2	--	SDLC
	VT100	--	XMODEM	Async
OTHER HOSTS[2]	VT100	--	XMODEM PACING+	Async
	3101*	--	XMODEM PACING+	Async

* Character mode
+Sending an ASCII text file to another system
1, 2, 3 - See notes on following page.

Figure 30. Data Link and Data Stream Matrix

Notes:

1. The IBM Operating System/2 Communications Manager will support combinations of these links subject to the limitations imposed by installed adapters, memory size and processor capacity. All supported SNA links can be shared by applications which may use up to five 3270 display sessions per workstation over IBM SDLC, Token-Ring, and DFT links. A maximum of 255 concurrent SNA LU6.2 sessions are supported over remote SDLC or local Token-Ring LAN links. Asynchronous links are serially reusable.
2. Asynchronous users requiring SNA support must use a protocol converter on the link.
3. Appropriately programmed

Performance Considerations for IBM Operating System/2 Program Developers

Introduction

This is a guide for designers of programs that will use Operating System/2 Standard or Extended Editions.

There are often several choices to be made in many areas when programming to take advantage of a multitasking, virtual memory operating system. This guide suggests ways to consider program and system performance (speed) when making these choices.

None of the rules listed here are hard and fast. Some may seem to penalize single program performance while benefiting overall system performance. When these conflict, remember that better system performance tends to make users happier overall. The assumption here is that the typical Operating System/2 user will be using multiple screen groups (sessions) and will not want a program in one session to monopolize system resources.

Multitasking

For this discussion, "task" is used generically for either "thread" or "process" or both.

1. Use as few processes and threads as possible. Multitasking has the following advantages:

   - Structure with lots of logical separation of function
   - "Asynchronous-ness" (independence)
   - Priority favoritism
   - Memory protection

   Don't use multitasking just for structure; it is an unnecessary expense. Use Call/Return; it is just as structured as multitasking and frequently less complicated.

   If some of your functions can run independently of each other, use multiple processes and/or threads; however use only as many as you have functions that actually run independently and concurrently. This promotes overlapping usage of various computer resources.

   If some functions need to be favored over others (for example, supporting time-sensitive I/O devices), you can isolate them into a separate thread and set its priority high. Don't group performance-sensitive functions that should operate at high priority with low-priority functions. (See also the separate section below on priority.)

   For memory protection requirements, you may want multiple processes in order to have independent descriptor tables. Remember, there is some protection in not exposing the names and segment handles of memory segments and files.

2. Try not to create and destroy tasks frequently. Build your tasking structure at initialization. The task creating services, DOSCreateThread and DOSExecPgm, have a lot of system work to accomplish. They are not nearly as fast as the task shoulder-tapping services such as DOSSemClear matched with DOSSemWait.

3. When you really need independent tasks on a timely basis, try to choose threads rather than processes because DOSCreateThread has much less system work to accomplish than DOSExecPgm.

4. Do as little processing as possible in Device Driver interrupt handlers. Code on the interrupt level is a sort of super-priority "task" that monopolizes the processor at the expense of all the threads and system performance. It also can also cause timeout problems with other devices.

   If your device requires complex interrupt processing, your device driver interrupt handler can issue a ProcRun Device Driver Helper Service (DevHelp) to schedule a piece of the device driver strategy routine to do interrupt processing while not on the hardware interrupt level.

Remember, even if you never create multiple tasks in your program, Operating System/2 will multitask between your program's thread and those of other programs the user is running.

Thread Priority

When Priority Is Not a Concern

Many user environments will not make priority a concern. If no threads alter their priority from the default, the only priority effect the user notices is that the foreground session generally gets priority over any other sessions, and these other sessions will fare evenly with each other.

If your program does not use any time-critical devices (fixed disks, diskettes and printers are generally not time-critical), then priority is of no concern. Your threads will run at regular class and at a nominal level.

When Priority Is a Concern

You may have some time-critical requirements (for example, robotics, process control and communications). In this case use DOSSetPrty to change your thread's priority class to time-critical during important processing or maintain a separate thread that has been initialized to time-critical class, and use this thread for the time-critical function.

If one thread processes both regular and time critical code, and you plan to change the level for the time-critical portion, save the level of the regular portion for restoring after the time-critical portion is finished.

Because your program might be sharing the computer with other programs that have time critical functions, the question of which level to use when performing these functions is important. Some program designers might choose 31 just to be sure. Don't do this arbitrarily because the user may be running another program that is more important than yours. Here is an example:

Criticality	Priority Level
Robotics, Process Control	20 - 31
Communications	10 - 19
Other	0 - 9

Figure 31. Example of Time Critical Requirements

During your program's initialization, you might prompt the user to tell you which priority levels to use for regular and time-critical classes.

Memory Management

1. Use the Memory Segment Suballocation Services, DOSSubAlloc and DOSSubFree, for your data, if you allocate and deallocate a lot. A suballocation request is faster than a segment allocation request. Allocating fewer large segments saves overhead in allocating and freeing memory, swapping segments, etc. Remember, if you have lots of segments, you have to keep juggling the segment registers. This costs processor overhead, and requires more complex code.

2. Don't use the DevHelp lock memory segment call to lock buffers any longer than your hardware needs them locked.

3. Don't use the long-term option in the DevHelp lock memory segment call, unless absolutely necessary. This causes a move to the fixed area (and possibly another move back after you unlock the segment) that costs you unnecessary overhead.

   If you decide that some of your areas really need to stay locked for a long time, then use the long-term option. (Don't leave short term areas locked for a long time because this fragments the movable/swappable section of memory.)

4. Remember that swapping is done by segment and not by suballocated chunk. Organize your segments, if possible, to have the more frequently used data/code all in one (or a few) segments; and the less frequently used data/code all in one (or a few) segments. This way, if swapping is necessary, it will tend to hit only your less frequently used functions. For example, put all your error handling code for all functions in one segment by itself.

Input/Output (I/O)

1. Organize your disk block sizes on 512-byte boundaries, if possible. This eliminates the problem of extra I/O requests to/from the CONFIG.SYS buffers.

2. Use asynchronous I/O if there is anything else your thread can process during I/O. If nothing else can be done in parallel, don't use asynchronous I/O. Operating System/2 will let other threads process while your thread is doing I/O.

3. Since the system creates and destroys a thread for each invocation of the asynchronous forms of

DOSRead and DOSWrite, you may want to use a separate thread and two semaphores to accomplish asynchronous I/O (see Figure 32 on page 56).

4. Use moderately large buffers. For example, reading a lot of data 512 bytes at a time from a disk is much more costly than reading it in 4K chunks. This is because each disk access has its own seek and rotation overhead.
5. Use "read ahead" or even "write behind" double buffering where applicable (see Figure 32 on page 56 or Figure 33 on page 56).

Example Flows for Asynchronous Processing

The following figures are coarse flowcharts for using multiple threads and RAM semaphores to accomplish asynchronous processing. The Set/Wait/Clear logic is identical if system semaphores are used. The general flow is also applicable to other forms of interprocess communication.

Have two doubleword storage locations that are the RAM Semaphores for controlling the I/O. Name one "do1read" (do one read); the other, "done." You will need a location for passing return codes.

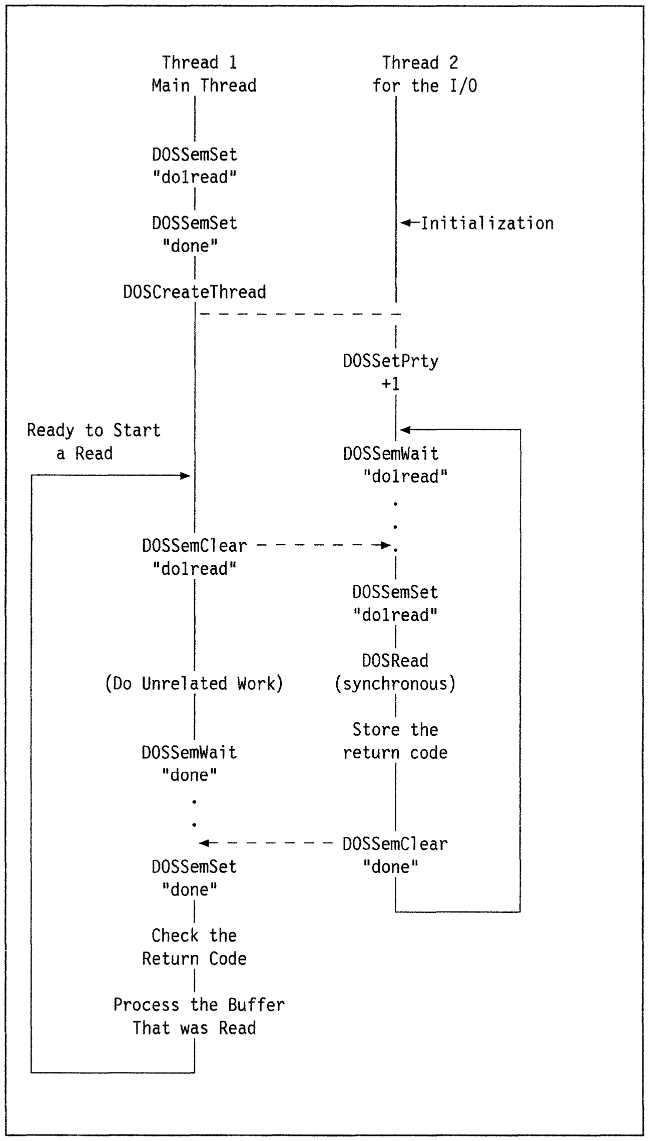


Figure 32. Using Multiple Threads To Do Your Asynchronous I/O.

Have two doubleword storage locations that are the RAM Semaphores for controlling the I/O. Name one "do1read" (do one read); the other, "done." Have two buffers equal to your block size; call these "buf1" and "buf2." Have a general buffer pointer, "userbuf," which GET will flip back and forth between "buf1" and "buf2." You need a location for passing return codes.

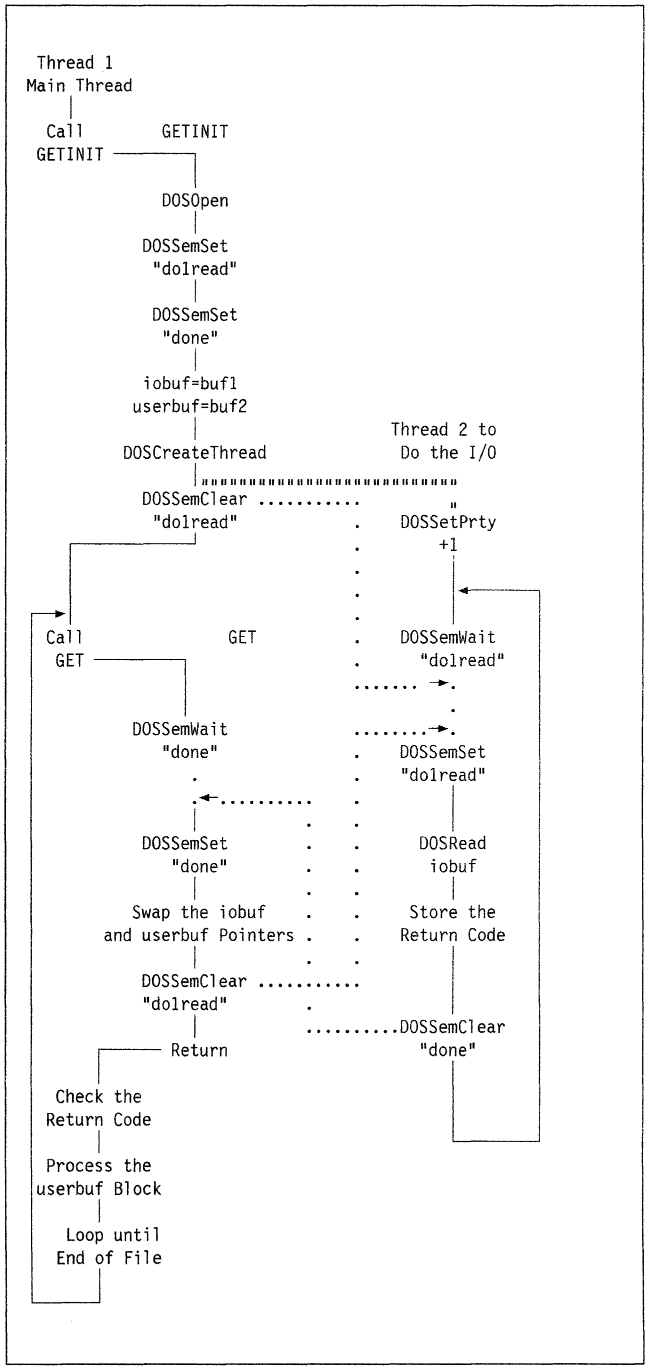


Figure 33. A GET Routine for Double-buffered "read-ahead" Processing of an Input Sequential File.

Have two doubleword storage locations that are the RAM Semaphores for controlling the I/O. Name one "do1write" (do one write); the other, "done." Have two buffers equal to your block size; call these "buf1" and "buf2." Have a general buffer pointer, "userbuf," which PUT will flip back and forth between "buf1" and "buf2." You need a location for passing return codes.

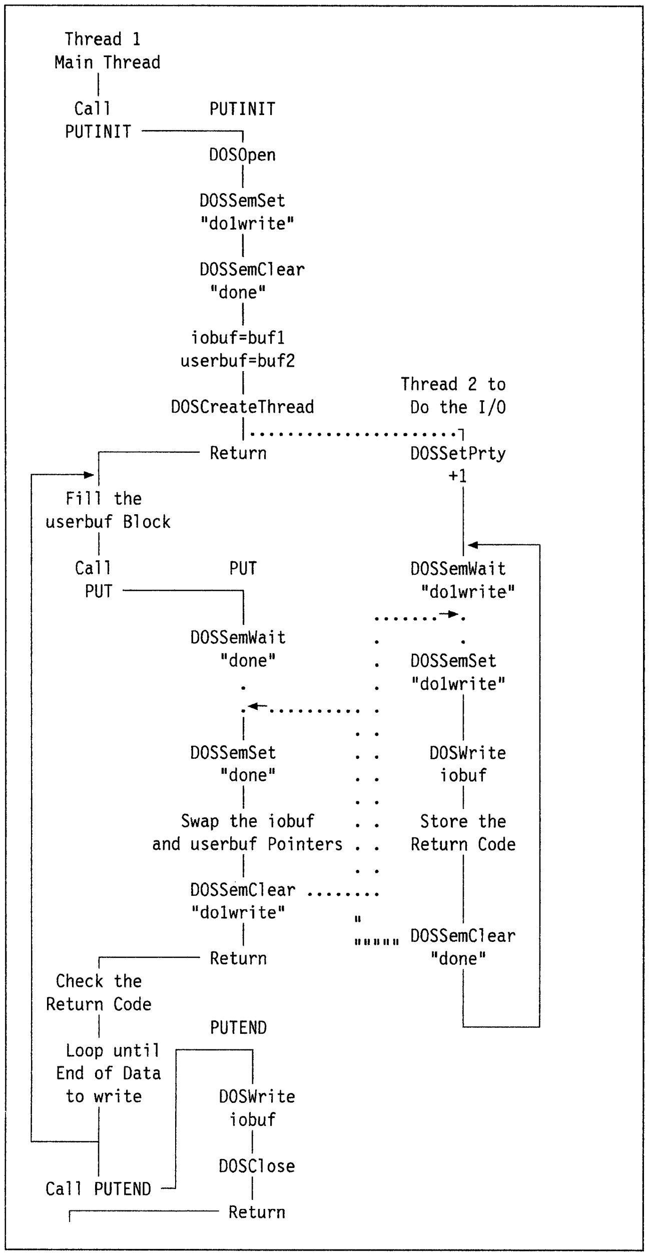


Figure 34. A PUT Routine for Double-Buffered "write-behind" Processing of an Output Sequential File.

IBM Personal System/2 Seminar Proceedings

Publication Number	Vol.	Topic
G360-2653	V5.1	IBM Personal System/2 Model 30 IBM Personal Computer DOS, Version 3.30
G360-2678	V5.2	IBM Personal System/2 Displays and Display Adapters
G360-2637	V5.3	IBM Personal System/2 Models 50, 60, 80 Micro Channel™ Architecture, Hardware Features and Design Considerations
G360-2747	V5.4	IBM Personal System/2 Models 50, 60, 80 VGA, BIOS & Programming Considerations
G360-2756	V5.5	IBM Operating System/2

G360-2756

IBM Corporation
Editor, IBM Personal System/2™ Seminar Proceedings
Internal Zip 3636
Post Office Box 1328
Boca Raton, Fl 33429-1328

IBM®